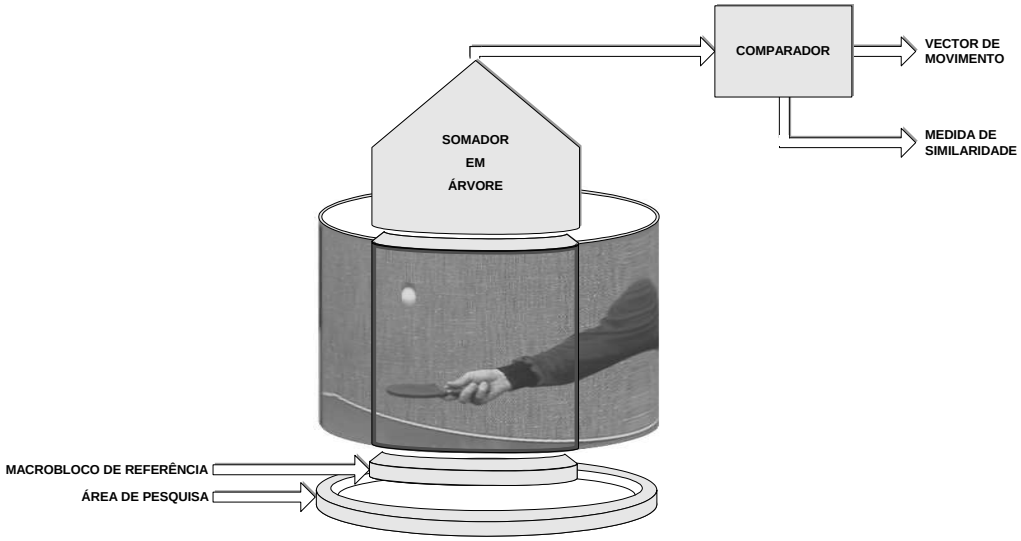
UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO



PROCESSADORES DEDICADOS PARA ESTIMAÇÃO DE MOVIMENTO EM SEQUÊNCIAS DE VÍDEO

Nuno Filipe Valentim Roma
(Licenciado)

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador Científico: Prof. Doutor Leonel Augusto Pires Seabra de Sousa

Presidente do Juri: Prof. Doutor Horácio Cláudio de Campos Neto
Vogais: Prof. Doutor José Alfredo Ribeiro da Silva Matos
Prof. Doutor Leonel Augusto Pires Seabra de Sousa

Lisboa
Janeiro de 2001

Dedico esta tese
aos meus pais
e ao meu irmão.

Resumo

Os sistemas de codificação de vídeo têm vindo a assumir um papel de importância crescente em diversas áreas, nomeadamente, a televisão digital, o videotelefone e a videoconferência, a videovigilância e as aplicações multimédia. O principal objectivo do trabalho apresentado é o desenvolvimento de arquitecturas e circuitos dedicados para estimação de movimento em sequências de vídeo, baseadas no método de emparelhamento de blocos por pesquisa exaustiva.

Com base na análise detalhada dos estimadores de movimento conhecidos, propõem-se novas arquitecturas dedicadas mais eficientes, no que diz respeito ao tempo de processamento e aos recursos requeridos. Para conseguir efectuar o processamento em tempo real, desenvolveram-se, também, unidades aritméticas rápidas, nomeadamente, para as operações de adição, subtracção e cálculo do valor absoluto.

Com base nas arquitecturas e unidades aritméticas propostas, desenvolveu-se um circuito integrado dedicado para estimação de movimento, tendo-se utilizado uma biblioteca de células lógicas baseada no processo tecnológico CMOS - $0,7$ µm. Dos resultados obtidos, concluiu-se que o circuito permite efectuar a estimação de movimento em tempo real sobre sequências de vídeo de resolução elevada; por exemplo, sequências de vídeo no formato 4CIF (576×704) podem ser processadas a um ritmo de 50 imagens por segundo.

Abstract

Video coding systems have been assuming an increasingly important role in several areas, namely digital television, videophone and videoconference, videosurveillance and multimedia applications. The main goal of this research work is the development of architectures and dedicated processors for motion estimation in video sequences, based on the full search block matching algorithm.

By performing a detailed analysis of the well known motion estimators, new and more efficient dedicated architectures are proposed, in what concerns the processing time and required hardware resources. In order to attain real time processing, new fast arithmetic units were also developed to perform a set of basic operations such as addition, subtraction and absolute value computation.

Based on the proposed architectures and arithmetic units, a dedicated integrated circuit for motion estimation was developed, by making use of a standard cell library for a CMOS - $0,7$ μm technology process. From the results, it is possible to conclude that the circuit is able to perform motion estimation in real time on high resolution image sequences; for example, video sequences in the 4CIF format (576×704) can be processed at a rate of about 50 images per second.

Palavras Chave:

- Codificação de Video;
- Estimação de Movimento;
- Emparelhamento de blocos;
- Processadores Especializados;
- Arquitecturas Sistólicas;
- Unidades Aritméticas Rápidas;
- Circuitos Integrados Dedicados para as Aplicações.

Keywords:

- Video Coding;
- Motion Estimation;
- Block Matching;
- Specialised Processors;
- Systolic Architectures;
- Fast Arithmetic Units;
- Application Specific Integrated Circuits.

Agradecimentos

Desejo expressar o meu especial agradecimento ao Prof. Leonel Sousa pela confiança demonstrada ao assumir a orientação desta dissertação, pela inesgotável disponibilidade e pelo incansável apoio e incentivo na realização deste trabalho.

Um agradecimento, em particular, ao Prof. Moisés Piedade pela sua amizade e encorajamento.

Quero também agradecer aos meus colegas do grupo de "Sistemas de Processamento de Sinal" (SIPS) do Instituto de Engenharia de Sistemas e Computadores (INESC-ID) e, em particular, aos Engs. Gonçalo Tavares, Oliver Sinnen e Paulo Lopes, pelo agradável ambiente de trabalho proporcionado.

Expresso ainda um agradecimento muito especial a todos os meus amigos pelo apoio, amizade e interesse ao longo deste trabalho e, em especial, ao Carlos Pinto Coelho pela colaboração e encorajamento constantes.

Por fim, quero expressar um agradecimento muito especial aos meus pais e irmão pela compreensão, paciência e apoio que me concederam ao longo dos últimos meses, que me absorveu grande parte da atenção que lhes era devida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

bit - binary digit

bps - bits per second

pixel - picture element

pps - pixels per second

dpi - dot per inch

fps - frames per second

MOPS - Mega OPerations per Second

GOPS - Giga OPerations per Second

MSB - Most Significant Bit

ASIC - Application Specific Integrated Circuit

VLSI - Very Large Scale Integrated circuits

VHDL - VHSIC[1] Hardware Description Language

VHSIC - Very High Speed Integrated Circuits

CAD - Computer Aided Design

CMOS - Complementary Metal-Oxide Semiconductor

TTL - Transistor-Transistor-Logic

DG - Dependence Graph

SFG - Signal Flow Graph

[1] Do Inglês *Very High Speed Integrated Circuits.*

GOB - Group of Blocks

CIF - Common Intermediate Format

QCIF - Quarter Common Intermediate Format

ITU - International Telecommunication Union

MPEG - Moving Picture Expert Group

JPEG - Joint Photographic Experts Group

MAC - Multiply and Accumulate

SAD - Sum of Absolute Differences

DCT - Discrete Cosine Transform

IDCT - Inverse Discrete Cosine Transform

VLC - Variable Length Code

DPCM - Differential Pulse Code Modulation

RISC - Reduced Instruction Set Computer

DSP - Digital Signal Processor

NC - Number of Cores

FAX - Facsimile

CD - Compact Disk

UMTS - Universal Mobile Telecommunication System

Lista dos Principais Símbolos

$\mathcal{N}$ - conjunto dos números inteiros

$\lfloor \mathbf{x} \rfloor$ - maior inteiro não superior a x

$\lceil \mathbf{x} \rceil$ - menor inteiro não inferior a x

a mod **b** - resto da divisão inteira de a por b

$\max\{d\,(c,l)\}$ - valor máximo de $d\,(c,l)$

$\max_{(u,v)}\{d\,(c,l)\}$ - coordenadas correspondentes ao valor máximo de $d\,(c,l)$

$\min\{d\,(c,l)\}$ - valor mínimo de $d\,(c,l)$

$\min_{(c,l)}\{\,d\,(c,l)\}$ - coordenadas correspondentes ao valor mínimo de $d\,(c,l)$

$\log_a b$ - logaritmo na base a de b

$\hat{w}$ - menor potência inteira de 2 não inferior a w: $\hat{w} = 2^{\lceil \log_2 w \rceil}$

$\#$ - cardinal de um conjunto

C_k^n - combinação de n elementos k a k

$\overline{X}$ - complemento para um do vector de bits X

$\mathcal{O}(f)$ - medida de complexidade da função f

$\odot$ - função de similaridade

$d\,(c,l)$ - medida de semelhança ou de dissemelhança

(c_M, l_M) - coordenadas do macrobloco com maior medida de similaridade

$MV\,(x,y)$ - vector de movimento estimado

Capítulo 1

Introdução

Conteúdo

1.1 Enquadramento e conceitos gerais

O processamento e a transmissão de dados têm assumido, desde há muito, uma importância fundamental na organização e evolução da sociedade em que vivemos. Em particular, os sistemas de comunicação com recurso ao vídeo têm tido, nos últimos anos, uma importância crescente nas mais diversas áreas, nomeadamente, a televisão digital, o videotelefone e a videoconferência, a videovigilância e as modernas aplicações multimédia. No futuro próximo, com o surgir das chamadas novas tecnologias de comunicação, de que é exemplo a introdução da terceira geração de telefones móveis UMTS[1], com capacidades de transmissão de sinais de voz, dados e imagem, prevê-se um aumento significativo da importância dos sistemas de codificação de vídeo na vida quotidiana.

Em termos gerais, define-se um sinal de vídeo como sendo uma série de imagens digitais bidimensionais ordenadas no tempo. Cada uma destas imagens resulta da discretização na amplitude, no espaço e no tempo da informação visual que é, em geral, contínua na fonte. Cada imagem amostrada é constituída por um conjunto de pontos elementares com níveis discretos de intensidade, designados correntemente por pixel[2]. Estes elementos são, então, distribuídos segundo matrizes bidimensionais rectangulares, equidistantes no espaço e ordenadas no tempo (ver figura 1.1). Numa sequência monocromática, cada um dos pixels representa, em geral, o nível de intensidade luminosa num dado ponto da imagem. Na generalidade dos sistemas, a gama de intensidades é representada utilizando uma resolução de 8 bits, pelo que cada pixel pode, assim, assumir um de 256 níveis distintos.

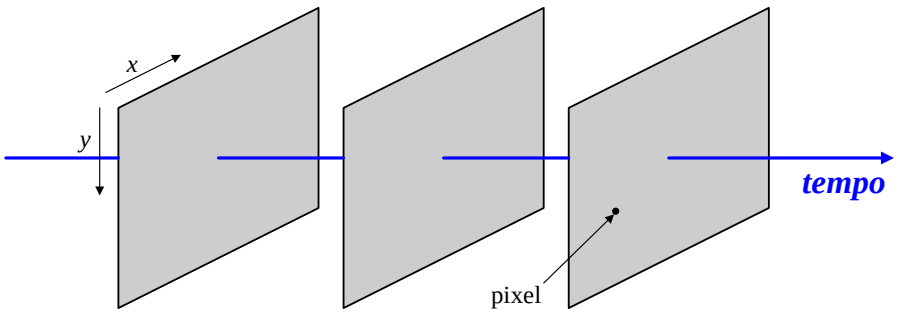


Figura 1.1: Definição do conceito de sequência de vídeo.

No caso de imagens policromáticas, a informação é, em geral, representada por três imagens, correspondendo a cada uma das componentes de cor: R (*red*), G (*green*) e B (*blue*). Como estas componentes apresentam uma elevada correlação entre si, na

[1] Do Inglês *Universal Mobile Telecommunication System*.
[2] Do Inglês *picture element*.

maior parte dos sistemas de codificação de vídeo o espaço de cor RGB é transformado noutro espaço, de componentes menos correlacionadas: uma componente correspondente à luminância (Y) e duas outras correspondentes à cor (C_B e C_R). Estas componentes são, normalmente, processadas como sendo três imagens independentes.

A necessidade de comprimir a representação do sinal de vídeo torna-se evidente sempre que se contabiliza o número de bits necessários para representar o sinal e se compara com a largura de banda dos canais de transmissão ou com o espaço no meio de armazenamento disponíveis. Sem o recurso a técnicas de compressão não existiriam muitas das actuais aplicações de vídeo, conforme se pode constatar com os exemplos seguintes [1, 2]:

- transmissão de imagens digitalizadas por FAX: na maioria dos aparelhos de FAX[3], a informação presente no documento começa por ser adquirida e digitalizada; tipicamente, uma página com o formato A4 (297mm $\times$ 210mm) é digitalizada com uma resolução de 200 dpi[4], resultando num total de 3,87 Mbits de informação; transmitir esta quantidade de informação por um modem de uso geral, a uma taxa de 14,4 kbps[5], demora cerca de 4,37 minutos; utilizando técnicas de compressão, o tempo de transmissão pode ser facilmente reduzido para cerca de 13 segundos;
- gravação de vídeo num CD: considere-se que se pretende armazenar num CD[6], com a capacidade de 650 Mbytes, uma sequência de imagens policromáticas de média resolução, constituídas por três componentes de cor com 512 $\times$ 512 pixels, sendo cada pixel representado por 8 bits. Assumindo um ritmo de cerca de 50 imagens por segundo, verifica-se que apenas seria possível armazenar cerca de 17 segundos de vídeo no CD.
- videotelefone: considere-se uma sequência de imagens constituídas por uma componente de luminância (Y) com 176 $\times$ 144 pixels e duas componentes de crominância (C_B e C_R) com 88 $\times$ 72 pixels (correspondente ao formato QCIF[7]); assumindo que cada pixel é codificado com 8 bits e que se utiliza um ritmo de transmissão de cerca de 30 fps[8], para transmitir a informação em tempo real tornar-se-ia necessário utilizar um canal com um débito binário de cerca de 8,7 Mbps; para tornar possível a sua transmissão num canal com um débito binário de cerca de 28,8 kbps, o sistema de compressão de vídeo terá de conseguir um factor de compressão mínimo de cerca de 317.

[3]Do Inglês *Facsimile*.
[4]Do Inglês *dot per inch*.
[5]Do Inglês *bits per second*.
[6]Do Inglês *Compact Disk*.
[7]Do Inglês *Quarter Common Intermediate Format*.
[8]Do Inglês *frames per second*.

Na tabela 1.1 apresentam-se mais alguns exemplos representativos da necessidade de utilizar técnicas de compressão, para os casos da transmissão de sinais de áudio, imagens e vídeo, apresentando-se as taxas de transmissão requeridas para a transmissão da informação antes e depois de comprimida [2]. O elevado número de bits necessário para representar este tipo de sinais faz com que o seu armazenamento ou a sua transmissão em canais com banda limitada só seja possível através da eliminação de informação redundante.

A tese que se apresenta enquadra-se no desenvolvimento de processadores eficientes para a codificação de vídeo. Nas secções seguintes apresentam-se as características gerais das normas de codificação de vídeo, referindo-se os principais objectivos do presente trabalho.

	Taxa de transmissão	
Aplicação	**Não comprimida**	**Comprimida**
Voz (8 kamostras/s, 8 bits/amostra)	64 kbps	2-4 kbps
Videotelefone (10 fps, 176 × 120 pixels, 8 bits/amostra)	5,07 Mbps	8-16 kbps
Áudio (8 kamostras/s, 8 bits/amostra)	64 kbps	16-64 kbps
Videoconferência (15 fps, 352 × 240 pixels, 8 bits/amostra)	30,41 Mbps	64-768 kbps
Áudio digital (stereo) (44,1 kamostras/s, 16 bits/amostra)	1,5 Mbps	128-1500 kbps
Vídeo digital em CD-ROM (30 fps, 352 × 240 pixels, 8 bits/amostra)	60,83 Mbps	1,5-4 Mbps
Difusão de vídeo digital (30 fps, 720 × 480 pixels, 8 bits/amostra)	248,83 Mbps	3-8 Mbps
Televisão de alta definição (59,94 fps, 1280 × 720 pixels, 8 bits/amostra)	1,33 Gbps	20 Mbps

Tabela 1.1: Débitos binários requeridos para a transmissão de sinais de voz, dados e vídeo.

1.2 Codificação de vídeo

Na última década, assistiu-se ao estabelecimento de várias recomendações para compressão de vídeo, tais como as H.261 [3], H.262 [4], H.263 [5] da ITU-T e as MPEG-1 [6], MPEG-2 [7] e MPEG-4. Os elevados factores de compressão oferecidos por estas normas são conseguidos, essencialmente, à custa das seguintes operações:

- redução do ritmo de amostragem, tanto no domínio do espaço como do tempo, das componentes de luminância e crominância;
- codificação ao nível do bloco utilizando a transformada discreta de coseno (DCT[9]);
- utilização de técnicas de predição através de compensação de movimento e de codificação das diferenças;
- codificação entrópica dos símbolos binários[10] gerados pelo codificador.

De seguida, passar-se-á a descrever, em linhas gerais, o princípio de funcionamento destas normas de codificação de vídeo.

1.2.1 Formato de entrada

Na maioria das normas de codificação de vídeo, a informação correspondente às três componentes YC_BC_R da imagem é organizada segundo uma estrutura em blocos. Em geral, define-se o macrobloco como sendo a unidade de informação mínima, que é constituída por 4 blocos de 8×8 pixels da componente Y de luminância, sobrepostos com 1 bloco de 8×8 pixels da componente C_B e com outro bloco da 8×8 pixels da componente C_R, segundo a disposição ilustrada na figura 1.2. As duas componentes de cor são subamostradas, garantindo-se uma sobreposição espacial das três componentes. Assim, cada imagem é constituída por uma série de macroblocos, justapostos dois a dois da esquerda para a direita e de cima para baixo. Define-se, ainda, a unidade de GOB[11], constituído por um conjunto de k linhas de macroblocos. Este tipo de estrutura implica que as dimensões horizontal e vertical da imagem sejam, em geral, múltiplas de 16.

1.2.2 Tipos de imagem

Em geral, nas normas de codificação de vídeo as imagens a codificar são classificadas em dois tipos: imagens do tipo INTRA e imagens do tipo INTER. As imagens do tipo INTRA

[9] Do Inglês *Discrete Cosine Transform.*
[10] Do Inglês *bit stream*
[11] Do Inglês *Group of Blocks.*

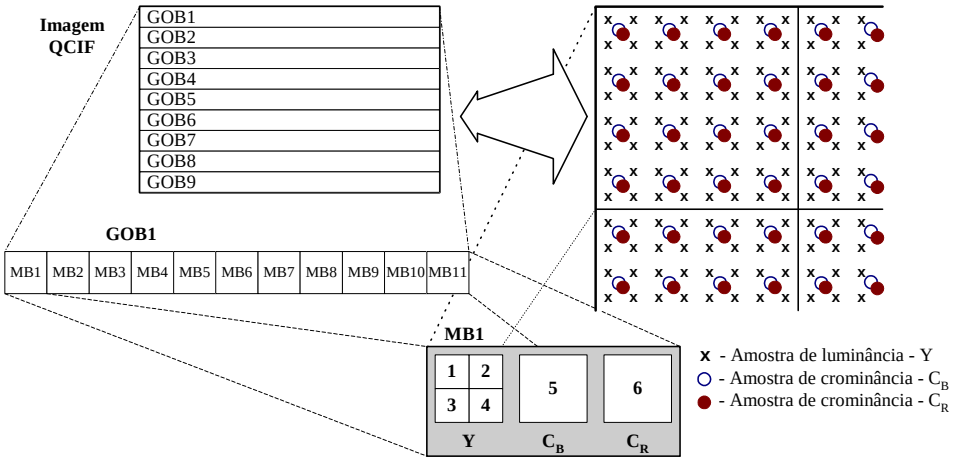


Figura 1.2: Estrutura de blocos da imagem.

(tipo I) são codificadas de uma forma independente no tempo, isto é, a sua codificação não apresenta qualquer dependência com outras imagens da sequência de vídeo, explorando-se, apenas, a redundância espacial. Embora permitam uma descodificação individual, são codificadas com taxas de compressão moderadas, à semelhança do que se verifica com a codificação de imagens isoladas na norma JPEG[12] [8]. Em contraste, as imagens do tipo INTER (tipo P) utilizam uma codificação baseada nas imagens anteriores do tipo INTRA ou do tipo INTER, através da utilização de técnicas de predição para reduzir as redundâncias temporais (codificação diferencial com estimação de movimento). Este tipo de imagens permite obter maiores taxas de compressão, pois utilizam as imagens vizinhas como pontos de partida para a codificação. Existe, ainda, a possibilidade de utilizar técnicas de predição bidireccional, através da utilização de imagens (do tipo I ou do tipo P) anteriores ou posteriores como referência para a predição da imagem actual (imagens do tipo B). As imagens do tipo B não são utilizadas como preditores de outras imagens do tipo B ou do tipo P, permitindo, no entanto, obter maiores taxas de compressão.

Na figura 1.3 apresenta-se a relação entre estes três tipos de imagens numa sequência de vídeo composta por oito imagens p_1 a p_8. As imagens p_1 e p_8 são do tipo I, enquanto que as imagens p_4 e p_7 são do tipo P. As restantes imagens são do tipo B. Neste exemplo, a imagem p_3 é codificada utilizando técnicas de predição baseadas na codificação diferencial com estimação de movimento utilizando as imagens p_1 e p_4 como referência.

[12] Do Inglês *Joint Photographic Experts Group*.

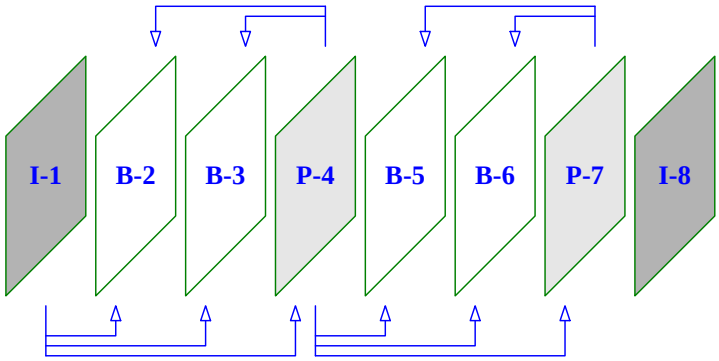


Figura 1.3: Relação entre imagens (p) do tipo I, do tipo P e do tipo B.

1.2.3 Codificador de vídeo

Em geral, as normas de codificação de vídeo não definem um processo de codificação específico [3, 4, 5, 6, 7]. Limitam-se, apenas, a definir a sintaxe da trama de bits e o processo de descodificação. Na figura 1.4 ilustra-se o diagrama de blocos de um codificador de vídeo típico.

Pré-processamento

A codificação de vídeo começa, em geral, com uma fase de pré-processamento. Nesta fase, procede-se à conversão das componentes de cor YC_BC_R, à adaptação do formato de imagem, e às operações de filtragem e de subamostragem. Nenhuma destas operações é, no entanto, especificada pelas normas de codificação.

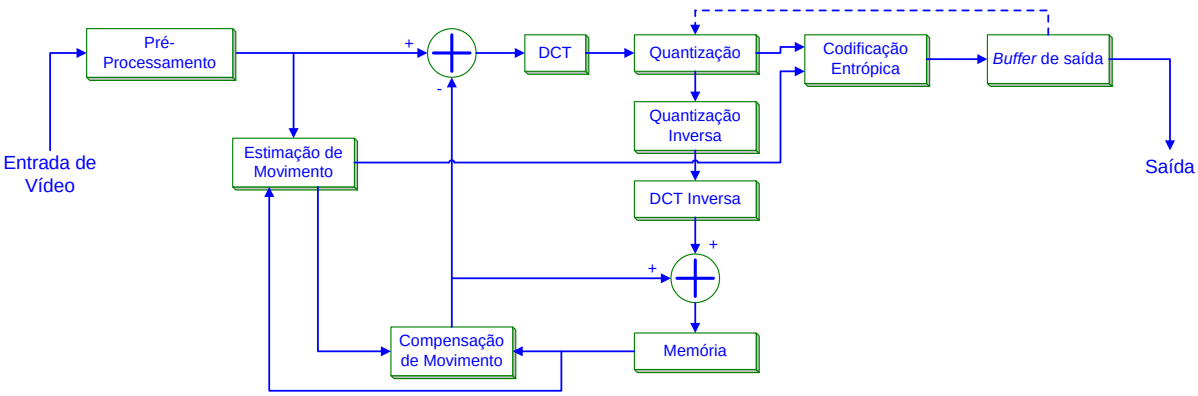


Figura 1.4: Diagrama de blocos de um sistema de codificação de vídeo.

Codificação

Efectuado o pré-processamento, o codificador procede à classificação da imagem actual segundo um dos três tipos possíveis: I, P ou B. Conforme se referiu, as imagens do tipo INTRA não são objecto de codificação preditiva (codificação das diferenças com estimação de movimento). Cada macrobloco é codificado, aplicando a transformada discreta de coseno (DCT), e os coeficientes da transformada são quantizados e codificados, utilizando um codificador do tipo entrópico com palavras de comprimento variável (VLC[13]). O processamento é efectuado ao nível dos blocos de 8×8 pixels. As palavras assim obtidas são então enviadas para o *buffer* de saída. Em aplicações onde se requer um ritmo binário constante utiliza-se, ainda, um controlador de *buffer* de saída, para ajustar os passos de quantização utilizados em função do débito binário.

Os coeficientes da transformada depois de quantizados são novamente desquantizados (Q^{-1}), calculando-se a transformada inversa de coseno (IDCT[14]). Converte-se, assim, a sua representação para o domínio da imagem. Na realidade, estas duas operações desempenham um papel semelhante ao do descodificador, garantindo, assim, a existência no codificador de uma cópia da imagem descodificada. Esta cópia é então armazenada numa memória local para usar na codificação do tipo preditivo das imagens seguintes. Este bloco do codificador permite, assim, que o codificador utilize exactamente as imagens vistas no descodificador, minimizando a acumulação de erros na codificação preditiva. Como a codificação entrópica não apresenta perdas, não é necessário incluir a unidade de descodificação na malha de retroacção.

No caso da codificação de imagens classificadas do tipo P ou B, o codificador efectua, como se referiu, uma codificação do tipo preditiva. Em vez de codificar directamente os macroblocos, são processados os dados correspondentes ao erro de predição. No caso de imagens do tipo P, cada macrobloco da imagem actual começa por ser processado pelo estimador de movimento, obtendo-se as coordenadas do macrobloco da área de pesquisa que melhor se assemelha com o macrobloco actual. Estes dois macroblocos são, então, subtraídos e a sua diferença é aplicada ao bloco da DCT.

No caso de imagens do tipo B, o processo de estimação é efectuado duas vezes: uma para a imagem anterior e outra para a imagem seguinte, obtendo-se, assim, dois vectores de movimento. O erro de predição poderá então ser obtido a partir de qualquer um destes vectores, ou a partir do vector resultante da sua média, sendo este método designado por codificação interpolativa entre imagens. O erro de predição é então codificado, bloco a bloco, com base na DCT.

[13] Do Inglês *Variable Length Code.*
[14] Do Inglês *Inverse Discrete Cosine Transform.*

Devido à utilização de predição bidireccional, torna-se por vezes necessário efectuar a reordenação das imagens codificadas antes de serem enviadas para o descodificador. Considerando, por exemplo, a sequência ilustrada na figura 1.3, como a imagem p_2 é uma imagem do tipo B, que depende das imagens p_1 e p_4, esta pode apenas ser enviada depois da imagem p_4. Assim, após se reordenarem todas as imagens, a ordem correcta da sequência a enviar será p_1, p_4, p_2, p_3, p_7, p_5, p_6 e p_8. Esta sequência garantirá, ao descodificador, toda a informação necessária para o processamento das imagens codificadas recebidas na forma de trama de bits. Como será de esperar, o descodificador terá, também ele, de reordenar as imagens descodificadas, para que sejam visualizadas na ordem correcta.

1.2.4 Descodificador de vídeo

O diagrama de blocos do descodificador de vídeo encontra-se ilustrado na figura 1.5. O circuito descodificador, conforme foi anteriormente explicado, é bastante semelhante ao circuito da malha de retroacção do codificador. Assim, as imagens de entrada começam por ser descodificadas através de um descodificador entrópico e classificadas em imagens do tipo I, P ou B, com base na informação presente no cabeçalho da trama recebida. Para cada macrobloco procede-se, então, à desquantização dos coeficientes que são, em seguida, convertidos para o espaço da imagem utilizando a transformada inversa de coseno (IDCT).

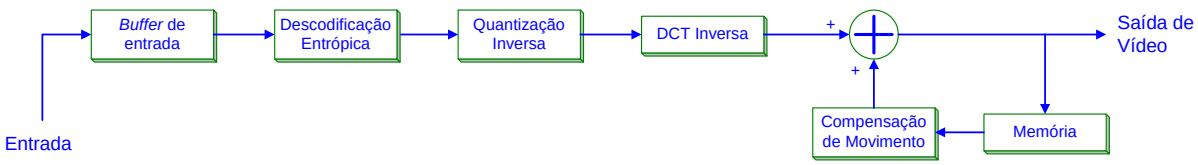


Figura 1.5: Diagrama de blocos de um sistema de descodificação de vídeo.

De seguida, no caso de imagens do tipo I, o descodificador limita-se a colocar o resultado da transformada inversa no *buffer* de saída e a armazená-la na sua memória local, para utilizar como referência na descodificação de imagens do tipo P ou B. No caso de imagens com codificação preditiva, os macroblocos são entregues ao bloco de compensação de movimento após o cálculo da IDCT. Neste bloco, efectua-se a soma dos pixels da área da imagem anterior, definida pelo vector de movimento recebido, com o resultado da transformada inversa. A imagem reconstruída é, então, enviada para o *buffer* de saída e armazenada na memória local.

1.2.5 Requisitos computacionais para codificar vídeo

Uma análise da complexidade dos sistemas descritos permite-nos concluir que, se por um lado o módulo descodificador pode ser facilmente implementado por *software* em qualquer computador de uso geral, a implementação do módulo codificador dificilmente permitirá ritmos de codificação em tempo real sem o recurso a sistemas de processamento dedicados.

Nas tabelas 1.2(a) e 1.2(b) apresenta-se uma estimativa correspondente aos ritmos de processamento requeridos para a codificação e a descodificação de imagens em tempo real [2]. Os valores apresentados referem-se à norma de codificação de vídeo H.261 [3] e a imagens no formato CIF[15] (288×352), codificadas e descodificadas a um ritmo de 30 fps.

Compressão	**MOPS**[16]
Conversão RGB para YC_BC_R	27
Estimação de movimento	608
Codificação Intra/Inter	40
Filtro de malha	55
Predição de pixel	18
DCT a 2 dimensões	60
Quantização em *zig-zag*	44
Codificação entrópica	17
Reconstrução da imagem	99
TOTAL	**968**

(a) Codificação.

Descompressão	**MOPS**
Descodificador entrópico	17
Quantização inversa	9
DCT inversa	60
Filtro de malha	55
Predição	30
Conversão YC_BC_R para RGB	27
TOTAL	**198**

(b) Descodificação.

Tabela 1.2: Ritmos de processamento para a codificação e descodificação de vídeo.

A partir da tabela 1.2(b) é possível constatar que o processo de descodificação requer, aproximadamente, um total de 200 MOPS. Este nível de processamento é hoje facilmente conseguido através do uso de processadores do tipo RISC[17] ou DSP[18] de uso geral. Contudo, o mesmo já não se pode concluir em relação aos valores correspondentes ao sistema de codificação, que requer um ritmo de aproximadamente 1000 MOPS. Destas, cerca de $62,8\%$ são requeridas pela unidade de estimação de movimento. Estes ritmos de processamento encontram-se ainda longe das capacidades dos actuais processadores de uso geral, sendo, por isso, inadequados para efectuar o processamento em tempo real.

[15] Do Inglês *Common Intermediate Format*.
[17] Do Inglês *Reduced Instruction Set Computer*.
[18] Do Inglês *Digital Signal Processor*.

Uma forma de resolver esta questão consiste na distribuição do peso computacional por vários co-processadores com funcionamento concorrente. Num sistema de codificação de vídeo típico, os vários blocos funcionais são alocados aos diferentes co-processadores da seguinte forma:

- módulo de cálculo da DCT e quantização (Q);
- módulo de cálculo da quantização inversa (Q^{-1}) e da IDCT;
- módulo de estimação e compensação de movimento.

De entre estes, o co-processador responsável pela estimação de movimento é, conforme se viu, aquele que apresenta maiores requisitos computacionais e que, por isso, mais atenção tem merecido por parte da comunidade científica.

1.3 Objectivos

Nos últimos anos, diversos processadores para estimação de movimento têm sido propostos por diversos autores. Contudo, estes processadores utilizam, geralmente, procedimentos de estimação de movimento simplificados, que conduzem a taxas de compressão reduzidas. Há projectos de arquitecturas para estimação de movimento que produzem resultados óptimos, mas traduzem-se em processadores com desempenho ou eficiência insuficientes.

Com o aparecimento das recentes normas de codificação de vídeo, novos mercados se abrem para as aplicações multimédia e difusão de vídeo em geral. Para além disso, com a evolução ao nível da integração dos circuitos, torna-se cada vez mais viável comercialmente a produção e o desenvolvimento dos circuitos para estimação de movimento.

Assim, os principais objectivos deste trabalho consistem em:

- definir a arquitectura para um processador dedicado para estimação de movimento que apresente níveis elevados de eficiência e que possa ser facilmente integrado num sistema de codificação de vídeo com funcionamento em tempo real;
- desenvolver unidades de cálculo rápido para as diversas operações envolvidas na estimação de movimento, para alcançar os níveis de desempenho requeridos;
- desenvolver um processador especializado a ser implementado num circuito integrado dedicado (ASIC[19]); para a concretização deste objectivo, procede-se à descrição dos circuitos utilizando a linguagem de descrição de *hardware* VHDL[20].

[19] Do Inglês *Application Specific Integrated Circuit.*
[20] Do Inglês *VHSIC Hardware Description Language.*

1.4 Organização da tese

A tese encontra-se organizada em oito capítulos, incluindo o primeiro e o último capítulos, de introdução e conclusão, respectivamente. Os quatro primeiros são capítulos gerais sobre estimação de movimento, apresentando os conceitos necessários ao desenvolvimento do trabalho e os principais algoritmos e arquitecturas propostos para se estimar movimento em sequências de vídeo. Nos três capítulos seguintes apresenta-se o trabalho realizado para o desenvolvimento de um processador dedicado para a estimação de movimento de elevado desempenho, concluindo-se a apresentação com um capítulo final, onde são apresentados os resultados experimentais obtidos.

No capítulo 2 efectua-se uma análise sucinta das técnicas de codificação de vídeo preditivas, começando-se por introduzir as técnicas baseadas na exploração das redundâncias temporais das imagens a codificar. De entre os métodos para estimação e compensação de movimento, descreve-se, em particular, o método de estimação por emparelhamento de blocos e apresentam-se as principais medidas de similaridade propostas ao longo dos anos.

No capítulo 3, procede-se à apresentação dos principais algoritmos para estimação de movimento, focando-se as principais vantagens e inconvenientes dos algoritmos de pesquisa óptimos face aos algoritmos sub-óptimos, tanto em termos da precisão dos resultados obtidos como do número de operações requeridas.

No capítulo 4 efectua-se uma análise detalhada das principais arquitecturas sistólicas de processadores dedicados para estimação de movimento propostas ao longo dos últimos anos. Faz-se uma descrição e analisam-se as características de vários tipos de arquitecturas obtidos a partir de um grafo de dependências genérico, representado num espaço tridimensional, nomeadamente, arquitecturas bidimensionais (AB2, Vos, Hsieh, AS2, Tipo 2 e Chang) e unidimensionais (AB1 e AS1). Apresenta-se uma análise comparativa das várias estruturas em termos do número de elementos processadores requeridos ($\propto$ *área*), e do número de ciclos de relógio necessário para efectuar o processamento ($\propto$ *tempo*). Esta análise constituiu o ponto de partida para a investigação realizada no âmbito desta tese.

No capítulo 5 é apresentada a nova arquitectura para estimação de movimento proposta nesta tese. Esta arquitectura, baseada na arquitectura bidimensional proposta por *Vos* [9], permite obter processadores mais eficientes, no que diz respeito ao tempo de processamento e à área de circuito requerida. Apresenta-se, ainda neste capítulo, um conjunto de arquitecturas que podem ser derivadas a partir da arquitectura base, analisando-se, comparativamente, o seu desempenho.

No capítulo 6 apresenta-se um estudo efectuado sobre unidades para cálculo rápido de operações aritméticas do tipo soma. Estas unidades são importantes para o cálculo rápido de diferenças, de valores absolutos e de somas, necessárias no processo de estimação de movimento. Consideraram-se estruturas para soma do tipo *ripple-carry* (que serviu como termo de comparação), *carry-lookahead*, baseadas em aritmética redundante e do tipo *prefix-adder*. Compara-se o desempenho das diferentes unidades e sugerem-se melhorias, tendo em vista a sua aplicação no circuito de estimação de movimento.

No capítulo 7 descreve-se o processador desenvolvido, com vista à sua implementação num circuito integrado dedicado (ASIC). Descrevem-se a estrutura dos elementos processadores activos e passivos, a estrutura da matriz de processamento, o somador em árvore, o comparador e, ainda, a unidade de controlo do processador.

No capítulo 8 apresentam-se as características do processador implementado num ASIC. Efectua-se uma breve descrição da tecnologia utilizada e das ferramentas de CAD[21] aplicadas no desenvolvimento do circuito. Apresentam-se os resultados experimentais obtidos, tecendo-se alguns comentários críticos. Apresentam-se, por fim, as conclusões mais significativas obtidas ao longo deste trabalho, sublinhando o facto do processador obtido ter capacidade para efectuar estimação de movimento em tempo real, para a maioria dos formatos de imagem de média e elevada dimensão.

1.5 Contribuições originais

Nesta dissertação procedeu-se a investigação com o objectivo de desenvolver um processador de estimação de movimento eficiente. Na opinião do autor, as contribuições mais importantes a salientar deste trabalho são as apresentadas de seguida:

- proposta de uma nova arquitectura para um processador de estimação de movimento que permite obter níveis de eficiência superiores às dos processadores já propostos; esta arquitectura introduz inovações ao nível da topologia do processador, do seu esquema de processamento, do sistema de transferência dos dados para os elementos processadores e do aumento do nível de paralelismo através do processamento simultâneo de vários macroblocos candidatos;
- proposta de unidades aritméticas eficientes para o cálculo das operações requeridas para a estimação de movimento, nomeadamente, de operações de soma, de subtracção e cálculo do valor absoluto; esta tarefa envolveu o estudo detalhado de unidades rápidas de somadores do tipo *carry-lookahead*, baseadas em aritmética redundante e do tipo *prefix-adder*.

[21] Do Inglês *Computer Aided Design*.

- desenvolvimento de um processador eficiente para estimação de movimento, implementado num circuito integrado dedicado e descrito na linguagem VHDL; o elevado desempenho deste processador deve-se, fundamentalmente, ao facto de se ter, pela primeira vez, considerado de uma forma integrada a investigação da arquitectura, do esquema de processamento e das unidades aritméticas que o constituem.

Capítulo 2

Compensação de movimento em sequências de vídeo

Conteúdo

2.1 Introdução

De acordo com a teoria dos sistemas de processamento de sinal, verifica-se que, para amostras de entrada muito correlacionadas, é possível obter uma estimativa relativamente precisa de uma amostra com base em amostras anteriores, isto é, as amostras anteriores podem ser utilizadas para obter uma predição da amostra actual. Esta propriedade permite definir um sistema de codificação preditivo, tal como se ilustra no diagrama de blocos da figura 2.1, onde a diferença entre uma dada amostra (x) e a sua predição ($\tilde{x}$) é codificada e transmitida.

A forma de codificação preditiva mais aplicada em processamento de imagem é designada por DPCM[1]. No caso particular da aplicação destes métodos à compressão de imagem, são utilizadas vizinhanças dos pixels para predição, explorando-se, assim, a correlação espacial para comprimir a informação. No caso de sequências de vídeo, verifica-se também que tramas consecutivas contêm, em geral, muitas semelhanças. Esta redundância temporal pode, também, ser explorada através de uma codificação do tipo DPCM entre tramas consecutivas. Neste caso, as tramas anteriores são usadas para predição no sistema de codificação, efectuando-se compensação de movimento entre tramas consecutivas.

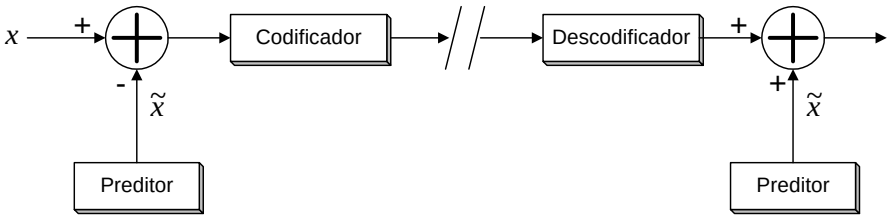


Figura 2.1: Diagrama de blocos de um sistema de codificação preditivo.

Neste capítulo descrevem-se os conceitos gerais da técnica de predição baseada na estimação de movimento focando-se, em particular, o método de estimação baseado na pesquisa exaustiva por emparelhamento de blocos com precisão inteira e de meio pixel.

2.2 Compensação de movimento

A predição baseada na compensação de movimento inclui duas fases distintas: a estimação dos vectores de movimento e a compensação de movimento propriamente dita, sendo a primeira computacionalmente mais exigente. Para este facto, concorrem os seguintes factores:

[1] Do Inglês *Differential Pulse Code Modulation*.

- o espaço de soluções tem uma dimensão bastante elevada;
- os algoritmos de procura exaustiva envolvem um elevado número de cálculos;
- a precisão e a correcção do vector óptimo obtido depende das característica das imagens a processar, tais como: o nível de textura, a luminância, o contraste e o ruído associado ao sistema de aquisição; este factor pode ser particularmente determinante em imagens com uma quantidade de informação reduzida, como por exemplo, imagens com extensas áreas de brilho constante, com abundância de contornos exclusivamente horizontais ou verticais, ou contendo padrões iguais e repetitivos.
- a existência de oclusões totais ou parciais de objectos nas imagens ao longo do tempo, dando origem a problemas de difícil solução.

Assim, na estimação de movimento, tramas consecutivas de uma sequência de vídeo são processadas para estimar os vectores de movimento (ou de deslocamento) dos pixels ou blocos de pixels. Os vectores de movimento calculados e a trama diferença, obtida com base na trama de predição resultante da compensação de movimento e a trama original, são codificadas e transmitidas.

A maioria das normas de codificação de vídeo, tal como as normas H.263 [5] e MPEG[2] [6], limitam-se a estabelecer a arquitectura geral do codificador e a topologia das várias estruturas elementares que se obtêm por decomposição da imagem a processar. Em geral, esta topologia resulta numa segmentação das imagens segundo uma estrutura regular, formada por vários blocos rectangulares de dimensão $N \times N$, designados por *Macroblocos*. Tipicamente, o valor de N usado na maioria das aplicações correntes varia entre 4 e 16. Neste trabalho, decidiu-se adoptar as recomendações da norma H.263 [5] e MPEG [6], consideram-se macroblocos de 16×16 pixels. Desta forma, seguindo as recomendações referentes à sintaxe e à semântica indicadas pelas normas, garante-se que a descodificação da trama de bits transmitida produz sempre o mesmo resultado, qualquer que seja o sistema de codificação utilizado. Contudo, o método de estimação dos vectores de movimento a adoptar não é, em geral, normalizado, sendo deixado como parâmetro da implementação.

Como a estimação de movimento é um factor determinante na eficiência do sistema de codificação, nomeadamente, no que diz respeito ao factor de compressão e à complexidade e capacidade de cálculo do sistema, dedicar-se-á o próximo capítulo apenas à análise dos diversos algoritmos investigados para estimação de movimento.

[2]Do Inglês *Moving Picture Expert Group*.

2.3 Emparelhamento de blocos

A técnica de emparelhamento de blocos é a mais utilizada para a estimação dos vectores de movimento devido, essencialmente, à simplicidade do processamento e à regularidade dos dados. Nesta técnica, cada imagem é segmentada em macroblocos de dimensão $N \times N$ e assume-se que todos os pixels de um dado macrobloco sofrem o mesmo movimento. É usual designar um bloco da imagem actual por *Macrobloco Actual* ou *Macrobloco de Referência* – $R(u, v)$ e um bloco da imagem anterior por *Macrobloco Anterior* ou *Macrobloco de Pesquisa* – $S(u, v)$.

Com a estimação do vector de movimento, para cada macrobloco da imagem actual pretende-se encontrar na imagem anterior o macrobloco que melhor se assemelha com o macrobloco actual, com base numa determinada medida de erro. Esta procura é feita numa determinada área de pesquisa definida na vizinhança do macrobloco da imagem anterior com as mesmas coordenadas do macrobloco actual, isto é, não deslocado. Esta área de pesquisa estende-se em ambas as direcções numa extensão de $\pm p$ pixels relativamente à posição do macrobloco de referência, correspondendo a uma área total de $(2p + N)(2p + N) = (N + 2p)^2$ pixels (ver figura 2.2). Durante o processo de procura, o macrobloco de referência é deslocado dentro desta área de pesquisa para as coordenadas (c, l). Para cada um dos deslocamentos é calculada a medida de similaridade, obtendo-se, assim, uma matriz com os vários valores de semelhança $d(c, l)$. Os índices (c_M, l_M) da matriz correspondentes ao máximo valor de semelhança da função de similaridade considerada $(\odot)$ são as coordenadas do vector de movimento estimado. Após se terem processado todos os macroblocos que constituem a imagem actual, obtém-se uma

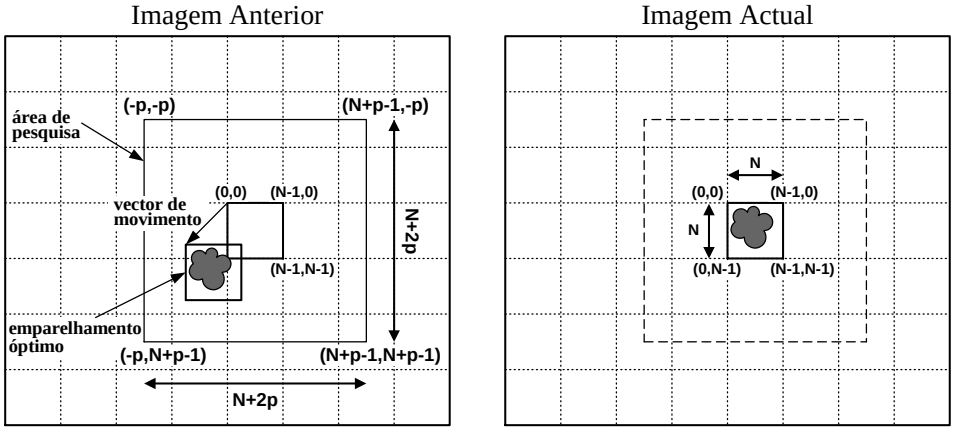


Figura 2.2: Algoritmo de emparelhamento de blocos para estimação dos vectores de movimento.

matriz $MV(x,y)$ com os vectores de movimento estimados para a imagem actual (ver equação 2.3).

$$d(c,l) = \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} R(u,v) \odot S(c+u, l+v) \tag{2.1}$$

$$d(c_M, l_M) = \arg\max_{(c,l)} \{ d(c,l) \ : \ -p \leq c,l < p \} \tag{2.2}$$

$$MV(x,y) = \{ d_{xy}(c_M, l_M) \ : \ 1 \leq x < N_h \, ; \ 1 \leq y < N_v \} \tag{2.3}$$

Convém notar que, no caso mais geral de estimação de movimento, a geometria de um dado objecto na imagem de referência não é, necessariamente, igual à geometria do mesmo objecto na imagem anterior. Na realidade, o objecto representado na sequência de imagens pode sofrer movimentos mais complexos do que simples movimentos de translação, tais como rotações, ampliações, reduções, etc. Estes tipos de movimentos não são convenientemente modelados por vectores de movimento. Contudo, a generalidade das normas de codificação de vídeo assume, apenas, o modelo correspondente a movimentos curtos de translação dos objectos, pelo que, na prática, a estimação efectuada segundo um sistema de coordenadas rectangulares por emparelhamento de blocos é suficiente.

2.3.1 Estimação de movimento com precisão de meio pixel

A descrição feita nas secções anteriores incidiu no processo de estimação de movimento com base num espaço discreto de pixels que constituem a imagem a codificar. Contudo, o movimento dos blocos que se verifica entre imagens consecutivas não é, necessariamente, descrito de uma forma precisa através de uma escala discretizada. De facto, para além de não existirem restrições quanto à natureza dos deslocamentos, estes não são, em geral, correlacionados com a grelha discreta utilizada. Assim, será natural esperar obter resultados mais precisos se forem consideradas resoluções mais finas nos espaços de procura.

Na sequência desta observação, algumas normas de codificação de vídeo, tal como o MPEG [6] e o H.263 [5], prevêem a utilização de uma precisão de meio pixel para a compensação de movimento. Torna-se assim possível obter valores para as componentes vertical e horizontal (c,l) do vector de movimento correspondentes a múltiplos de 0,5. Neste caso, o valor dos pixels $S(c+u, l+v)$ são obtidos por intermédio de interpolação, normalmente efectuada com uma função linear, dos valores correspondentes aos pixels vizinhos com coordenadas inteiras.

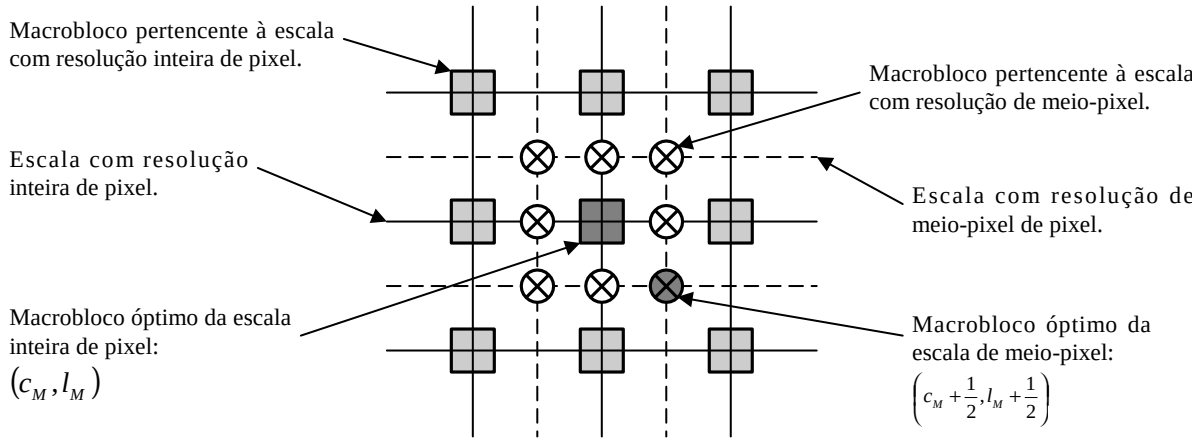


Figura 2.3: Estimação dos vectores de movimento com precisão de meio pixel.

Como, em geral, este método requer o uso de um espaço de memória elevado, é utilizado um esquema de estimação dividido em duas fases:

1. Na primeira fase, começa-se por proceder à estimação do vector de movimento utilizando uma precisão inteira. O vector com precisão inteira assim obtido, $MV_{int}\,(c,l)$ corresponderá, então, ao maior valor de similaridade obtido no processo de procura e será, por isso, usado como estimativa inicial para a segunda fase.

2. Na segunda fase procede-se, então, a um refinamento dos resultados da primeira fase, de forma a obter o vector de movimento com uma maior precisão. Este processo de refinamento é, em geral, realizado da seguinte forma (ver figura 2.3):

   - Definem-se 8 novos macroblocos numa escala com resolução de meio pixel (representada a traço interrompido) na área de pesquisa em redor do macrobloco escolhido na primeira fase. Na figura 2.3 representaram-se com o símbolo $\otimes$ as posições correspondentes a estes macroblocos. Estas posições podem ser definidas como $\left(c + \frac{m}{2}, l + \frac{n}{2}\right)$, onde $m, n \in \{-1, 0, 1\}$. Como parte dos pixels pertencentes a cada um destes macroblocos não se encontram definidos na imagem original, é necessário proceder ao seu cálculo por intermédio de interpolação bilinear, usando, para isso, o valor dos pixels vizinhos (ver figura 2.4).
   - Avalia-se, de seguida, a medida de similaridade correspondente a cada um destes 8 macroblocos, comparando-os com o macrobloco de referência. Tal como se definiu na equação 2.1, define-se como $d\left(c + \frac{m}{2}, l + \frac{n}{2}\right)$ o valor da medida de semelhança correspondente ao macrobloco com o canto superior esquerdo localizado na posição $\left(c + \frac{m}{2}, l + \frac{n}{2}\right)$.

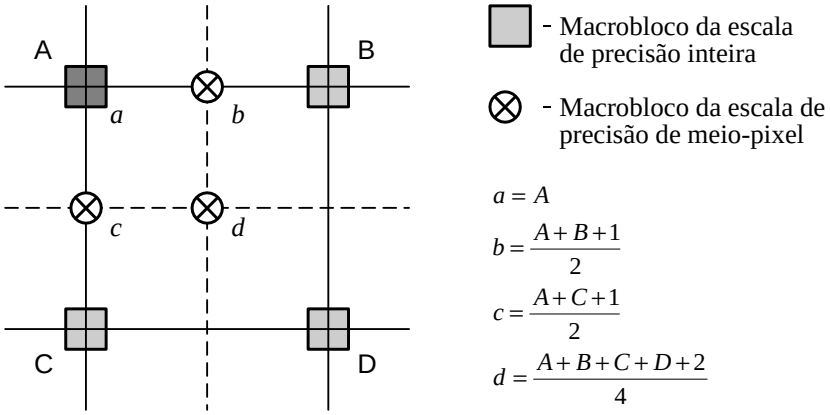


Figura 2.4: Interpolação bilinear necessária para obter uma precisão de meio pixel, segundo a norma H.263 [5].

- Calculadas as medidas de semelhança para todos os macroblocos, escolhe-se a posição $\left(\frac{p}{2}, \frac{q}{2}\right)$ (em que $p, q \in \{-1, 0, 1\}$) correspondente ao macrobloco com maior medida de similaridade do grupo definido pelos 8 macroblocos periféricos e pelo macrobloco central. Assim, esta posição corresponderá ao refinamento desejado da estimativa inicial (c, l), obtida na primeira fase, com uma precisão de meio pixel. O vector de movimento final será, então, definido como sendo $\left(c + \frac{p}{2}, l + \frac{q}{2}\right)$. Convém notar que pode perfeitamente acontecer a situação de a estimativa inicial (c, l) corresponder à posição com maior similaridade. Nesse caso, ter-se-ia, naturalmente, $p = q = 0$, o que correspondente a uma posição definida na escala com resolução inteira.

De igual forma, poder-se-ia, naturalmente, definir um método semelhante para estimar os vectores de movimento noutras sub-escalas, tal como, por exemplo, em escalas com resolução de um quarto de pixel.

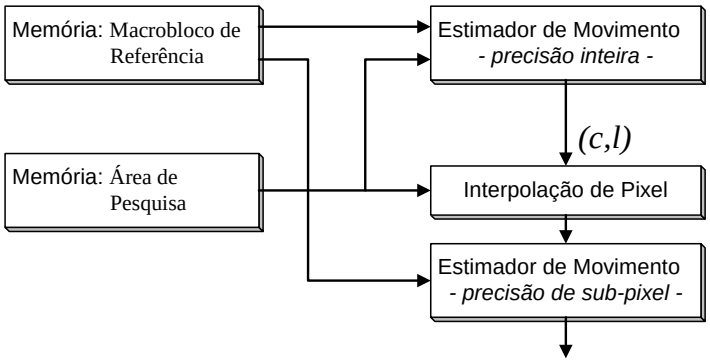


Figura 2.5: Sistema de estimação de movimento com precisão de meio-pixel.

Em muitos dos circuitos para estimação de movimento actuais, estas duas unidades de pesquisa, com precisão inteira e com precisão de meio pixel, são implementadas em unidades separadas (ver figura 2.5). Contudo, como a primeira fase do algoritmo corresponde à parte do procedimento de pesquisa mais exigente em termos computacionais, a investigação de circuitos sobre estimação de movimento tem incidido sobre a estimação de movimento com precisão inteira [10, 11].

2.4 Medidas de similaridade

Nos últimos anos, várias medidas de semelhança têm sido propostas para estimação de movimento [**?**]. Na tabela 2.1, apresentam-se algumas destas medidas, representando-se por $R(u,v)$ o conjunto dos pixels que constituem o macrobloco de referência, por $S(u+c, v+l)$ o conjunto de pixels da área de pesquisa para um deslocamento de (c,l) do macrobloco actual, por $\overline{R}$ a média local dos pixels que constituem o macrobloco de referência: $\overline{R} = \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} R(u,v)$ e por $\overline{S(c,l)}$ a média local do macrobloco candidato na área de pesquisa sob comparação: $\overline{S(c,l)} = \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} S(c+u, l+v)$.

Estas medidas podem ser divididas em duas classes distintas:

- Medidas de semelhança, onde o critério de aproximação é baseado no cálculo da correlação entre os dois blocos em comparação. Assim, a melhor aproximação corresponderá ao par de macroblocos para o qual se obtém o maior valor para estas medidas. Exemplos destas medidas são: SCC, NCC, ZNCC e MOR.
- Medidas de diferença ou dissemelhança, onde o critério de aproximação é baseado na diferença entre cada pixel de cada um dos blocos. Nestes casos, a melhor aproximação corresponderá ao par de blocos com que se obtém o menor valor de dissimilaridade. Exemplos destas medidas são: NZSSD, SSD, SAD, NSSD, ZSSD, ZSAD, LSSD and LSAD.

Em cada uma destas classes, destacam-se algumas funções que se apresentam como versões normalizadas em relação ao valor médio ou ao desvio padrão de funções mais simples. O objectivo destas normalizações consiste em tornar as medidas, tanto quanto possível, insensíveis a variações do nível de brilho e contraste de $R(u,v)$ e $S(u,v)$. Embora as funções descritas apresentem algumas analogias evidentes, elas apresentam requisitos computacionais diferentes. Enquanto que para a função mais simples, a SCC, apenas é necessário efectuar $N \times N$ operações do tipo MAC[3], para outras é necessário o uso de unidades aritméticas capazes de efectuar operações mais complexas, tais como o cálculo

[3] Do Inglês *Multiply and Accumulate*.

Medida de Semelhança	Definição
Simple Cross-Correlation **SCC**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1} R(u,v)\cdot S(c+u,l+v)$
Normalized Cross-Correlation **NCC**(**c**, **l**)	$\frac{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1} R(u,v)\cdot S(c+u,l+v)}{\sqrt{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1} R^2(u,v)\cdot \sum_{v=0}^{N-1}\sum_{u=0}^{N-1} S^2(c+u,l+v)}}$
Zero Mean Normalized Cross-Correlation **ZNCC**(**c**, **l**)	$\frac{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-\overline{R}\right)\cdot\left(S(c+u,l+v)-\overline{S(c,l)}\right)}{\sqrt{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-\overline{R}\right)^2\cdot\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(S(c+u,l+v)-\overline{S(c,l)}\right)^2}}$
Moravec **MOR**(**c**, **l**)	$\frac{2\cdot\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-\overline{R}\right)\cdot\left(S(c+u,l+v)-\overline{S(c,l)}\right)}{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-\overline{R}\right)^2+\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(S(c+u,l+v)-\overline{S(c,l)}\right)^2}$
Normalized Zero Mean Sum of Squared Differences **NZSSD**(**c**, **l**)	$\frac{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left[\left(R(u,v)-\overline{R}\right)-\left(S(c+u,l+v)-\overline{S(c,l)}\right)\right]^2}{\sqrt{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-\overline{R}\right)^2\cdot\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(S(c+u,l+v)-\overline{S(c,l)}\right)^2}}$
Sum of Squared Differences **SSD**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-S(c+u,l+v)\right)^2$
Sum of Absolute Differences **SAD**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left\lvert R(u,v)-S(c+u,l+v)\right\rvert$
Normalized Sum of Squared Differences **NSSD**(**c**, **l**)	$\frac{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-S(c+u,l+v)\right)^2}{\sqrt{\sum_{v=0}^{N-1}\sum_{u=0}^{N-1} R^2(u,v)\cdot\sum_{v=0}^{N-1}\sum_{u=0}^{N-1} S^2(c+u,l+v)}}$
Zero Mean Sum of Squared Differences **ZSSD**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left[\left(R(u,v)-\overline{R}\right)-\left(S(c+u,l+v)-\overline{S(c,l)}\right)\right]^2$
Zero Mean Sum of Absolute Differences **ZSAD**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left\lvert\left(R(u,v)-\overline{R}\right)-\left(S(c+u,l+v)-\overline{S(c,l)}\right)\right\rvert$
Locally Scaled Sum of Squared Differences **LSSD**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left(R(u,v)-\frac{\overline{R}}{\overline{S(c,l)}}\cdot S(c+u,l+v)\right)^2$
Locally Scaled Sum of Absolute Differences **LSAD**(**c**, **l**)	$\sum_{v=0}^{N-1}\sum_{u=0}^{N-1}\left\lvert R(u,v)-\frac{\overline{R}}{\overline{S(c,l)}}\cdot S(c+u,l+v)\right\rvert$

Tabela 2.1: Funções de similaridade.

da *raiz quadrada* (ZNCC, NZSSD, NSSD), do *valor absoluto* (SAD, ZSAD, LSAD) e da *divisão inteira* (NCC, ZNCC, MOR, NZSSD, NSSD, LSSD, LSAD). Estes requisitos são, muitas vezes, um aspecto determinante para a escolha da função de similaridade a utilizar para uma dada aplicação.

Por forma a reduzir os requisitos computacionais necessários para o cálculo da semelhança entre os macroblocos, foram propostas algumas medidas de similaridade simplificadas. Um exemplo destas medidas é a *Normalized Absolute Difference* (NSAD). Esta função baseia-se na soma de valores pertencentes ao conjunto $\{0,1\}$, dependente do resultado da comparação entre um dado limiar e o valor absoluto da diferença entre o pixel do macrobloco de referência e o pixel do macrobloco anterior:

$$NSAD\,(c,l) = \sum_{v=0}^{N-1}\sum_{u=0}^{N-1} T\,(v,u,c,l) \tag{2.4}$$

$$T\,(v,u,c,l) = \begin{cases} 1 & ,\ |R\,(u,v) - S\,(c+u,l+v)| > Limiar \\ 0 & ,\ caso\ contrário \end{cases} \tag{2.5}$$

Outra função proposta designa-se por *Maximum Pixel Difference* (MPD) e corresponde ao máximo valor absoluto dos pixels sob comparação:

$$MPD\,(c,l) = \max_{(u,v)} |R\,(u,v) - S\,(c+u,l+v)|\,;\ \ 0 \leq u,v < N \tag{2.6}$$

2.5 Requisitos computacionais

De entre as medidas atrás referidas, tomar-se-á, como exemplo, a medida SAD (*Sum of Absolute Differences*), também designada por MAD (*Mean Absolute Difference*) ou distância L_1. Esta função tem sido a mais frequentemente utilizada nas várias aplicações envolvendo estimação de vectores de movimento ou de disparidades entre pares de imagens constituídas por pixels definidos numa gama de tons de cinzento. Esta função requer um esforço computacional moderado e conduz a resultados próximos dos obtidos com as funções mais complexas [**?**]. O valor da medida de dissimilaridade $d\,(c,l)$ de um determinado macrobloco com um vector de movimento estimado de (c,l) é dado por:

$$d\,(c,l) = \sum_{v=0}^{N-1}\sum_{u=0}^{N-1} |R\,(u,v) - S\,(c+u,l+v)|\,;\ \ -p \leq c,l < p \tag{2.7}$$

Para calcular d, para um dado valor de (c,l), são necessárias N^2 operações de subtracção, de cálculo do valor absoluto e de acumulação ($3N^2$ operações). Várias estratégias podem ser usadas para determinar o par de macroblocos correspondente ao melhor emparelhamento. De entre elas, destaca-se a técnica de procura exaustiva, que conduz a

um resultado óptimo com base num processamento regular. Esta estratégia promove a pesquisa em todas as $(2p+1)^2$ posições possíveis na área de pesquisa, obtendo-se o vector de movimento determinando a configuração correspondente ao valor mínimo de dissimilaridade:

$$MV = \arg\min_{(c,l)}\{\, d\,(c,l) \;:\; -p \leq c,l < p\,\} \tag{2.8}$$

Assim, para imagens de dimensão $N_h \times N_v$ pixels (N_h pixels por linha e N_v linhas por imagem), o método de emparelhamento de blocos com pesquisa exaustiva envolve:

$$\frac{N_h \times N_v}{N^2}\,(2p+1)^2\,3N^2 = 3\,(2p+1)^2 \times (N_h \times N_v) \tag{2.9}$$

operações por imagem. Se se considerar um ritmo de f_F imagens por segundo (I/s), conclui-se que este método requer um ritmo de $3\,(2p+1)^2\,(N_h \times N_v)\,f_F$ operações por segundo. Considerando $p = 15$ e o formato de imagem CIF - $N_h = 352$, $N_v = 288$, $f_F = 30\ I/s$ - é necessário realizar 8,77 GOPS[4].

[4]Do Inglês *Giga OPerations per Second*.

Capítulo 3

Algoritmos para estimação de movimento

Conteúdo

3.1 Introdução

O desempenho do bloco estimador de movimento é determinante na eficiência de um sistema de codificação de vídeo. Por isso, tem-se verificado, nos últimos anos, um grande esforço de investigação de algoritmos eficientes para estimação de movimento, que se podem classificar em duas classes distintas:

- *Algoritmos de pesquisa óptimos*, baseados numa procura exaustiva num conjunto de macroblocos candidatos da imagem anterior, do macrobloco que melhor emparelha com o macrobloco a codificar.

- *Algoritmos de pesquisa sub-óptimos*, baseados em simplificações ao nível dos processos de procura do melhor macrobloco candidato e das operações de comparação, para reduzir o tempo de processamento e/ou os recursos de cálculo necessários.

Assim, dado o impacto deste bloco na complexidade computacional e na eficiência da codificação, é importante seleccionar o melhor algoritmo para uma dada aplicação.

3.2 Algoritmos de pesquisa exaustiva

De acordo com a equação 2.1, um método de processamento regular para estimar o vector de movimento de um dado macrobloco consiste no cálculo da medida de semelhança ($d(c,l)$) para cada macrobloco candidato da área de pesquisa. Este método, designado de *procura exaustiva*, pode ser descrito através do pseudo-código apresentado na figura 3.1, onde, a título de exemplo, se utilizou a medida de similaridade SAD[1].

Este tipo de algoritmo é o que permite chegar à solução óptima, sendo, por isso, aconselhado para efectuar codificação de vídeo de alta qualidade e para débitos binários reduzidos.

Contudo, este método apresenta algumas desvantagens: para além de apresentar níveis de complexidade muito elevados (da ordem de $\mathcal{O}(N_h \times N_v)$), para se alcançar processamento em tempo real é necessário garantir um elevado ritmo de entrada dos dados e ritmos de processamento muito grandes, muitas vezes apenas conseguidos recorrendo a processamento paralelo massivo. No entanto, devido à sua estrutura regular, este tipo de algoritmos apresenta características adequadas para a implementação de circuitos integrados dedicados para a estimação de movimento.

[1] Do Inglês *Sum of Absolute Differences*.

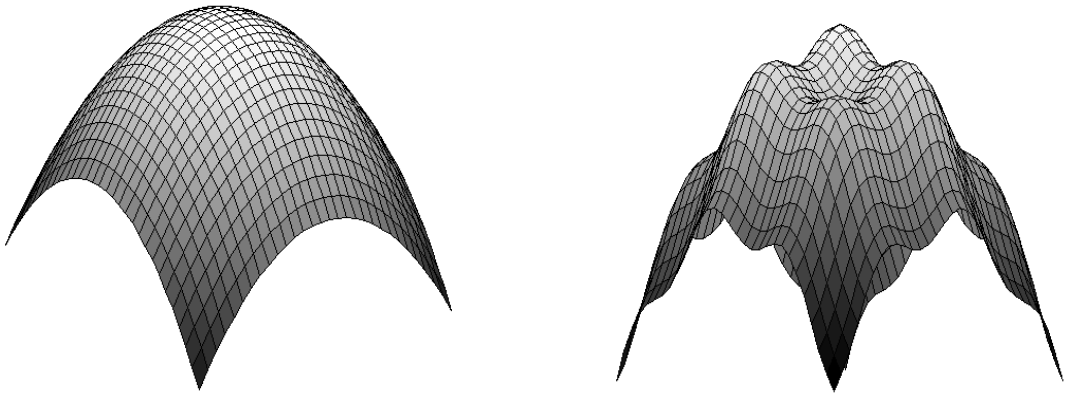
```
(x, y) ⇐ (0, 0) {inicializacao dos vectores de movimento}
SAD (x, y) ⇐ ∞
for c = −p to p do
  for l = −p to p do
    SAD (c, l) ⇐ 0 {inicializacao da medida SAD}
    for u = 0 to N do
      for v = 0 to N do
        SAD (c, l) + = |R (u, v) − S (c + u, l + v) |
      end for
    end for
    if SAD (c, l) < SAD (x, y) then
      (x, y) = (c, l) {comparacao}
      SAD (x, y) = SAD (c, l)
    end if
  end for
end for
return (x, y) {vector de movimento = (x, y)}
```

Figura 3.1: Algoritmo de pesquisa exaustiva do vector de movimento óptimo.

3.3 Algoritmos de pesquisa sub-óptimos

Tal como se referiu anteriormente, o algoritmo de pesquisa exaustiva produz o resultado óptimo, mas requer um elevado número de operações (ver equação 2.7). Para reduzir os requisitos computacionais, tem-se assistido à investigação de algoritmos de procura que exigem um número mais reduzido de operações, mas que fornecem apenas soluções sub-óptimas. No entanto, é importante referir que esta redução do número de operações tem, como principal inconveniente, o facto de requerer um bloco de controlo bastante mais complexo. Por esta razão, este tipo de algoritmos de pesquisa é muitas vezes vocacionado para implementações baseadas em *software*, pois a sua implementação suportada em processadores digitais especializados é, em geral, bastante mais complexa, devido à sua natureza pouco regular.

A generalidade dos algoritmos de pesquisa sub-óptima baseia-se na hipótese, nem sempre verdadeira, de que existe um máximo único e absoluto da medida de similaridade utilizada no espaço de procura, a partir do qual as medidas de semelhança entre os macroblocos candidatos e o macrobloco de referência decrescem monotonicamente em todas as direcções. Como esta hipótese não se verifica em todas as situações, a procura termina, muitas vezes, na detecção de máximos locais de semelhança (ver figura 3.2). É devido a esta característica que este tipo de algoritmos é muitas vezes designado de



(a) Situação simples, com um máximo global de semelhança.

(b) Superfície de erro com máximos locais.

Figura 3.2: Diferentes tipos de superfícies de erro.

sub-óptimo, em contraste com os algoritmos de procura exaustiva.

Apesar deste inconveniente, convém referir que, em muitas aplicações, esta característica não afecta significativamente o desempenho final do sistema. De facto, do ponto de vista da codificação de vídeo, pode-se afirmar que o objectivo principal do bloco de estimação não é, necessariamente, o de encontrar o macrobloco correspondente ao emparelhamento óptimo, mas sim o de encontrar o macrobloco que irá permitir uma codificação mais eficiente. Mesmo na situação em que o emparelhamento não seja o mais correcto, o algoritmo de codificação continuará a funcionar, embora com uma eficiência menor.

Nesta perspectiva, a estimação de movimento deve ser vista como uma forma de minimizar a quantidade total de bits que resultam da codificação da informação correspondente ao erro de predição e aos vectores de movimento. De facto, os erros de estimação podem-se considerar perfeitamente toleráveis desde que a quantidade de bits, após a codificação, não aumente. Por isso, em macroblocos onde se verificam vários vectores de movimento possíveis com medidas de similaridade muito próximas, muitos algoritmos optam por escolher o vector de que resultará uma codificação com uma menor quantidade de bits.

A generalidade dos algoritmos de procura rápida baseia-se, essencialmente, numa redução do espaço de procura através de operações de decimação ao nível de diferentes parâmetros (ver figura 2.5), dando assim origem a algoritmos que podem ser classificadas em três categorias distintas:

- *algoritmos com decimação ao nível dos vectores candidatos*, em que se reduz o conjunto de macroblocos candidatos que constituem a área de pesquisa da imagem anterior; este método corresponde a reduzir o número de iterações dos ciclos em

c e em l do pseudo-código da figura 3.1; os macroblocos eliminados aquando da decimação podem ser utilizados em fases posteriores de refinamento da pesquisa.

- *algoritmos com decimação ao nível do pixel*, em que apenas uma parte dos 16×16 pixels que constituem o macrobloco é usada para o cálculo das medidas de similaridade $d(c, l)$. No pseudo-código apresentado na figura 2.5, esta operação corresponde a reduzir o número de iterações dos ciclos em u e em v.

- *algoritmos de pesquisa hierárquicos com redução da resolução das imagens*, em que se aplicam os algoritmos de procura exaustiva sobre versões de menor resolução das imagens originais (obtidas por decimação), para encontrar os vectores de movimento. Neste tipo de algoritmos é usual aplicar um filtro passa-baixo antes de se efectuar a decimação, para evitar fenómenos de *aliasing* que poderiam dar origem a emparelhamentos incorrectos. Do ponto de vista da redução do número total de operações necessárias, este método é considerado bastante eficiente, visto consistir na aplicação simultânea de decimações ao nível do pixel e ao nível dos vectores de movimento candidatos. Em geral, efectua-se primeiro uma estimativa dos vectores de movimento, procedendo-se, em seguida, ao refinamento da pesquisa, obtendo-se vectores de movimento com precisão inteira de pixel.

Nas secções seguintes descrevem-se alguns dos principais algoritmos de cada uma das categorias referidas.

3.3.1 Pesquisa com decimação ao nível dos vectores candidatos

Um grande número de algoritmos deste tipo têm sido propostos por diversos autores, apresentando-se, de seguida, dois destes algoritmos de pesquisa logarítmica e de pesquisa paralela hierárquica.

Pesquisa Logarítmica

Tal como se descreveu na secção 2.3, a área de pesquisa corresponde, em geral, a uma zona quadrada com uma área total de $(p + N + p)^2 = (N + 2p)^2$ pixels (ver figura 2.2), correspondendo a $(2p + 1)^2$ macroblocos candidatos. Na representação matricial da figura 3.3 representaram-se, através de quadrados, os centros de cada um dos $(2p + 1)^2$ macroblocos candidatos que constituem a área total de pesquisa. O algoritmo de *pesquisa logarítmica*, proposto por Koga *et al.* [12], segue um esquema de processamento bastante semelhante ao do conhecido algoritmo de *pesquisa binária*.

Este algoritmo começa por dividir a área de pesquisa em duas regiões, sendo uma correspondente ao interior do quadrado com os lado definidos no intervalo $\left[-\frac{\lceil p \rceil}{2}, \frac{\lceil p \rceil}{2}\right]$ e a outra ao exterior deste quadrado. Numa primeira fase, consideram-se, apenas, 8 macroblocos situados na fronteira destas regiões para efeitos de pesquisa. Assim, designando-se por d_1 a distância entre os centros de cada par de macroblocos adjacentes, o cálculo das funções de similaridade realizar-se-á, apenas, nas seguintes posições: $(0,0)$, $(0,d_1)$, $(0,-d_1)$, $(-d_1,0)$, $(d_1,0)$, (d_1,d_1), $(d_1,-d_1)$, $(-d_1,d_1)$ e $(-d_1,-d_1)$. A distância d_1 é dada por: $d_1 = 2^{k-1}$, onde $k = \lceil \log_2 p \rceil$. Na figura 3.3 ilustra-se a disposição destes macroblocos para o caso particular em que $p = 7$, $k = 3$ e $d_1 = 4$ pixels.

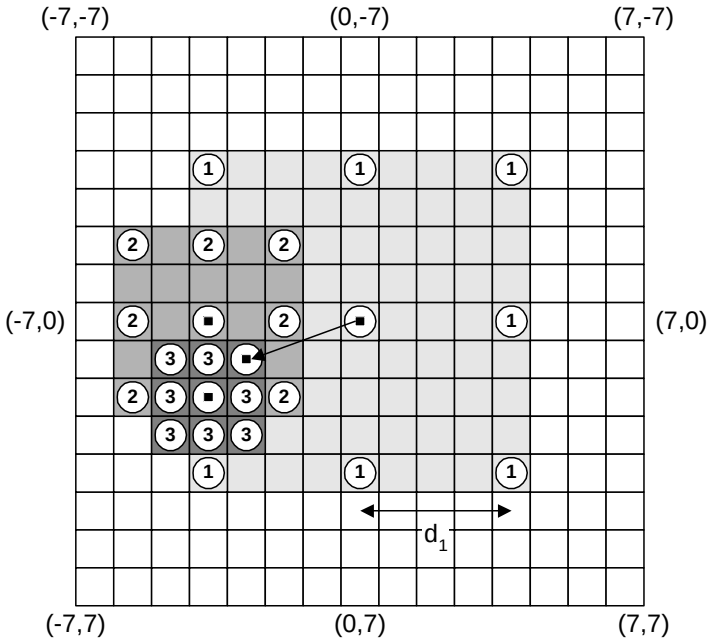


Figura 3.3: Algoritmo de pesquisa logarítmica proposto por Koga [12].

No passo seguinte, selecciona-se o macrobloco correspondente ao maior valor de semelhança para centro da nova região de procura, definem-se novamente 8 posições à distância de $d_2 = d_1/2$ e repete-se o passo anterior, usando as medidas de semelhança correspondentes aos novos 8 macroblocos candidatos.

Este procedimento é, assim, realizado até à iteração k, em que os centros dos 8 macroblocos candidatos estarão distanciados de $d_k = 1$. Calculados os valores de similaridade para cada um destes macroblocos de pesquisa, determina-se aquele com maior valor de semelhança. A posição deste macrobloco definirá, por fim, o vector de movimento para o macrobloco de referência considerado.

Desta forma, este algoritmo efectua a estimação do vector de movimento realizando,

apenas, $(8k+1)$ cálculos da função de semelhança. Considerando o processamento sequências de imagens com uma resolução de $N_h \times N_v$ pixels e um ritmo de f_F imagens por segundo, o número de macroblocos candidatos a processar por segundo é de $3N_hN_vf_F\,(8k+1)$. Tomando, como exemplo, o caso em que $p = 15$ $(k = 4)$ e o formato de imagens CIF, em que $N_h = 352$ pixels por linha, $N_v = 288$ linhas e $f_F = 30$ imagens por segundo, conclui-se que este algoritmo requer cerca de 301 MOPS. Assim, verifica-se que os requisitos computacionais deste algoritmo são apenas cerca de 3,4% dos associados ao algoritmo de pesquisa exaustiva.

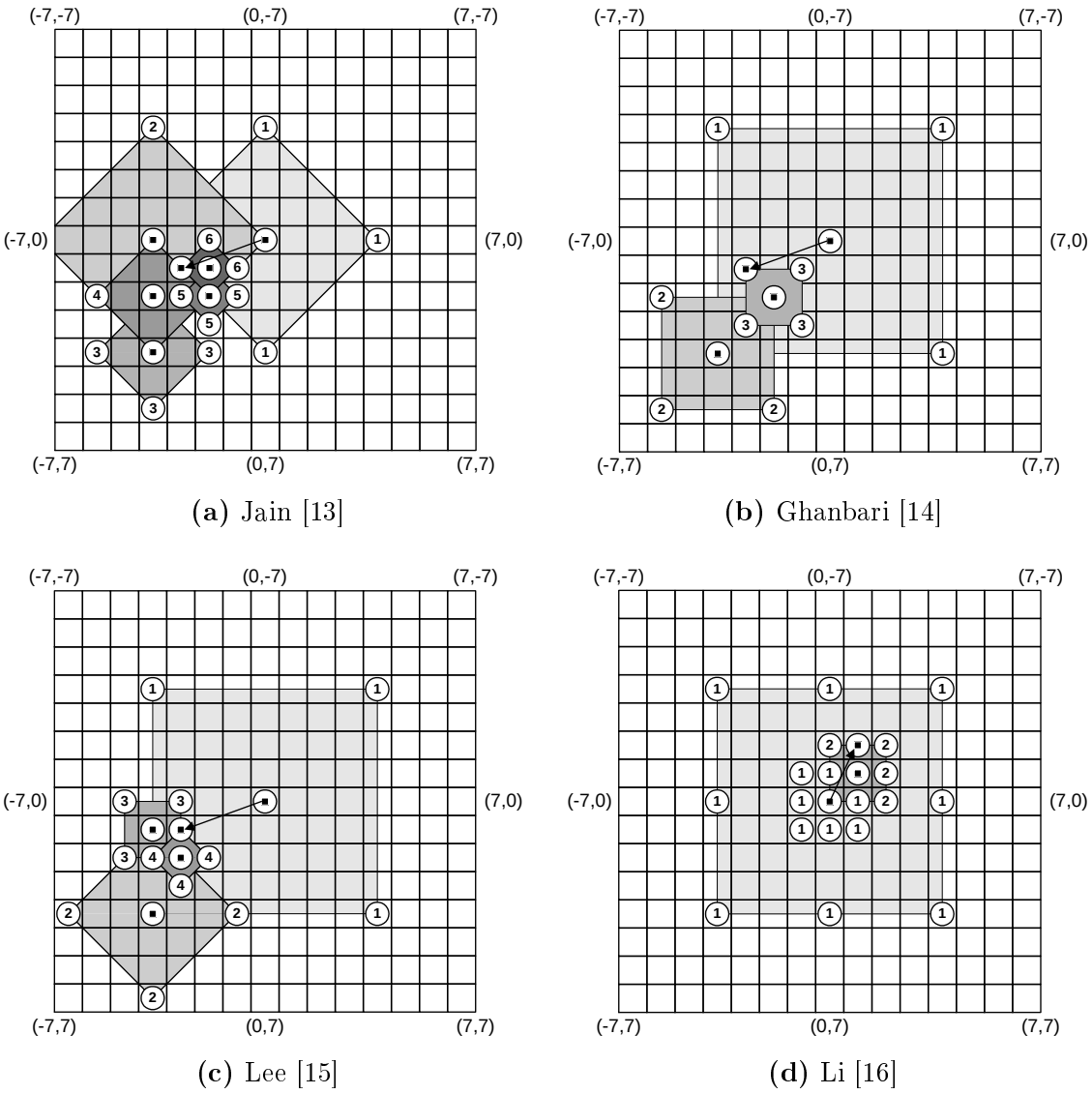
Na figura 3.3 encontra-se descrito o procedimento de procura na sua forma mais conhecida, frequentemente referida na literatura como *algoritmo dos 3 passos*, onde $p = 7$ e $k = 3$. Esta área de procura é, geralmente, considerada aceitável para aplicações do tipo de videoconferência. As posições de procura correspondentes a cada um dos 3 passos foram marcadas com os símbolos ①, ② e ③:

- no primeiro passo tomou-se como macrobloco central o correspondente às coordenadas $(0, 0)$; calcularam-se os 9 valores da função de similaridade correspondentes às posições marcadas com o símbolo ①, sendo a distância entre o centro de cada um destes macroblocos de $d_1 = 4$ pixels; considere-se que a posição $(-4, 0)$ corresponde ao candidato com maior valor de similaridade;

- no segundo passo, tomando-se o macrobloco situado na posição $(-4, 0)$ como centro da área de pesquisa, calcularam-se os 8 valores de similaridade correspondentes aos macroblocos candidatos marcados com o símbolo ②; neste nível, a distância entre o centro de cada um dos macroblocos candidatos é de $d_2 = d_1/2 = 2$ pixels. Suponha-se, agora, que o candidato com maior medida de similaridade se encontra localizado em $(-4, 2)$.

- por fim, no terceiro e último passo, tomou-se como centro o macrobloco situado na posição $(-4, 2)$ e calcularam-se os 8 valores da medida de similaridade para cada um dos candidatos marcados com o símbolo ③; a distância entre o centro de cada um destes macroblocos é de $d_3 = d_2/2 = 1$ pixel; considerando que a posição $(-3, 1)$ corresponde ao maior valor de similaridade, o vector de movimento estimado tem as coordenadas $(-3, 1)$.

Um algoritmo de pesquisa logarítmica com características muito semelhantes foi proposto por Jain [13], usando 4 macroblocos de pesquisa localizados dos vértices de uma área com a forma de um losango. Este algoritmo apresenta a vantagem de necessitar, em média, de um menor número de comparações entre macroblocos, apesar de, em certos



(a) Jain [13] **(b)** Ghanbari [14]

(c) Lee [15] **(d)** Li [16]

Figura 3.4: Algoritmos de pesquisa logarítmica propostos por vários autores.

casos, o algoritmo poder requerer um número total de passos maior (ver figura 3.4(a)). Ghanbari [14] propôs, também, um algoritmo baseado na pesquisa de 4 posições localizadas nos vértices de zonas quadradas definidas na área de pesquisa correspondente a cada nível (ver figura 3.4(b)).

Lee [15] propôs uma alteração para adaptar a dimensão da área de pesquisa do algoritmo (usando um factor de 1/4 ou de 3/4), em função da rapidez de convergência. Para além disso, sugeriu, ainda, alternar a forma geométrica da área de pesquisa entre um quadrado e um losango. Os macroblocos candidatos definem-se, também, nos vértices de cada uma destas áreas (ver figura 3.4(c)). Li [16] apresentou uma proposta de opti-

mização do algoritmo tradicional, tirando partido da elevada probabilidade de o vector de movimento estimado se encontrar na vizinhança do centro da área de pesquisa. Assim, no primeiro passo, o algoritmo começa por pesquisar 2 grupos de macroblocos candidatos: as 8 posições da periferia da área de pesquisa e as 8 posições em redor do centro $(0,0)$. Se a posição com maior valor de similaridade corresponder a um dos macroblocos do primeiro grupo, o funcionamento do algoritmo é idêntico ao de um algoritmo de pesquisa logarítmica. Se, pelo contrário, o macrobloco candidato com maior valor de similaridade se encontrar no segundo grupo, o algoritmo continua a pesquisa a partir dessa posição, efectuando um único passo com uma distância $d_2 = 1$ (ver figura 3.4(d)). Mostra-se, experimentalmente, que este método permite obter acelerações significativas na estimação de vectores de movimento, explorando os movimentos lentos ao longo do tempo.

Pesquisa paralela hierárquica a uma dimensão

Ao contrário do algoritmo de pesquisa logarítmica, o algoritmo de pesquisa paralela efectua a procura de uma forma independente nas duas direcções do sistema de eixos i e j.

Assim, dada uma área de pesquisa definida no intervalo $[-p, p]$ considere-se a distância entre macroblocos candidatos adjacentes $S = 2^{\lfloor \log_2 p \rfloor}$ e designe-se a origem (ou centro) da área de pesquisa localizada na posição $(0,0)$ por (d_i, d_j). De acordo com esta designação, determina-se, em paralelo:

- o macrobloco candidato correspondente ao maior valor de similaridade, de entre o conjunto de macroblocos localizados ao longo do eixo i, nas posições $(d_i - S, d_j)$, (d_i, d_j) e $(d_i + S, d_j)$; actualizando o valor de d_i para a coordenada correspondente a essa posição;

- o macrobloco candidato correspondente ao maior valor de similaridade de entre o conjunto de macroblocos localizados ao longo do eixo j, nas posições $(d_i, d_j - S)$, (d_i, d_j) e $(d_i, d_j + S)$; após se efectuar a actualização do valor de d_j com a coordenada correspondente a esta posição faz-se, então, $S = \frac{S}{2}$.

Este processamento paralelo é, assim, repetido sucessivamente até se ter $S = 0$. A posição final (d_i, d_j) corresponderá, então, ao vector de movimento estimado, correspondente ao melhor emparelhamento.

Na figura 3.5 encontra-se ilustrado o funcionamento deste algoritmo para o caso de $p = 7$. Assim, de acordo com o que se descreveu anteriormente, começa-se por usar uma distância entre macroblocos candidatos igual a $S = 2^{\lfloor \log_2 7 \rfloor} = 4$.

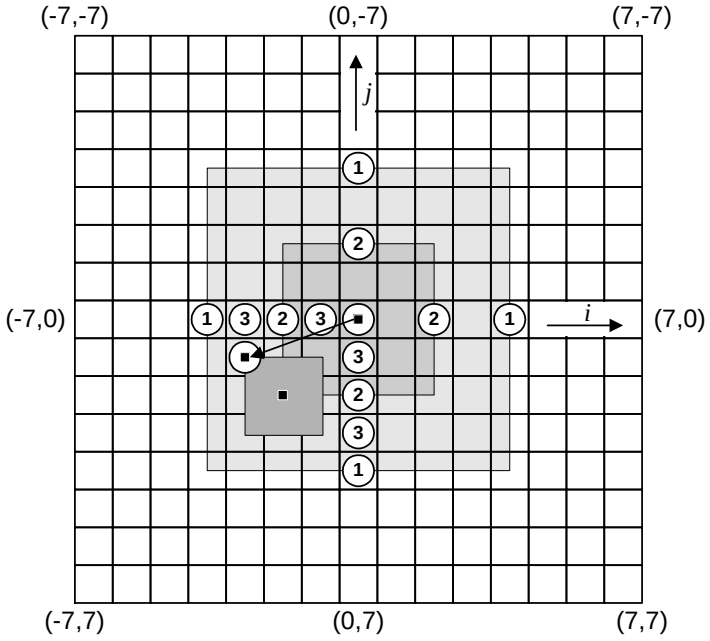


Figura 3.5: Algoritmo de pesquisa paralela.

No primeiro passo, correspondente ao eixo i, calculam-se os valores de similaridade para as posições marcadas com o símbolo ①. Suponha-se que a ordenada $i = 0$ corresponde à posição com maior valor de similaridade. Em paralelo, calculam-se também os valores de similaridade para cada uma das posições marcadas com o símbolo ① no eixo j. Suponha-se que a ordenada $j = 0$ corresponde à posição com maior valor de similaridade. Assim, a origem para a pesquisa no passo seguinte será a posição $(0, 0)$.

No segundo passo, começa-se por reduzir o valor de S para $S = 2$. De seguida, determina-se qual das posições marcadas com o símbolo ② no eixo i (e centradas em torno de $(0, 0)$) corresponde ao maior valor de similaridade. Considere-se que esta posição corresponde à ordenada $i = -2$. Da mesma forma, determina-se a posição com maior valor de similaridade de entre o conjunto de posições definidas ao longo do eixo j. Considere-se que esta posição corresponde à ordenada $j = 2$. Assim, a origem para a pesquisa no passo seguinte será a posição $(-2, 2)$.

No terceiro passo, o valor de S é, uma vez mais, dividido por dois, obtendo-se $S = 1$. De seguida, determina-se a posição correspondente ao maior valor de similaridade de entre as posições marcadas com o símbolo ③ no eixo i. Em paralelo, efectua-se também a pesquisa da posição correspondente ao maior valor de semelhança de entre as posições marcadas com o símbolo ③ no eixo j. Suponha-se que estas posições correspondem às coordenadas $(-3, 1)$. Como neste caso se tem $S = 1$, dá-se como terminado o processo

de procura. Assim, o valor $(-3, 1)$ corresponderá, então, ao valor final do vector de movimento estimado para o macrobloco actual.

Desta forma, verifica-se que, no máximo, são necessários um total de 13 cálculos da função de similaridade. Considerando um formato de imagem CIF, com $N_h = 352$ pixels por linha, $N_v = 288$ linhas e $f_F = 30$ imagens por segundo, conclui-se que este algoritmo requer cerca de 118,6 MOPS, correspondente a 1,35% do número de cálculos do algoritmo de pesquisa exaustiva e a 40% do número de operações do algoritmo de pesquisa logarítmica.

Para além desta redução em termos de complexidade, este algoritmo apresenta, ainda, duas outras propriedades importantes:

- o cálculo das funções de semelhança é paralelizável;
- o fluxo de dados é regular, pois as posições de procura encontram-se definidas ao longo dos eixos i e j.

Contudo, como facilmente se percebe, os resultados deste algoritmo dependem fortemente da existência de máximos locais da função de similaridade, tal como se referiu no início da secção 3.3.

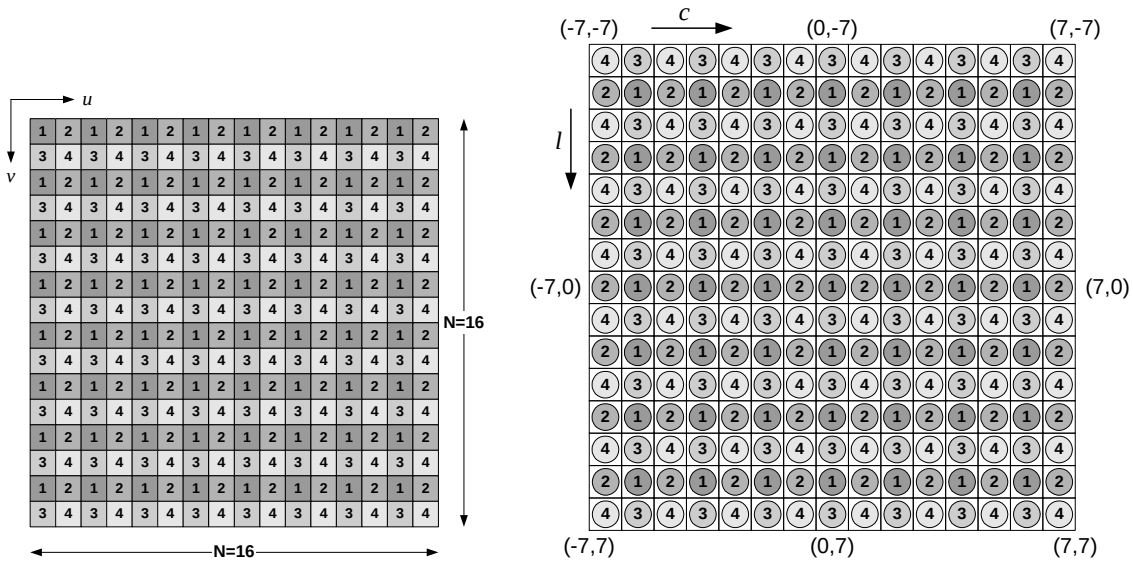
3.3.2 Pesquisa com decimação ao nível do pixel

Tal como se referiu anteriormente, pode-se reduzir a complexidade do processo de estimação de movimento através de uma decimação ao nível dos pixels usados na computação da medida de similaridade considerada. De facto, como o processo de emparelhamento utilizado pressupõe o mesmo movimento para todos os pixels de um determinado macrobloco, pode-se estimatimar o movimento com um número de pixels menor. Contudo, tal como se referiu no início da secção 3.3, o processo de decimação pode reduzir a precisão com que se estima o vector de movimento. Para além disso, pelo facto deste tipo de algoritmos utilizar medidas de similaridade calculadas com um menor número de amostras, apresentam uma maior sensibilidade ao nível do ruído presente nas imagens.

Neste contexto, podem-se identificar duas aproximações distintas nos algoritmos com decimação: *decimação espacial* e *projecção segundo as direcções principais*.

Decimação espacial

A aproximação proposta por Liu [?] baseia-se num tipo de processamento semelhante ao usado no método de procura exaustiva, mas usando apenas, 1/4 do número total de pixels (ver figura 3.6) para o cálculo da medida de similaridade. Na figura 3.6, os pixels do



(a) Macrobloco actual.

(b) Distribuição dos macroblocos candidatos na área de pesquisa considerada.

Figura 3.6: Algoritmo de decimação espacial proposto por Liu [**?**].

macrobloco actual representados com o símbolo 1 são usados para o cálculo das medidas de semelhança do macrobloco candidato $(0, 0)$ e dos macroblocos localizados nas posições $(2c, 2l)$, representados com o símbolo ①. Os pixels marcados com o símbolo 2 são usados para o cálculo da medida de similaridade dos macroblocos candidatos $(2c, 2l + 1)$, representados com o símbolo ②. Os pixels marcados com o símbolo 3 são usados para os macroblocos $(2c + 1, 2l)$, representados com o símbolo ③, e os pixels marcados com o símbolo 4 são usados para o cálculo da medida de semelhança dos macroblocos candidatos localizados nas posições $(2c + 1, 2l + 1)$, representados com o símbolo ④.

Alternando o padrão de pixels considerados para o processo de comparação, e associando-o com o padrão de macroblocos candidatos a comparar, é possível processar todos os pixels do macrobloco actual em grupos de 4 macroblocos candidatos. Além disso, esta alternância elimina a possibilidade de não se considerarem grupos contínuos de pixels, segundo as direcções horizontal, vertical e diagonal no processo de estimação.

Com este esquema de processamento consegue-se reduzir, em cerca de 75%, a complexidade computacional do algoritmo de pesquisa exaustiva.

Projecção segundo as direcções principais

Uma aproximação diferente para reduzir o número de pixels usado no cálculo das medidas de similaridade foi proposta por Ogura [17]. Nesta aproximação, efectua-se a projecção do

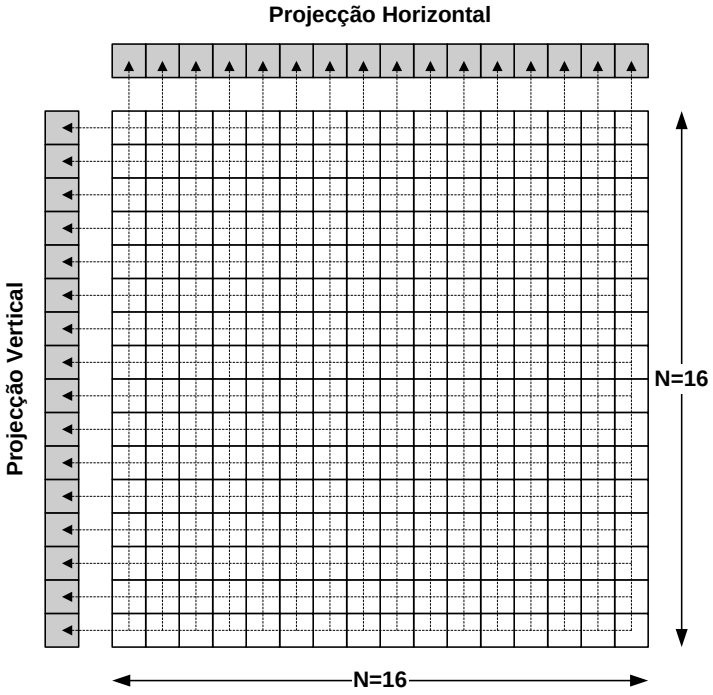


Figura 3.7: Projecções horizontal e vertical do macrobloco actual de dimensão 16×16.

macrobloco actual segundo várias direcções e utilizam-se essas projecções para estimar o movimento (ver figura 3.7). Cada elemento da projecção segundo uma determinada coluna (ou linha) resulta da soma de todos os 16 pixels da coluna (linha) do macrobloco actual. Assim, no cálculo da medida de similaridade da equação 2.1, os dois somatórios segundo os índices u e v (os $N \times N$ pixels) são substituídos por comparações das projecções respectivas segundo as direcções horizontal ou vertical. Esta aproximação permite, por isso, reduzir o número total de operações de um factor de $\frac{2}{N} = \frac{N+N}{N \times N}$. No caso de macroblocos de 16×16 pixels obtém-se, assim, uma redução do número de operações para cerca de 12,5%, relativamente ao algoritmo de procura exaustiva.

Para estimar os vectores de movimento com uma precisão maior pode-se utilizar, também, projecções segundo as direcções diagonais. Na prática, verifica-se que, para aplicações de vídeoconferência em canais de baixo débito, onde se requer uma taxa de compressão elevada, a utilização de duas projecções, segundo as direcções horizontal e vertical, permite obter resultados com uma precisão muito próxima da obtida com métodos de procura exaustiva. Note-se, ainda, que este método pode, também, ser combinado com as diferentes técnicas de decimação ao nível dos vectores de movimento candidatos descritas anteriormente, reduzindo-se, ainda mais, a complexidade computacional do algoritmo de pesquisa.

3.3.3 Pesquisa hierárquica com redução de resolução das imagens

Os algoritmos de pesquisa hierárquica baseiam-se na combinação dos métodos anteriormente descritos, baseados na decimação ao nível dos pixels e na decimação ao nível dos vectores candidatos. Na figura 3.8, apresenta-se a hierarquia de processamento deste tipo de algoritmos. Através da operação de decimação são formadas imagens de menor resolução, neste caso três níveis de resolução: *Nível 0* (resolução original) e *Níveis 1 e 2*. Desta forma, considerando o macrobloco assinalado na imagem actual com coordenadas (x, y) no nível 0, este macrobloco corresponde a macroblocos com coordenadas $\left(\frac{x}{2}, \frac{y}{2}\right)$ e $\left(\frac{x}{4}, \frac{y}{4}\right)$ nos níveis 1 e 2, respectivamente. Considere-se, também, que o macrobloco no nível 0 apresenta uma dimensão de 16×16 pixels e que a área de pesquisa se estende de $\pm p$ pixels em ambas as direcções.

O procedimento de procura deste algoritmo começa no nível 2, de menor resolução. Neste nível, a dimensão de cada macrobloco é de 4×4 pixels. Da mesma forma, a área de procura estender-se-á de $\pm\frac{p}{4}$ pixels em ambas as direcções. Assim, dadas as reduzidas dimensões dos macroblocos e das áreas de pesquisa utilizadas neste nível, torna-se possível utilizar o método óptimo de procura exaustiva, com uma área de pesquisa centrada na posição $\left(\frac{x}{4}, \frac{y}{4}\right)$. Considere-se, a título de exemplo, que a medida de similaridade tem o valor máximo para a posição (u_2, v_2).

No nível seguinte (nível 1), a estimação de movimento é efectuada usando macroblocos com a dimensão de 8×8 pixels. Contudo, a área de pesquisa estará, neste caso, centrada na posição $\left(\frac{x}{2} + 2u_2, \frac{y}{2} + 2v_2\right)$ e estender-se-á de apenas ± 1 pixel em torno da origem. Considere-se que a posição (u_1, v_1) corresponde ao macrobloco candidato com maior medida de semelhança neste nível.

Por fim, no nível 0 (com resolução máxima), a procura é efectuada utilizando macroblocos com a dimensão de 16×16 pixels. Neste nível, a área de pesquisa estará centrada na posição $(x + 2u_1, y + 2v_1)$ e, tal como no nível anterior, estender-se-á em torno deste centro de apenas ± 1 pixel em ambas as direcções. A posição do macrobloco candidato com maior medida de similaridade corresponderá, por fim, ao vector de movimento estimado para o macrobloco actual em análise.

Este procedimento permite reduzir o número total de operações através de uma diminuição dos macroblocos candidatos considerados e do número de pixels usados no cálculo das medidas de disparidade. A título de exemplo, considere-se o procedimento de estimação para o caso de imagens com o formato CIF, com $N_h = 352$ pixels por linha, $N_v = 288$ linhas e $f_F = 30$ imagens por segundo. Considere-se, também, macroblocos com dimensão de $N \times N$ pixels (em que $N = 16$), áreas de pesquisa de dimensão $(p + N + p)\,(p + N + p)$ pixels (com $p = 15$) e a utilização do algoritmo óptimo de procura

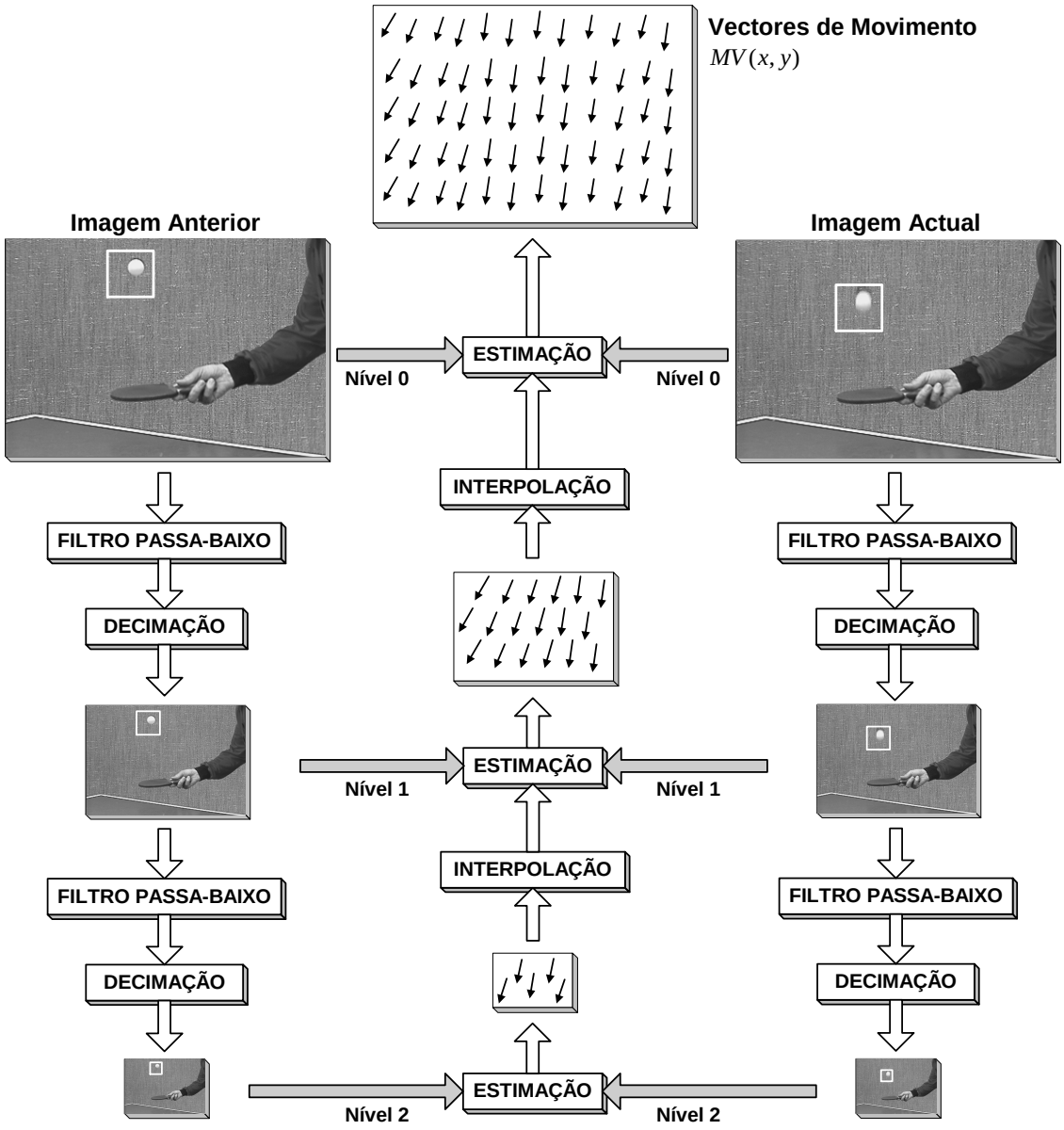


Figura 3.8: Algoritmo de processamento hierárquico com redução de resolução das imagens.

exaustiva em todos os níveis da hierarquia:

- no nível 2, a imagem é representada com 88×72 pixels e cada macrobloco é constituído por 4×4 pixels, sendo o número total de macroblocos a processar por segundo dado por $\frac{88 \times 72}{4 \times 4} \times 30 = 11880$; o número de macroblocos candidatos por macrobloco da imagem actual é, então, dado por $(2 \times p_2 + 1)^2 = (2 \times 4 + 1)^2 = 81$, com $p_2 = \lceil \frac{p}{4} \rceil$; como o número de operações por cada macrobloco é de $4 \times 4 \times 3 = 48$, o número total de operações por segundo requerido é de $11880 \times 81 \times 48 = 46,19$ MOPS.

- no nível 1, a imagem é constituída por 176×144 pixels e um macrobloco tem a dimensão de 8×8 pixels; o número total de macroblocos a processar por unidade de tempo é também 11880; contudo, o número de macroblocos candidatos por macrobloco de referência é de apenas $(2 \times p_1 + 1)^2 = (2 \times 1 + 1)^2 = 9$; como o número de operações por macrobloco é de $8 \times 8 \times 3 = 192$, o número total de operações por segundo é de $11880 \times 9 \times 192 = 20,53$ MOPS.

- no nível 0, a imagem tem uma dimensão de 352×288 pixels e um macrobloco é constituído por 16×16 pixels; o número total de macroblocos a processar por unidade de tempo é também de 11880 e, tal como no nível anterior, o número total de macroblocos candidatos por macrobloco de referência é de $(2 \times p_0 + 1)^2 = (2 \times 1 + 1)^2 = 9$; neste nível, o número de operações por macrobloco candidato é de $16 \times 16 \times 3 = 768$, e o número total de operações por segundo é de $11880 \times 9 \times 768 = 82,11$ MOPS.

Desta forma, verifica-se que para realizar um processamento em tempo real é necessário efectuar 149 milhões de operações por segundo. Assim, conclui-se que os requisitos computacionais deste algoritmo são, para este caso, de apenas $1,69\%$ dos correspondentes ao algoritmo de procura exaustiva. Contudo, convém referir que este algoritmo requer algumas operações adicionais, tais como, filtragem passa-baixo, decimação e interpolação, que fazem aumentar o valor da sua complexidade computacional.

Conclui-se, assim, que os algoritmos de pesquisa hierárquica com redução da resolução das imagens são eficientes do ponto de vista da complexidade computacional. Contudo, apresentam algumas desvantagens, tal como: requerem um maior espaço de memória para armazenar as imagens correspondentes aos vários níveis; certas regiões contendo objectos de pequenas dimensões podem ser completamente eliminadas com o processo de decimação, dando origem a erros na estimação de movimento.

3.3.4 Pesquisa por arrefecimento simulado

Os algoritmos por arrefecimento simulado[2] têm sido recentemente apresentados como heurísticas alternativas aos métodos de pesquisa anteriormente descritos. Com estes algoritmos pretende-se resolver um problema de minimização (ou maximização) evitando, sempre que possível, a convergência para mínimos (ou máximos) locais [18]. Para isso, este tipo de algoritmos não segue uma direcção específica de procura para encontrar a solução óptima, evoluindo segundo um comportamento aleatório do tipo do encontrado nas máquinas de Boltzmann na selecção dos macroblocos a considerar.

[2]Do Inglês *simulated annealing*.

Nos algoritmos por arrefecimento simulado começa-se por considerar uma solução inicial do problema. Em seguida, define-se através de um critério arbitrário, uma forma de chegar a uma solução "vizinha". A selecção deste critério é, frequentemente, a parte mais crítica deste algoritmo, pois a escolha de uma má vizinhança pode conduzir a resultados piores do que os esperados. De seguida, procede-se à comparação das duas soluções: se a solução vizinha for melhor do que a solução anterior, efectua-se o deslocamento parcial e passa-se a utilizar esta solução como posição inicial na iteração seguinte. Se, pelo contrário, a solução vizinha for pior do que a solução anterior, a evolução do algoritmo faz-se de uma forma aleatória. Assim, a probabilidade de aceitação da nova solução dependerá de uma dada variável aleatória, baseada num modelo probabilístico com uma função de densidade de probabilidade dada por $P(\Delta E) = e^{\frac{-\Delta E}{T}}$. Esta aceitação condicionada de soluções permite, assim, que este tipo de algoritmos evite a convergência para mínimos (ou máximos) locais. Em geral, este processo é efectuado através de uma escolha aleatória de um número uniformemente distribuído no intervalo $]0,1[$, seguido de comparação com $P(\Delta E)$: se esse número for menor que $P(\Delta E)$, a nova posição é rejeitada; caso contrário, essa posição será considerada no início da nova iteração. A escolha da função $P(\Delta E)$ leva o sistema a evoluir segundo uma distribuição de Boltzmann.

Na figura 3.9 apresenta-se o algoritmo de pesquisa por arrefecimento simulado para estimação de movimento. Pretende-se, mais uma vez, determinar a posição do macrobloco da imagem anterior que apresenta o maior valor de similaridade com o macrobloco da imagem actual. Como se referiu, a escolha da posição inicial é feita de uma forma arbitrária, considerando-se, em geral, a origem correspondente ao vector de movimento $(0,0)$. Da mesma forma, o critério de escolha da posição vizinha pode ser qualquer. Como exemplo, considere-se que se escolhe o macrobloco de pesquisa distanciado de 1 pixel da posição inicial em qualquer uma das direcções possíveis. Em seguida, procede-se ao cálculo do valor da função de similaridade para esta nova posição $d(MV^{n+1})$, em que n representa o número da iteração e $MV^{n+1} = (c^{n+1}, l^{n+1})$. Se $d(MV^{n+1}) < d(MV^n)$, o algoritmo prossegue para a iteração seguinte, considerando esta nova posição como posição inicial. Caso contrário, se $d(MV^{n+1}) \geq d(MV^n)$, a aceitação desta nova posição far-se-á, apenas, com uma dada probabilidade. De notar que o modelo probabilístico utilizado apresenta uma função densidade de probabilidade com um decréscimo exponencial ao longo do tempo (ver figura 3.9).

Tal como se referiu anteriormente, este modelo probabilístico é, frequentemente, um factor determinante para se obter resultados satisfatórios, tal como o critério de escolha da área de pesquisa a considerar para definir a posição seguinte. Da mesma forma, o parâmetro T (frequentemente designado de temperatura) define a forma de variação da

1. $i \Leftarrow 0$; $T \Leftarrow T_{max}$
Escolha da solução inicial $d\left(MV^0\right)$ aleatoriamente

2. Gerar aleatoriamente uma solução $d\left(MV^{i+1}\right)$, vizinha da solução $d\left(MV^i\right)$

3. Calcular $\Delta E = d\left(MV^{i+1}\right) - d\left(MV^i\right)$

4. Calcular $P = \begin{cases} e^{\frac{-\Delta E}{T}} & , se\ \Delta E > 0 \\ 1 & , se\ \Delta E \leq 0 \end{cases}$

5. **Se** $P = 1$,
- Aceitar a mudança

Caso contrário,
- Sortear um número aleatório Q, definido no intervalo $]0, 1[$
- **Se** $Q < P$,
  - Aceitar a mudança

  Caso contrário,
  - Continuar

6. $i \Leftarrow i + 1$

7. **Se** $i < i_{max}$,
Voltar a **2**

8. $i \Leftarrow 0$; $d\left(MV^0\right) \Leftarrow d\left(MV^{i_{max}}\right)$

9. $T = \frac{K}{\ln(i+1)}$

10. **Se** $T > T_{min}$,
- Voltar a **2**

Caso contrário,
- Termina

Figura 3.9: Algoritmo de pesquisa por arrefecimento simulado.

probabilidade de aceitação ao longo do tempo. A constante K apresenta-se como um valor de parametrização do algoritmo, $d\left(MV^i\right)$ é a solução obtida na iteração i e T_{min} é o valor mínimo de temperatura, a partir do qual o algoritmo termina.

Em geral, considera-se terminada a fase de procura deste algoritmo ao fim de um dado número máximo de iterações i_{max}, ou no caso de se verificar que, durante M iterações consecutivas, a diferença entre a solução óptima até aí encontrada e a actual é menor que um dado valor residual ϵ.

Conforme se pode constatar, este tipo de algoritmos encontra-se especialmente vocacionado para implementações de *software*. De facto, a irregularidade do seu processamento tornaria a implementação em *hardware* muito complexa e, provavelmente, bastante ineficiente.

Capítulo 4

Estimadores de movimento com arquitecturas sistólicas

Conteúdo

4.1 Introdução

Como se referiu nos capítulos anteriores, os algoritmos de pesquisa exaustiva são os que conduzem à solução óptima, apresentando uma estrutura de processamento regular que simplifica o projecto e a implementação de circuitos dedicados para a estimação de movimento em tempo real.

Contudo, e tal como foi descrito na secção 2.5, este tipo de processamento envolve um número de operações bastante elevado. Considerando uma imagem com $N_h \times N_v$ pixels, segmentada em macroblocos de $N \times N$ pixels, e uma área de pesquisa constituída por $(2p+N) \times (2p+N)$ pixels, são necessárias um total de $3(2p+1)^2 \times (N_h \times N_v)$ operações para processar uma imagem completa (ver equação 2.9). Se se considerar um ritmo de f_F imagens por segundo, conclui-se que o processador dedicado terá de executar um total de $3(2p+1)^2 (N_h \times N_v) f_F$ operações por segundo.

Para além disso, é possível constatar também que este tipo de processamento envolve uma taxa de transferência de dados bastante elevada. Para uma área de pesquisa com $(2p+N)^2$ pixels e um macrobloco de referência de N^2 pixels, representados por uma palavra binária de b bits, verifica-se que para processar f_F imagens por segundo será necessária uma taxa de transferência de dados de entrada (r_{in}) da ordem de:

$$r_{in} = b.((2p+N)^2 + N^2).\frac{N_h \times N_v}{N^2}.f_F \tag{4.1}$$

bits/s. Admitindo $p \approx N$, pode-se considerar um valor aproximado para a expressão anterior de:

$$r_{in} \approx 10.b.N_h.N_v.f_F \tag{4.2}$$

valor que, tal como o ritmo de operações executadas, é independente da dimensão do macrobloco considerado (N).

Quanto ao ritmo de dados de saída, considera-se como resultado final do processamento o par de valores que constituem a componente horizontal e vertical do vector de movimento, calculado a partir de um conjunto de $(2p+1)^2$ macroblocos candidatos possíveis. Cada uma destas componentes é, então, representada através de uma palavra binária constituída por $\log_2(2p+1)$ bits. Conclui-se, assim, que será necessário uma taxa de transferência de saída (r_{out}) da ordem de:

$$r_{out} = 2.\log_2(2p+1).\frac{N_h \times N_v}{N^2}.f_F \tag{4.3}$$

Os elevados ritmos de operação e de transferência de dados requeridos têm levado, nos últimos anos, à proposta de arquitecturas com funcionamento em *pipeline* do tipo sistólico, para a realização de estimadores de movimento. Este tipo de arquitecturas,

regulares e com um funcionamento completamente síncrono, adequam-se às características do processamento associado à pesquisa exaustiva por emparelhamento de blocos [19, 9, 20].

As arquitecturas sistólicas são baseadas em aglomerados de elementos processadores relativamente simples e localmente interligados, que permitem obter taxas de computação e de transferência de dados muito elevadas sem recorrer a barramentos globais para a transferência dos dados entre os vários elementos processadores. A aceleração do processamento aumenta, em geral, proporcionalmente com o número de elementos processadores. O grau de concorrência explorado resulta, muitas vezes, de um compromisso entre o desempenho e os custos de *hardware*, nomeadamente, no que se refere à área de semicondutor necessária para implementar o circuito.

Nas secções seguintes será apresentado um conjunto de arquitecturas para a implementação de processadores sistólicos dedicados para pesquisa exaustiva. As arquitecturas, propostas ao longo dos últimos anos, podem obter-se a partir da descrição do algoritmo na forma de grafo de dependências (DG[1]) e de diferentes funções de transposição para estruturas sistólicas [21]. As diferentes arquitecturas são comparadas quanto à rapidez de cálculo e custos computacionais de implementação.

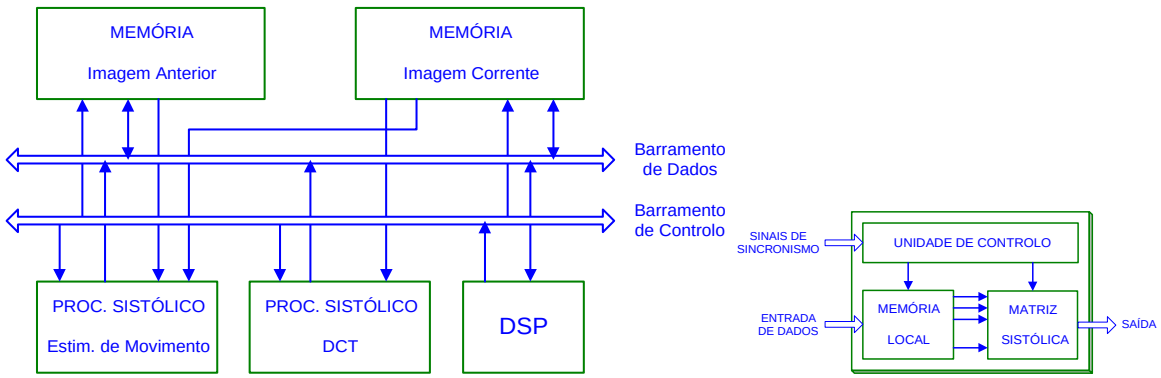
4.2 Processadores sistólicos

Ao longo dos últimos anos, várias arquitecturas de processadores sistólicos para estimação de movimento têm sido propostas por diversos autores [19, 9, 22, 20, 23], utilizando o algoritmo de emparelhamento de blocos descrito nas secções anteriores. Por serem dedicados, este tipo de processadores apresenta uma estrutura, de um modo geral, pouco flexível, efectuando apenas o conjunto restrito de operações para o qual foram projectados. No entanto, permite atingir ritmos de processamento bastante elevados, necessários para o processamento em tempo real.

Alguns autores têm proposto sistemas híbridos, combinando estruturas sistólicas, de elevado desempenho mas pouco flexíveis, com processadores digitais de sinal (DSP) programáveis, utilizados para efectuar blocos condicionais e partes menos regulares dos algoritmos. Um exemplo deste tipo de sistemas encontra-se ilustrado na figura 4.1, onde, para além do processador digital de sinal, se pode observar a presença de dois processadores dedicados sistólicos: um para efectuar a estimação de movimento e outro para calcular a transformada discreta de coseno (DCT).

Na figura 4.1, pode-se identificar um outro tipo de elemento do circuito, designado

[1] Do Inglês *Dependence Graph*.



(a) Arquitectura de um sistema híbrido.

(b) Estrutura interna de um processador sistólico.

Figura 4.1: Diagrama de blocos de um sistema de processamento híbrido, composto por dois processadores sistólicos e um DSP.

por memória local. Estes circuitos de memória são indispensáveis para tirar partido das características dos processadores dedicados sistólicos, garantindo um fluxo de dados de entrada e de saída constante e a um ritmo elevado. O acesso directo a memórias externas pelo processador apresenta desvantagens evidentes, devido ao elevado tempo de acesso e a conflitos que podem resultar da necessidade de efectuar acessos simultâneos à memória, resolvidos normalmente através de ciclos de espera (*wait states*) indesejáveis.

Verifica-se, também, muito frequentemente que, embora os dados correspondentes ao pixels dos macroblocos a processar pertencam a zonas bem localizadas da imagem, o mesmo não se verifica em relação à sua localização na memória física, correspondendo, muitas vezes, a endereços não contíguos do espaço de endereçamento. Como consequência, torna-se muitas vezes fundamental a presença no sistema de um bloco, cuja principal função é a de gerar os diversos endereços necessários à obtenção do fluxo de dados regular, indispensável a um funcionamento eficiente do processador sistólico. Para aumentar o desempenho dos sistemas de codificação, têm também sido utilizadas memórias locais do tipo *cache*, mais rápidas e de menor dimensão do que a memória externa, para se reduzir ao máximo o número de acessos à memória externa, que é, em geral, maior e de acesso mais lento. Muito frequentemente, este tipo de memórias rápidas apresenta, apenas, a dimensão necessária para armazenar os dados correspondentes ao macrobloco de referência e à área de pesquisa sob processamento [24]. T. Komarek e P. Pirsch apresentaram um estudo bastante detalhado [19] sobre o impacto dos tempos de acesso à memória no período do ciclo de funcionamento dos estimadores de movimento sistólicos.

4.3 Mapeamento de algoritmos em estruturas sistólicas

O mapeamento de algoritmos em processadores sistólicos é, em geral, efectuado utilizando um procedimento sistemático, semelhante ao descrito por Kung [19]. Segundo esta metodologia, o algoritmo começa por ser decomposto nas suas operações elementares sendo, de seguida, convertido numa sequência de operações de atribuição única. Assim, considerando um dado algoritmo com n graus de liberdade, obtém-se desta decomposição uma nova formulação definida num espaço n-dimensional. Esta nova formulação pode ser, então, descrita através de um grafo de dependências (DG), composto por nós (correspondentes às operações básicas), e por arcos (interligações) que expressam as dependências entre a computação nos vários nós. Os diferentes tipos de nós que compõe este grafo dependerão, assim, das várias operações que definem o algoritmo.

O mapeamento do grafo de dependências numa estrutura sistólica pode, muitas vezes, ser feito de uma forma quase directa, substituindo cada nó pelo elemento processador correspondente e cada arco por um barramento de dados adequado. No entanto, dadas as características das tecnologias actuais de integração em muito larga escala (VLSI[2]) actuais, apenas arquitecturas definidas segundo uma ou duas dimensões apresentam características com interesse prático. Assim, mapeamentos mais eficientes, onde cada elemento processador possui um nível de utilização optimizado, através da execução de operações correspondentes a múltiplos nós do grafo numa política de *time-share*, podem apresentar vantagens significativas. Tal é possível através da definição de um agendamento temporal[3] adequado e da aplicação de técnicas de projecção [21], obtendo-se, então, um grafo de fluxo de sinal (SFG[4]) definido segundo um espaço com dimensão menor ou igual à do grafo de dependências inicial. Algoritmos definidos em espaços de dimensão superior a três devem ser mapeados utilizando técnicas do tipo multi-projecção, para obter processadores com estruturas bidimensionais ou unidimensionais.

Terminado este procedimento, pode ser necessário aplicarem-se técnicas de sistolização e de escalamento temporal, através da introdução e transferência de vários elementos de atraso nos vários arcos do grafo, de forma a garantir a existência de, pelo menos, uma unidade de atraso entre cada nó de processamento. Elimina-se, assim, a existência de possíveis linhas de distribuição globais obtidas na sequência de etapas anteriores. O processador sistólico pretendido obtém-se de uma forma directa do grafo de fluxo de sinal obtido, substituindo cada nó pelo elemento processador correspondente e cada elemento de atraso por um registo de *pipeline*. O processador final é constituído por uma matriz

[2]Do Inglês *Very Large Scale Integrated circuits*.
[3]Do Inglês *time scheduling*.
[4]Do Inglês *Signal Flow Graph*.

de elementos processadores, separados por vários registos de *pipeline*. O período de funcionamento do *pipeline* é determinado pelo máximo tempo de atraso registado no conjunto de elementos processadores.

O processador que se obtém por transposição do grafo de dependências depende das funções de transposição e agendamento utilizados. Podem obter-se processadores com características distintas, sendo, por isso, necessário proceder a uma selecção da solução mais adequada para a aplicação, tendo, também, em atenção as características da tecnologia.

4.4 Arquitecturas sistólicas para algoritmos de pesquisa exaustiva

Conforme se ilustra na figura 3.1, o algoritmo de pesquisa exaustiva para estimação de movimento apresenta um total de 4 graus de liberdade, a que corresponde um DG num espaço 4-D, definido pelos versores ortogonais: $(\vec{u}, \vec{v}, \vec{c}, \vec{l})$. Estas quatro direcções podem ser agrupadas duas a duas, de acordo com a interpretação geométrica do espaço sob processamento: o espaço definido pelo par de versores $(\vec{u}, \vec{v})$ corresponde ao conjunto de pixels que constituem o macrobloco de referência (imagem actual) e o espaço definido pelos versores $(\vec{c}, \vec{l})$ correspondente ao espaço de pesquisa (imagem anterior).

Na figura 4.2 apresenta-se o grafo de dependências tridimensional correspondente ao algoritmo da figura 3.1 considerando l constante, isto é, reduzindo a representação ao espaço definido pelos versores $(\vec{u}, \vec{v}, \vec{c})$. Desta forma, efectua-se a procura do macrobloco que apresenta maior medida de similaridade ao longo do conjunto de macroblocos candidatos presentes numa linha da área de pesquisa, indexados pelo valor da coordenada c. O resto do algoritmo define-se, assim, segundo um espaço com uma única dimensão, correspondente ao valor da coordenada l. Desta forma, os vários vectores de movimento com maior medida de similaridade, correspondentes a cada uma das linhas de pesquisa, terão de ser comparados entre si, para encontrar o vector correspondente ao máximo valor de similaridade de entre o conjunto definido por todos os macroblocos na área de pesquisa.

No grafo da figura 4.2, o símbolo **AD** representa o cálculo do valor absoluto da diferença entre um dado pixel do macrobloco de referência e o pixel do macrobloco candidato correpondente: $|R(u,v) - S(c+u, l+v)|$. Os valores absolutos calculados ao longo de uma coluna do macrobloco são, então, somados nos vários nós representados com o símbolo **A**. Por fim, a comparação e a selecção do vector de movimento correspondente ao macrobloco candidato com maior similaridade é efectuada nos nós representados com o

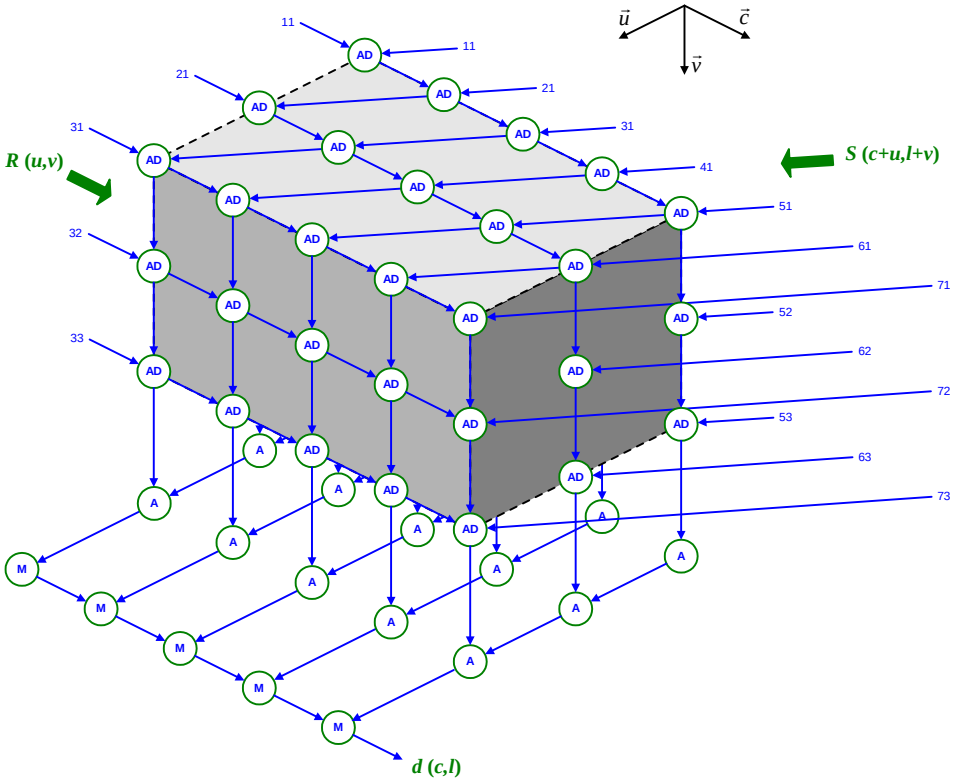


Figura 4.2: Grafo de dependências do algoritmo de estimação de movimento com pesquisa exaustiva, para $N = 3$ e $p = 2$.

símbolo **M**.

Como se referiu na secção anterior, apenas as estruturas unidimensionais e bidimensionais apresentam características com algum interesse prático, tendo em vista a sua implementação em *hardware*. Torna-se, assim, necessário efectuar projecções para reduzir o número de graus de liberdade do grafo de dependências apresentado. De igual forma, como o número de elementos de atraso presentes nos vários caminhos possíveis do circuito (definidos entre uma dada entrada e o porto de saída) deve ser igual, verifica-se que os dados correspondentes a entradas mais próximas da saída do circuito deverão entrar atrasados relativamente aos dados correspondentes a portos de entrada mais distantes do porto de saída do resultado final.

Nas secções seguintes, apresentam-se várias estruturas obtidas a partir de uma ou duas projecções do grafo de dependências segundo direcções distintas. Como se poderá constatar, adoptam-se, preferencialmente, vectores de projecção paralelos aos três versores, pois dão origem a estruturas com entradas e saídas de dados regulares e envolvendo um número mínimo de elementos processadores. Analisam-se, também, as principais características das estruturas apresentadas, nomeadamente, o tempo de processamento e os custos de *hardware*. Salvo indicação em contrário, e sem prejuízo de generalização,

considerar-se-ão nas estruturas apresentadas blocos de referência de 3×3 pixels ($N = 3$) e áreas de pesquisa com deslocamentos máximos em cada direcção dados por $p = 2$.

4.4.1 Estrutura Tipo 1 ou AB2

Efectuando uma única projecção do grafo de dependências segundo o vector $\vec{d} = (0, 0, 1)$, correspondente a projectar os diferentes nós segundo a direcção perpendicular ao plano definido pelos versores $(\vec{u}, \vec{v})$ num único nó, obtém-se a estrutura tradicionalmente designada por AB2, ou Tipo 1 [9], apresentada na figura 4.3. O vector de *scheduling* utilizado é o vector $\vec{s} = (1, 1, 0)$ e os elementos de atraso (registos de *pipeline*) foram representados com o símbolo "•". Nesta estrutura, a cada elemento processador é atribuído o valor de cada um dos $N \times N$ pixels do macrobloco de referência, pelo que a medida de similaridade do macrobloco candidato é, assim, efectuada de uma forma concorrente pelos N^2 elementos processadores. A adição dos valores parciais calculados por estes elementos processadores é efectuada pelos vários somadores localizados na extremidade inferior da figura, efectuando-se, depois, a comparação dos valores no elemento processador localizado junto do porto de saída.

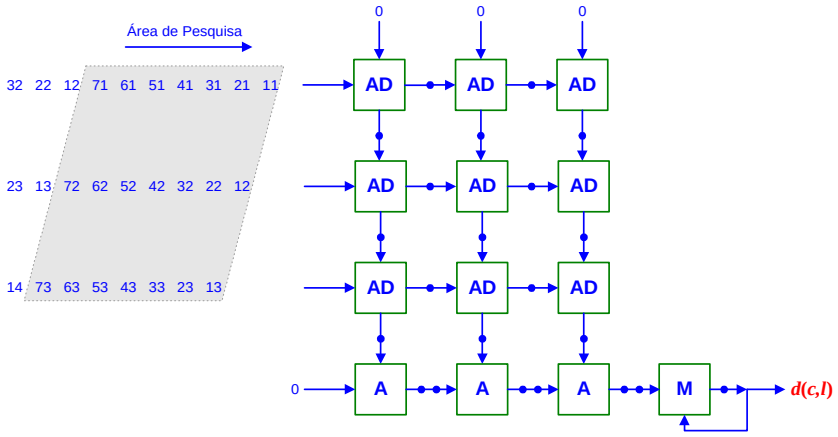


Figura 4.3: Estrutura AB2.

Nesta estrutura, a entrada dos valores dos pixels correspondentes ao espaço de pesquisa, $S(c+u, l+v)$, permite o processamento de linhas de pesquisa consecutivas de uma forma sequencial. Desta forma, o processo de procura do macrobloco candidato com maior medida de similaridade, de entre o conjunto composto pelos $(2p+1)$ macroblocos candidatos que compõe cada linha de pesquisa, requer um total de $(2p + N)$ ciclos de relógio, pelo que o número total de ciclos necessários para processar cada macrobloco de referência é de:

$$c_{AB2} = (2p + 1).(2p + N) \tag{4.4}$$

Esta estrutura apresenta, como principal vantagem, o facto de os pixels correspondentes ao macrobloco de referência, $R(u, v)$, permanecerem constantes em cada elemento processador ao longo de todo o processo de procura, dentro de uma determinada área de pesquisa pesquisa. Para além disso, a forma de entrada dos dados correspondentes ao espaço de pesquisa permite a utilização de estratégias não regulares de procura, permitindo, por exemplo, comparar apenas os macroblocos candidatos localizados a uma determinada distância do macrobloco de referência.

Estrutura Tipo 1 - Vos

Vos e Stegherr [9, 25] propuseram uma implementação deste tipo de estruturas que, através da utilização de um conjunto de $2\,(2p \times N)$ registos auxiliares, permite obter um nível de eficiência da utilização dos N^2 elementos processadores muito próximo dos 100% (ver figura 4.4).

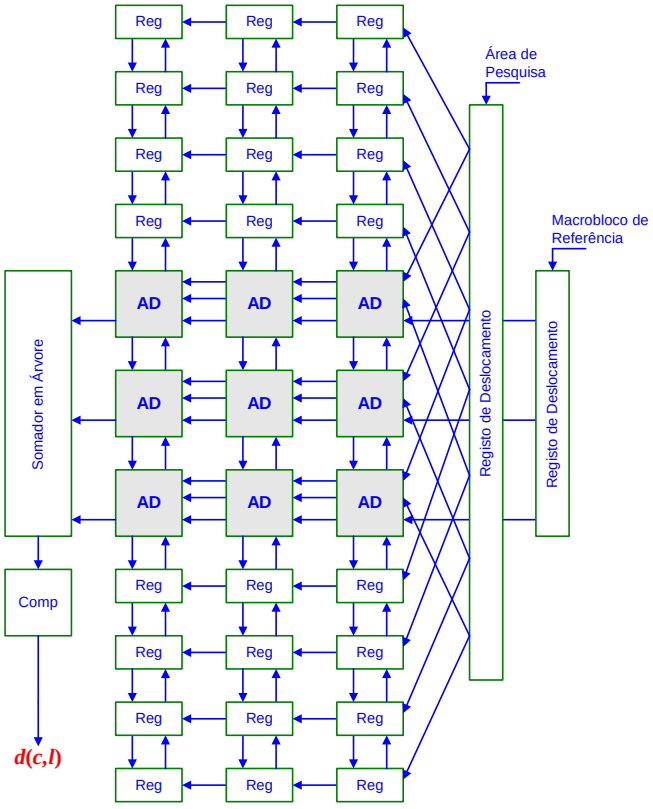


Figura 4.4: Estrutura Tipo 1 proposta por Vos [9].

O princípio de funcionamento baseia-se num esquema de processamento em que o fluxo de dados da área de pesquisa circula ao longo dos vários elementos processadores e dos $2\,(2p \times N)$ registos auxiliares, segundo um esquema do tipo "zig-zag" descendente.

Assim, evita-se o desperdício de ciclos de relógio no preenchimento da estrutura entre cada linha de pixels. As várias medidas de similaridade calculadas nos N^2 elementos processadores são, tal como na estrutura AB2, acumuladas ao longo de cada linha de elementos processadores e adicionadas, por fim, no somador em árvore de N entradas (ver figura 4.4).

De todas as arquitecturas dedicadas para estimação de movimento propostas ao longo dos últimos anos, reconhece-se a superioridade desta arquitectura, relativamente à eficiência de utilização dos recursos de *hardware* e tempo de processamento. Como consequência, optou-se por utilizar esta estrutura como base do trabalho de investigação agora apresentado, procedendo-se a uma explicação mais detalhada do seu funcionamento no capítulo seguinte.

Para atenuar o efeito do ruído presente em sequências de imagens reais, que dá muitas vezes origem ao aparecimento de vectores de movimento de elevada amplitude em sequências com imagens paradas, Vos e Stegherr [9] propuseram, também, uma alteração do critério de erro utilizado, penalizando o aparecimento de vectores de movimento de elevada amplitude:

$$d'(c,l) = d(c,l) + \alpha.\sqrt{c^2+l^2} \tag{4.5}$$

$$d'(c_M, l_M) = \arg\max_{(c,l)}\{\, d'(c,l) \;:\; -p \le c,l < p \,\} \tag{4.6}$$

onde $d(c,l)$ é dada por 2.1. O parâmetro α deverá ser cuidadosamente escolhido para não favorecer, de forma indesejada, os vectores de movimento de menor amplitude.

Estrutura Tipo 1 - Hsieh

Hsieh e Lin [22] propuseram uma variante da arquitectura Tipo 1 ou AB2 descrita na secção 4.4.1, cuja principal característica é o facto de tanto os dados correspondentes aos pixels do macrobloco de referência como os correspondentes à área de pesquisa serem introduzidos no processador de uma forma sequencial, por uma única entrada. O esquema correspondente a esta estrutura encontra-se apresentado na figura 4.5.

O processador proposto é composto por três blocos principais: uma matriz constituída por elementos processadores e registos de deslocamento com ligações sistólicas, um somador paralelo com estrutura tipo "árvore" e um bloco comparador e selector do macrobloco candidato com maior medida de similaridade.

Assim, os dados de entrada correspondentes aos pixels da área de pesquisa começam por entrar no porto existente no primeiro elemento processador de uma forma sequencial e são, então, deslocados para a direira através dos elementos processadores seguintes ou dos registos de deslocamento. Cada um destes registos encontra-se, por sua vez,

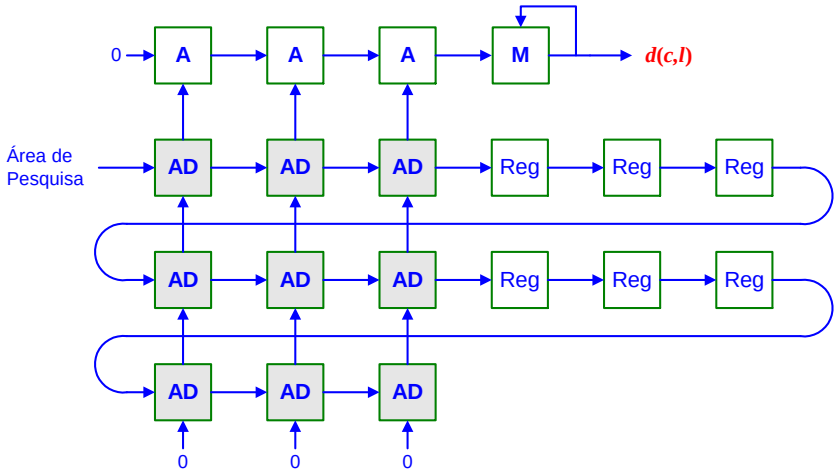


Figura 4.5: Estrutura Tipo 1 proposta por Hsieh [22].

ligado ao primeiro elemento processador da linha seguinte, pelo que toda a estrutura apresenta uma forma semelhante a um processador linear unidimensional, com os registos de deslocamento intercalados entre séries de N elementos processadores.

As somas dos valores absolutos das diferenças entre os pixels do macrobloco de referência e os pixels correspondentes aos macroblocos candidatos da área de pesquisa são efectuadas, em paralelo, pelos vários elementos processadores que constituem a matriz sistólica. Os vários resultados parciais são enviados para o somador em árvore (A), onde é calculado o valor correspondente à medida de similaridade do macrobloco candidato correspondente. Este valor é, então, entregue à unidade de comparação e selecção do macrobloco correspondente à maior medida de similaridade, completanto, assim, o procedimento de pesquisa. Desta forma, efectua-se a comparação correspondente a um macrobloco candidato em cada ciclo de relógio.

A pincipal vantagem deste circuito reside, precisamente, no facto de, apesar de apresentar um funcionamento paralelo, apenas necessitar do novo valor de um pixel da área de pesquisa em cada ciclo de relógio, evitando o uso de um número elevado de portos de interligação com o sistema exterior. Para além disso, apresenta também algumas características de modularidade e flexibilidade no que diz respeito à dimensão da área de pesquisa. De facto, através da simples alteração do número de registos de deslocamento presentes em cada linha do processador, é possível alterar o número de macroblocos candidatos envolvidos na pesquisa.

Desta forma, como facilmente se verifica, o processo de procura do macrobloco candidato com maior medida de similaridade de entre o conjunto composto pelos $(2p+1)^2$ macroblocos candidatos possíveis requer um total de $c_{Hsieh} = (2p+N)^2$ ciclos de relógio.

4.4.2 Estrutura AB1

A estrutura AB1, apresentada na figura 4.6, resulta de uma segunda projecção da estrutura AB2 segundo o vector $\vec{d} = (1, 0)$. De forma semelhante, esta estrutura pode ser derivada do grafo de dependências inicial, apresentado na figura 4.2, usando uma dupla projecção segundo os vectores $\vec{d_1} = (0, 0, 1)$ e $\vec{d_2} = (1, 0)$.

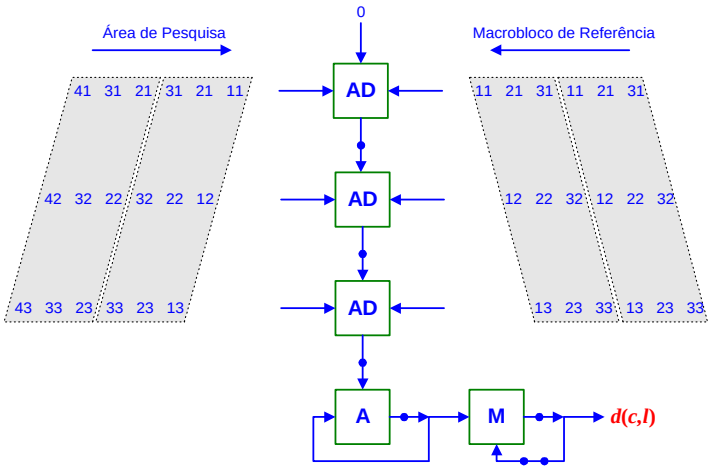


Figura 4.6: Estrutura AB1.

Devido à estrutura unidimensional desta arquitectura, o processamento de todos os $(2p + 1)^2$ macroblocos candidatos envolve um total de $N.(2p + 1)^2$ ciclos de relógio. Contudo, como se torna necessário efectuar o preenchimento da coluna de elementos processadores com os valores correspondentes ao início da linha de pesquisa seguinte, $S\,(c + u, (l + 1) + v)$, no final de cada linha, são necessários $N - 1$ ciclos de relógio extra, a que não corresponde qualquer valor válido do espaço de procura. Desta forma, o número total de ciclos de relógio necessários para processar cada macrobloco de referência é de:

$$c_{AB1} = (2p + 1).N.(2p + 1 + N - 1) \tag{4.7a}$$

$$= N.(2p + 1).(2p + N) \tag{4.7b}$$

A propósito desta estrutura linear, Vos e Stegherr [9, 25] propuseram um esquema de bancos de memória que permite, de uma forma regular e recorrendo a um circuito de controlo bastante simples, introduzir as palavras binárias correspondentes aos pixels do macrobloco de referência e da área de pesquisa sob comparação. O circuito proposto permite, ainda, efectuar o pré-carregamento dos dados correspondentes ao macrobloco de referência seguinte, evitando, assim, a utilização de ciclos de relógio extra no final do processamento de cada macrobloco.

4.4.3 Estrutura AS2

A estrutura AS2, apresentada na figura 4.7, resulta do grafo de dependências inicial, apresentado na figura 4.2, usando uma única projecção segundo o vector $\vec{d} = (0, 1, 0)$. Esta estrutura corresponde a projectar os diferentes nós do grafo segundo o plano definido pelos versores $(\vec{u}, \vec{c})$. O vector de *scheduling* utilizado é o vector $\vec{s} = (1, 0, 1)$. Neste caso, a posição dos vários elementos processadores corresponde à posição dos $N.(2p+1)$ pixels que constituem uma linha de pesquisa. Assim, a cada coluna desta matriz estará associado um valor distinto de deslocamento do macrobloco de referência. As várias medidas de similaridade são obtidas através da soma dos valores calculados ao longo de cada coluna da matriz. Estes valores parciais são então acumulados e comparados pelos elementos processadores localizados na extremidade inferior de cada coluna.

Desta forma, como o processo de comparação de cada um dos $(2p+1)$ macroblocos candidatos possíveis, que compõe cada linha de pesquisa, requer um total de N ciclos de relógio, o número total de ciclos necessários para processar cada macrobloco de referência é de:

$$c_{AS2} = N.(2p+1) \tag{4.8}$$

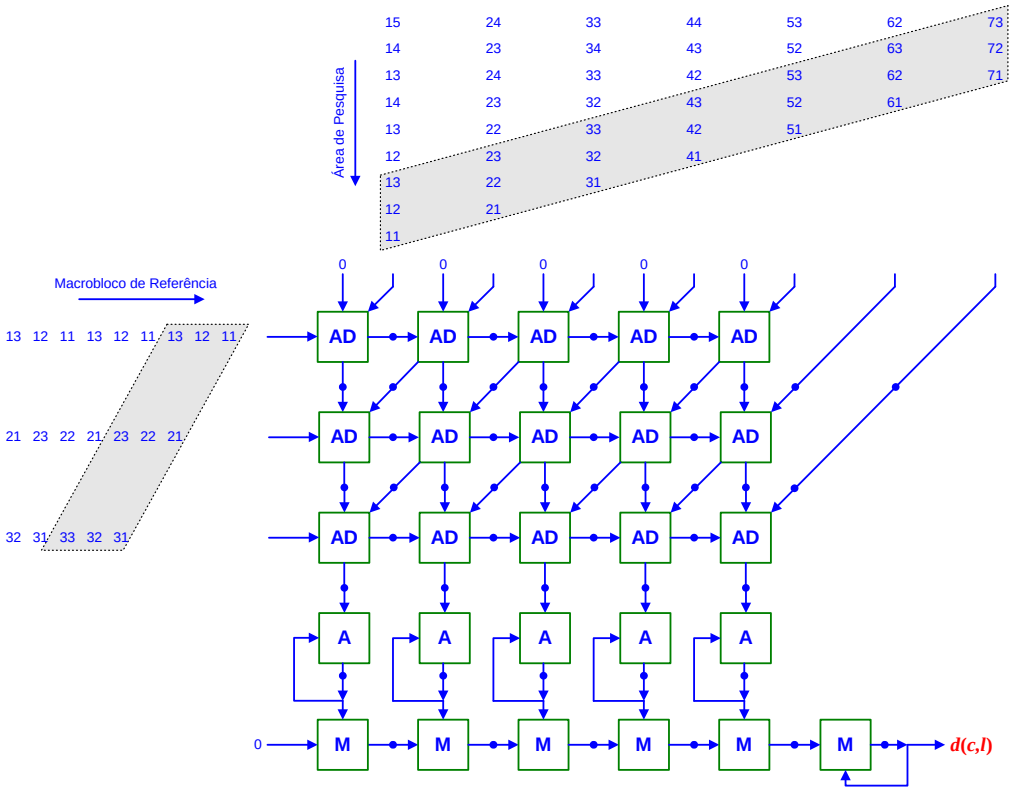


Figura 4.7: Estrutura AS2.

4.4.4 Estrutura AS1

Tal como a estrutura AB1, é possível também obter uma estrutura linear com uma dimensão correspondente a uma linha da área de pesquisa. Esta estrutura, apresentada na figura 4.8, caracteriza-se por ter uma estrutura de dados sequencial, tanto para o macrobloco de referência como para a área de pesquisa. Contudo, para garantir um fluxo regular de dados, a entrada correspondente ao macrobloco de referência apresenta alguns valores correspondentes a dados não válidos, representados por pontos na figura 4.8.

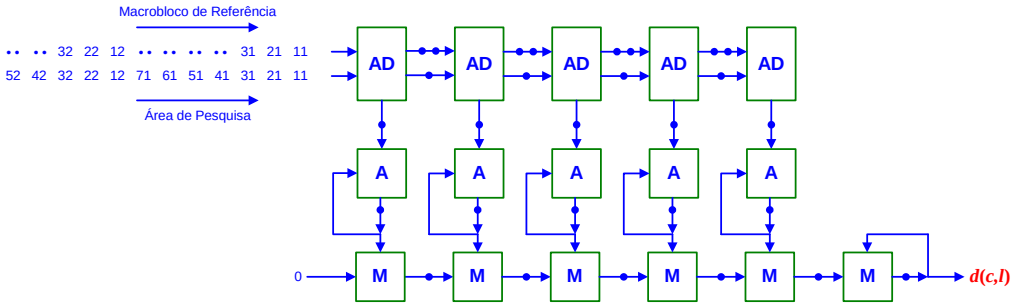


Figura 4.8: Estrutura AS1.

Neste caso, o processamento de cada linha de macroblocos requer um total de $N(2p+N)$ ciclos, pelo que o processamento dos $(2p+1)^2$ macroblocos candidatos envolve um número de ciclos de relógio dado por:

$$c_{AS1} = N.(2p+N).(2p+1) \tag{4.9}$$

Yang *et al.* [23] apresentaram duas propostas de processadores dedicados semi-sistólicos utilizando este tipo de estrutura, que permitem o processamento, em paralelo, de uma linha de pesquisa. As propostas apresentadas permitem, ainda, a utilização de diferentes dimensões do macrobloco e da área de pesquisa, recorrendo a aglomerados de processadores em *pipeline*. Em qualquer das situações, a entrada dos dados correspondentes aos pixels da área de pesquisa é efectuada sempre através de dois portos de entrada. Estes autores propuseram, ainda, processadores dedicados com estruturas semelhantes para estimar movimento com precisão de meio pixel e de um quarto de pixel.

4.4.5 Estrutura Tipo 2

Vos e Stegherr [9] propuseram, ainda, uma outra estrutura a que designaram de Tipo 2, em que é invertida a ordem de execução dos ciclos do algoritmo apresentado na figura 3.1. Com esta inversão, consideram-se agora como ciclos internos os ciclos indexados com as

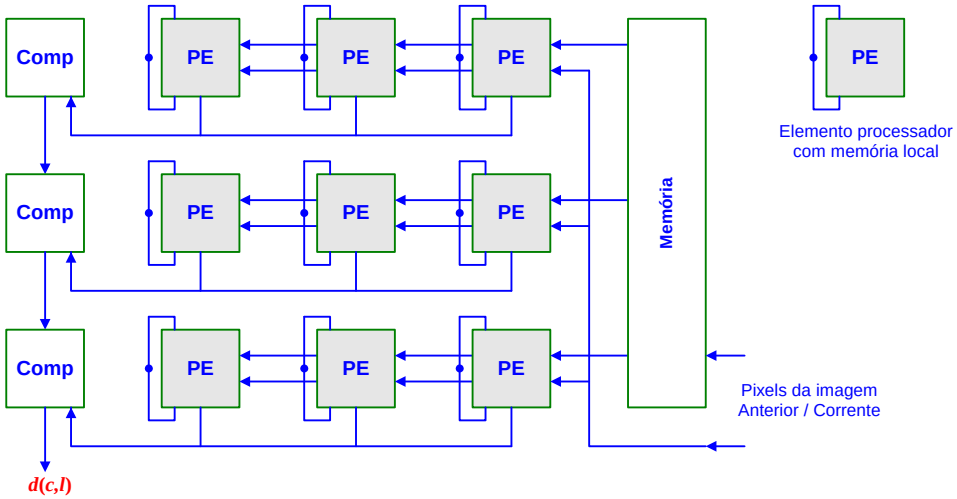


Figura 4.9: Estrutura Tipo 2.

variáveis c e l. Como consequência, em cada ciclo de execução será processado um dado pixel $R(u, v)$ do macrobloco de referência, e todos os pixels da área de pesquisa que têm de ser comparados com este pixel deverão estar disponíveis durante este ciclo.

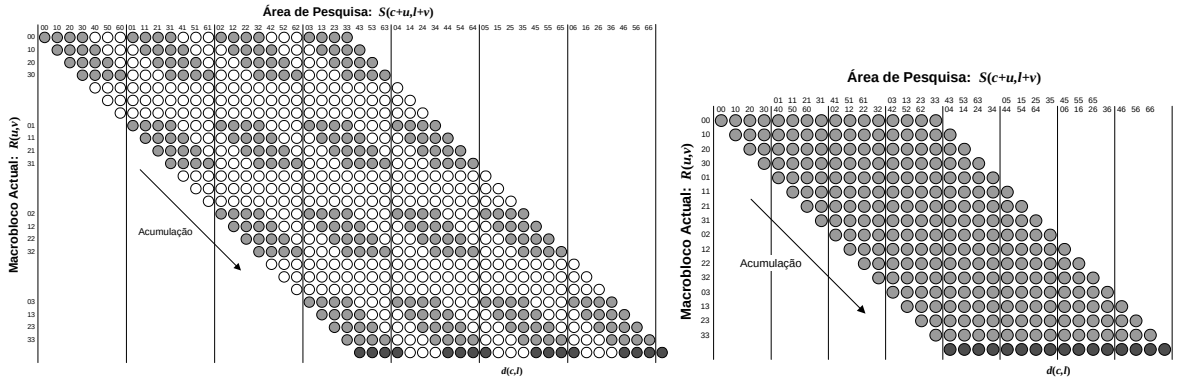
Assim, a cada um dos $(2p+1)^2$ elementos processadores corresponderá um dado vector de movimento. Idealmente, no final de N^2 ciclos de relógio todos os pixels do macrobloco de referência terão sido processados, pelo que estarão disponíveis $(2p + 1)^2$ medidas de similaridade distintas, correspondentes a cada um dos potenciais vectores de movimento.

Na figura 4.9 apresenta-se uma estrutura deste tipo, para o caso particular de se ter uma área de pesquisa com um deslocamento máximo em cada direcção dado por $p = 1$, sendo, por isso, constituída por 3×3 elementos processadores.

Este tipo de estrutura apresenta vantagens evidentes em casos onde o número de vectores de movimento a considerar não é muito elevado (áreas de pesquisa pequenas). Nesses casos, quando comparadas com estruturas do Tipo 1, este tipo de estruturas requer um número de elementos processadores e áreas de implementação em silício significativamente menores, mesmo em situações com ritmos de amostragem elevados.

4.4.6 Estruturas propostas por Chang [20]

Chang *et al.* [20] propuseram, também, estruturas de processadores para estimação de movimento utilizando, porém, uma notação diferente da transformação aqui descrita para a derivação de processadores sistólicos, utilizada por Komarek [19] e por Kung [21]. O grafo de dependências inicial, definido segundo as quatro direcções ortogonais dadas pelos versores $(\vec{u}, \vec{v}, \vec{c}, \vec{l})$, é subdividido em sub-grafos bidimensionais, considerando dois dos



(a) Sem optimização da taxa de utilização dos elementos processadores.

(b) Compressão da estrutura obtida para aumentar a taxa de utilização.

Figura 4.10: Estruturas propostas por Chang [20].

índices como índices da estrutura bidimensional de elementos processadores e os restantes dois como índices internos das várias operações envolvidas nos elementos processadores. Assim, e assumindo a equivalência de duas estruturas definidas pelos índices $(\vec{a}, \vec{b})$ e $(\vec{b}, \vec{a})$, existe um conjunto constituído por $C_2^4 = 6$ estruturas distintas possíveis, algumas das quais já descritas neste capítulo.

Na sua proposta, Chang distribuiu repetidamente cada uma das projecções $(\vec{u}, \vec{v})$ assim obtidas (*slices*) num espaço bidimensional $(\vec{c}, \vec{l})$, através de uma operação que designou de *tiling* (ver figura 4.10(a)). Apresentou, também, propostas no sentido de obter taxas de utilização dos elementos processadores o mais elevadas possível. Para isso, começou por classificá-los de acordo com a sua actividade como OP (*operation*) e NOP (*no-operation*). De seguida, tentou eliminar a presença de elementos do tipo NOP através da sua sobreposição com os elementos do tipo OP, donde concluiu a necessidade de o sistema apresentar várias entradas de dados para os pixels da área de pesquisa, para evitar a existência deste tipo de elementos processadores (ver figura 4.10(b)).

4.4.7 Comparação da eficiência das várias estruturas

As estruturas descritas nas secções anteriores podem ser comparadas em relação a diferentes aspectos, tais como: a dimensão, o número de elementos processadores utilizados e o número total de ciclos de relógio necessários para o cálculo de um vector de movimento. De entre estas medidas, o tempo de processamento é, na prática, o parâmetro determinante para efeitos de comparação de processadores vocacionados para aplicações orientadas para o processamento em tempo real. Assim, convém começar por recordar

que, tal como se referiu na secção 2.5, para efectuar a estimação de um vector de movimento utilizando a medida de dissemelhança SAD, é necessário um total de $3N^2(2p+1)^2$ operações aritméticas. Contudo, se se considerar um esquema de um processador genérico com funcionamento em *pipeline*, em que em cada ciclo de relógio é possível efectuar uma subtracção, um cálculo de valor absoluto, uma adição, uma comparação e uma operação de selecção, o número de ciclos necessário para efectuar a estimação do vector de movimento, pode ser reduzido para $N^2(2p+1)^2$. Para além disso, se a arquitectura de processamento for do tipo paralelo, com N_{pe} elementos processadores, o número de ciclos de relógio necessários poderá, então, ser reduzido para cerca de $c = N^2(2p+1)^2/N_{pe}$ se, em cada instante, todos os elementos processadores se encontrarem a realizar trabalho útil. Note-se que, na prática, o número de elementos processadores, N_{pe}, representa o número de elementos representados com o símbolo "**AD**" nos grafos de fluxo de sinal apresentados nas secções anteriores.

Considerando T o período de relógio, o tempo de processamento necessário para a estimação de um vector de movimento será dado por:

$$t_v = T \times c = T \times \frac{N^2(2p+1)^2}{N_{pe}} \tag{4.10}$$

Admitindo uma imagem formada por $N_h \times N_v$ pixels, em que cada macrobloco é constituído por $N \times N$ pixels, o número total de vectores de movimento a estimar em cada imagem será dado por:

$$N_{vf} = \frac{N_h \times N_v}{N^2} \tag{4.11}$$

pelo que o tempo de estimação total será de:

$$t_{vf} = t_v \cdot \frac{N_h \times N_v}{N^2} \tag{4.12a}$$

$$= T \cdot (2p+1)^2 \cdot \frac{N^2}{N_{pe}} \cdot \frac{N_h \times N_v}{N^2} \tag{4.12b}$$

$$= T \cdot (2p+1)^2 \cdot \frac{N_h \times N_v}{N_{pe}} \tag{4.12c}$$

que é independente da dimensão do macrobloco utilizada (N).

Definindo-se R como sendo o número médio de ciclos de relógio por pixel da imagem processada:

$$R = \frac{T_{vf}}{T} \cdot \frac{1}{N_h \times N_v} \tag{4.13}$$

Estrutura	N_{pe}	c	R
SinglePE	1	$N^2 \cdot (2p+1)^2$	$(2p+1)^2 \approx 4N^2$
AB1	N	$N \cdot (2p+1) \cdot (2p+N)$	$\frac{(2p+1)\cdot(2p+N)}{N} \approx 6N$
AS1	$2p+1$	$N \cdot (2p+N) \cdot (2p+1)$	$\frac{(2p+N)\cdot(2p+1)}{N} \approx 6N$
Tipo 1 - AB2	$N \times N$	$(2p+1) \cdot (2p+N)$	$\frac{(2p+1)\cdot(2p+N)}{N^2} \approx 6$
Tipo 1 - Vos	$N \times N$	$(2p+1)^2$	$\frac{(2p+1)^2}{N^2} \approx 4$
Tipo 1 - Hsieh	$N \times N$	$(2p+N)^2$	$\frac{(2p+N)^2}{N^2} \approx 9$
AS2	$N \times (2p+1)$	$N \cdot (2p+1)$	$\frac{(2p+1)}{N} \approx 2$
Tipo 2	$(2p+1) \times (2p+1)$	N^2	1

Tabela 4.1: Comparação entre as várias estruturas.

obtém-se então o valor:

$$R = \frac{(2p+1)^2}{N_{pe}} \tag{4.14a}$$

$$= \frac{c}{N^2} \tag{4.14b}$$

Na tabela 4.1 apresentam-se os valores de c e de R para cada um dos processadores descritos nas secções anteriores, utilizando, cada um, N_{pe} elementos processadores. Convém notar que estes valores são, na realidade, valores estimados por defeito. De facto, as estimativas efectuadas nas secções anteriores dos vários valores de c não tiveram em linha de conta o tempo necessário para retirar e introduzir os dados dos vários registos do processador sempre que se verifica a mudança de macrobloco de referência. Consequentemente, os valores reais serão, por isso, um pouco superiores. Como base de comparação, consideraram-se, também, os valores correspondentes a uma estrutura hipotética constituída por apenas um elemento processador, a que se designou "SinglePE". Como seria de esperar, o número médio de ciclos de relógio por pixel processado, (R), diminui com o aumento do número de elementos processadores N_{pe}.

Se se considerar a aproximação de que o valor máximo de deslocamento permitido em cada direcção (p) é da ordem do tamanho do macrobloco $(p \approx N)$, o número total de macroblocos candidatos será:

$$(2p+1)^2 \approx (2N+1)^2 \approx 4N^2 \tag{4.15}$$

Neste caso, verifica-se facilmente que, em arquitecturas bidimensionais, em que $N_{pe} = \mathcal{O}(N^2)$, o número médio de ciclos de relógio por pixel (R) será independente do deslocamento máximo (p) e do tamanho do macrobloco (N) (ver tabela 4.1).

Verifica-se, ainda, que algumas estruturas lineares unidimensionais, tais como a AB1, apresentam, como principal vantagem, o facto de todos os elementos processadores se encontrarem acessíveis do exterior, pelo que podem ser carregados directamente a partir do banco de memória. Contudo, tal assunção parte do pressuposto de que é possível obter, do banco de memória, uma largura de banda suficiente para entregar os dados correspondentes aos pixels do macrobloco de referência e da área de pesquisa a todos os elementos processadores da estrutura. De facto, para estruturas com um número elevado de elementos processadores, o número de saídas do banco de memória pode tornar-se incomportável. Para evitar este tipo de problemas, noutras estruturas, tais como na AS1, a entrada de dados efectua-se apenas por um dos elementos processadores, sendo depois entregues aos restantes elementos através de ligações sistólicas. Contudo, neste tipo de estruturas, é muitas vezes necessário introduzir ciclos suplementares para transferir os dados para e do processador, aumentando, por isso, a latência total do circuito e fazendo, assim, diminuir a sua eficiência.

Em contraste, verifica-se que, embora as estruturas bidimensionais sejam constituídas por N ou por $(2p+1)$ vezes mais elementos processadores do que as estruturas unidimensionais, a largura de banda da memória requerida não aumenta muito significativamente. Da mesma forma, verifica-se que a latência destas estruturas apresenta valores bastante semelhantes. Assim, vários autores consideram este tipo de estruturas bidimensionais como o compromisso entre a largura de banda da memória requerida e o número de ciclos de relógio necessários para a transferência de dados entre o processador e o exterior [26].

4.5 Estrutura óptima

Como se referiu no capítulo 1, o bloco de estimação de movimento é, em geral, um dos blocos de processamento mais exigentes num sistema de codificação de vídeo, podendo operar com diferentes ritmos de dados, desde 50 kpps[5] até várias dezenas de Mpps. Existe, por isso, um vasto leque de diferentes aplicações para cada uma das arquitecturas de estimadores de movimento por emparelhamento de blocos descritas neste capítulo. Contudo, o factor fundamental para a escolha de uma determinada arquitectura é, muitas vezes, o *ritmo de pixel*, dependente, em geral, da resolução espacial e temporal da sequência de imagens a codificar. Outros factores relevantes são o *tamanho do macrobloco* considerado, a *dimensão da área de pesquisa* e o *formato de entrada dos dados*. Para além disso, há ainda a considerar outros aspectos de índole mais tecnológica, tais como a *frequência de relógio* máxima permitida pelos elementos processadores utilizando uma

[5]Do Inglês *pixels per second*.

dada tecnologia, a *área* de material semicondutor ocupada ou, de forma semelhante, o *número máximo de transístores* utilizado pelo circuito, e ainda a *potência consumida*. De entre estes factores, convém destacar:

1. *Ritmo de pixel* - trata-se de um factor determinante do nível máximo de reutilização possível das estruturas de *hardware*, que influencia o número máximo de projecções possível do grafo de dependências inicial e, por consequência, o nível de paralelismo e concorrência das operações realizadas. Como se viu anteriormente, a reutilização de elementos processadores, através de múltiplas projecções do grafo de dependências inicial, constitui uma das principais estratégias para reduzir a área de semicondutor requerida pelo circuito.

2. *Dimensão do macrobloco ($\propto N$), e da área de pesquisa ($\propto p$)* - comparando duas estruturas bidimensionais, tais como a AB2, que requer um total de N^2 elementos processadores e cerca de $(2p+1)(2p+N)$ ciclos de relógio para processar um macrobloco; com a estrutura AS2, que requer um total de $N \times (2p+1)$ elementos processadores e um total de $N \times (2p+1)$ ciclos de relógio, verifica-se a existência de um compromisso entre área ocupada e tempo de processamento. É com base neste compromisso que se efectua a escolha entre estruturas bidimensionais ou entre estruturas unidimensionais.

3. *Entrada e armazenamento dos dados de entrada* - este factor determina a forma em como os dados de entrada são apresentados ao processador, quer seja num formato correspondente a uma linha constituída por N_h pixels ou a um bloco de pixels. Este factor irá, de uma forma semelhante, influenciar o funcionamento do bloco gerador de endereços para a memória de imagem.

Em regra, o *ritmo de pixel* e as *dimensões do macrobloco* e da *área de pesquisa* são os parâmetros que mais influenciam a selecção do tipo de arquitectura a utilizar numa dada aplicação. Para ritmos de pixel baixos, até cerca de 1 Mpps, (correspondentes, por exemplo, ao formato CIF com 10 imagens por segundo), as estruturas unidimensionais, com elevados níveis de partilha de elementos processadores, permitem obter processadores de baixo custo, ocupando áreas de semicondutor reduzidas. Pelo contrário, em aplicações onde se torna importante obter taxas de codificação muito elevadas, como por exemplo para codificação de vídeo de alta definição, em que é comum obter débitos da ordem dos 72 a 144 Mpps, verifica-se que apenas as estruturas bidimensionais permitem obter desempenhos capazes de processar a ritmos tão elevados.

4.6 Arquitecturas compostas por multi-processadores

Considerando o sistema de processamento a funcionar a uma frequência de relógio dada por f_{clk}, recebendo dados a uma frequência de pixel dada por f_{pixel} e processando macroblocos de dimensão $N \times N$, é possível obter a seguinte relação entre estas frequências, o número total de pixels a processar (N^2) e o número de ciclos de relógio necessários para efectuar a estimação do vector de movimento (c):

$$c \times \frac{1}{f_{clk}} \leq N^2 \times \frac{1}{f_{pixel}} \tag{4.16a}$$

$$f_{clk} \geq \frac{c \times f_{pixel}}{N^2} \tag{4.16b}$$

Nos casos em que a tecnologia utilizada não permite o funcionamento com frequências de relógio que garantam a condição obtida, a utilização de um único processador mostra-se insuficiente para garantir a estimação correcta dos vectores de movimento em tempo real. Nesses casos, uma das alternativas consiste na agregação de multi-processadores para, assim, se obter o ritmo de processamento desejado. Vos e Schöbinger [24] propuseram arquitecturas constituídas por agregados de estruturas do Tipo 1 (AB2), tanto com funcionamento em paralelo como em *pipeline*.

As estruturas com funcionamento em paralelo foram propostas para resolver problemas relacionados, por exemplo, com a utilização de áreas de pesquisa de dimensões elevadas, efectuado a separação dos pixels que as constituem em sub-bandas com sobreposição [24]. Para além disso, a utilização deste tipo de agregados foi ainda proposta em processadores com funcionamento baseado nos vários algoritmos de pesquisa subóptima descritos na secção 3.3.

No que se refere às estruturas com funcionamento em *pipeline*, estes autores propõem, por exemplo, a sua utilização em sistemas de estimação de movimento baseados em algoritmos com pesquisa hierárquica (ver secção 3.3.3), onde cada processador efectua a pesquisa em cada um dos níveis que compõem o esquema de processamento, fornecendo os resultados parciais então obtidos ao processador seguinte do *pipeline*. Propõem, ainda, a utilização destas estruturas para macroblocos de elevada dimensão, fazendo, assim, a sua sub-divisão em sub-macroblocos a serem atribuidos a cada um dos processadores que compõem o agregado. Contudo, este tipo de estruturas com processamento em série possui o inconveniente de apresentar latências bastante mais elevadas, visto que os dados a processar têm de percorrer toda a cadeia de processadores que constituem o agregado. Para atenuar este tipo de inconvenientes, Vos e Schöbinger propuseram, também, agregados mistos, compostos por estruturas com funcionamento série-paralelo.

Capítulo 5

Arquitectura proposta para estimação de movimento

Conteúdo

5.1 Introdução

Tal como foi referido no final do capítulo anterior, a escolha da arquitectura do processador de estimação de movimento deve ter em consideração as especificações associadas às diferentes aplicações da codificação de vídeo.

No que se refere à implementação propriamente dita, é usual considerarem-se dois parâmetros:

- *Tempo de processamento* (T) necessário para a estimação do vector de movimento e que influenciará, de uma forma directa, a capacidade do processador de efectuar o processamento em tempo real.
- *Área de implementação* (A), como indicador da quantidade de recursos de *hardware* necessária para implementar o circuito e, em consequência, do custo do próprio processador.

Em geral, os esforços de optimização de um dado processador, efectuados pela equipa de desenvolvimento, centram-se na minimização conjunta destas duas medidas, procurando, assim, obter uma arquitectura capaz de efectuar a estimação do vector de movimento no mais curto espaço de tempo e utilizando a menor quantidade de recursos possível. É por este motivo que se tem tornado corrente a utilização de uma outra medida, calculada com base no produto da área de semicondutor requerida e do tempo de processamento, que se designa por produto *Área-Tempo* (AT). Esta medida permite a comparação, de uma forma simples, dos diversos tipos de processadores sendo, por isso, um dos critério de selecção da solução ideal utilizado no desenvolvimento de arquitecturas dedicadas para circuitos de estimação de vectores de movimento.

Contudo, nem sempre é fácil obter este tipo de medidas a partir do esquema de processamento de uma determinada arquitectura. De facto, tanto a área de implementação (A) como o tempo de processamento (T) dependem fortemente da tecnologia utilizada para a implementação do circuito. Para além disso existem, ainda, em cada tecnologia, alguns limites físicos que impossibilitam a implementação de circuitos ocupando áreas de semicondutor superiores a um determinado valor limite.

Desta forma, torna-se muitas vezes necessária a utilização de estimativas tanto para a área (A) como para o tempo de processamento (T), que sejam independentes da tecnologia utilizada para a sua implementação. Assim, e dado que todas as arquitecturas descritas anteriormente são constituídas pelo mesmo tipo de elemento processador fundamental (o bloco AD), optou-se por considerar como estimativa da área de implementação o número

total de elementos processadores deste tipo requeridos por uma dada arquitectura:

$$A^* = N_{pe} \tag{5.1}$$

Da mesma forma, optou-se também por considerar como estimativa do tempo total de processamento o número total de ciclos de relógio (c), necessário à pesquisa do vector de movimento com maior medida de similaridade. Não se consideram, assim, os vários ciclos de relógio necessários para o enchimento do *buffer* de entrada e para o preenchimento inicial dos registos presentes nos vários elementos processadores:

$$T^* = c \tag{5.2}$$

Na tabela 5.1 apresentam-se os valores estimados das três medidas de desempenho consideradas, para cada uma das arquitecturas descritas no capítulo anterior.

Para relacionar as três medidas (A, T, AT) com as dimensões do macrobloco de referência (N) e da área de pesquisa ($2p+N$), definiu-se um novo parâmetro k, correspondente à razão entre o deslocamento máximo considerado em cada direcção (p), e a dimensão do macrobloco de referência (N):

$$k = \frac{p}{N} \tag{5.3}$$

Parametrizando as três medidas presentes na tabela 5.1 com esta nova medida k, procedeu-se à estimativa da área (A^*), tempo de processamento (T^*) e produto área-tempo de processamento (AT^*) quando k varia entre 0 e 2. Os resultados obtidos encontram-se representados nos gráficos das figuras 5.1, 5.2 e 5.3 para $N = 16$.

Estrutura	$N_{pe} \propto A^*$	$c \propto T^*$	$N_{pe} \cdot c \propto AT^*$
SinglePE	1	$N^2(2p+1)^2$	$N^2(2p+1)^2$
AB1	N	$N(2p+1)(2p+N)$	$N^2(2p+1)(2p+N)$
AS1	$2p+1$	$N(2p+N)(2p+1)$	$N(2p+N)(2p+1)^2$
Tipo 1 - AB2	$N \times N$	$(2p+1)(2p+N)$	$N^2(2p+1)(2p+N)$
Tipo 1 - Vos	$N \times N$	$(2p+1)^2$	$N^2(2p+1)^2$
Tipo 1 - Hsieh	$N \times N$	$(2p+N)^2$	$N^2(2p+N)^2$
AS2	$N \times (2p+1)$	$N(2p+1)$	$N^2(2p+1)^2$
Tipo 2	$(2p+1) \times (2p+1)$	N^2	$N^2(2p+1)^2$

Tabela 5.1: Comparação das arquitecturas descritas no capítulo anterior, com base nas medidas A, T e AT.

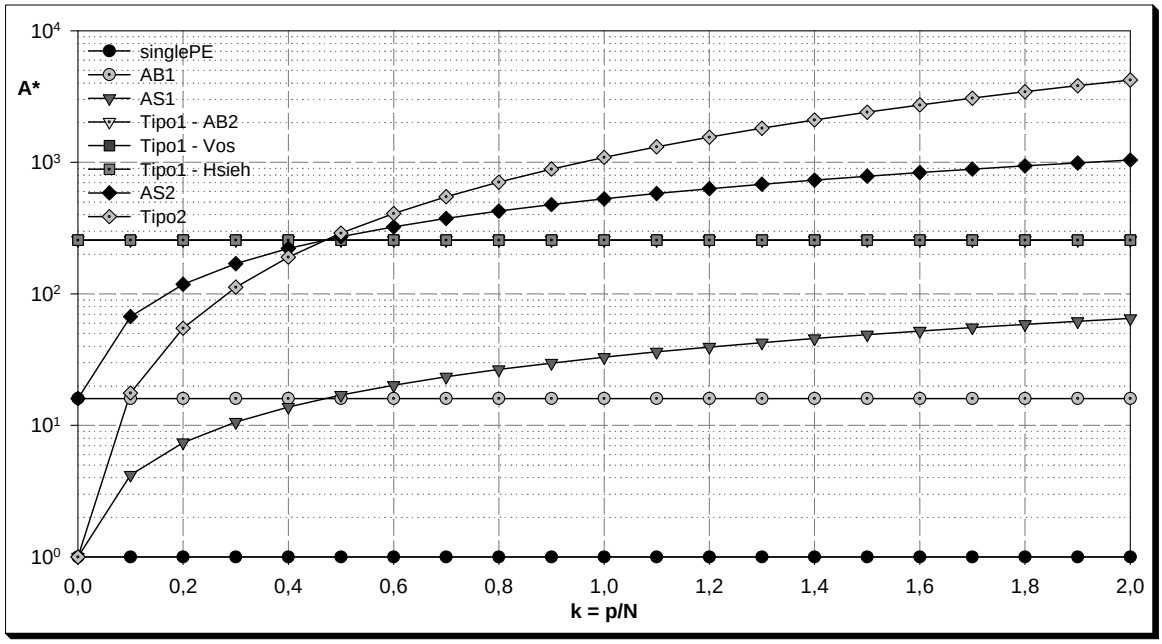


Figura 5.1: Variação da área de implementação A^* com o factor k.

Na figura 5.1 encontra-se representada graficamente a variação da área requerida (A^*) com k. Verifica-se que as arquitecturas unidimensionais exigem menos recursos de *hardware* do que as arquitecturas bidimensionais. Para além disso, pode ainda observar-se que enquanto as estruturas AB1 e Tipo 1 requerem uma área de implementação constante, independente da área de pesquisa considerada, as estruturas AS1, AS2 e Tipo 2 requerem uma área de implementação crescente com a dimensão da área de pesquisa. Estas últimas estruturas apresentam vantagens para áreas de pesquisa de dimensão reduzida, nomeadamente, para $p \leq N/2$. Para áreas de pesquisa maiores, verifica-se que a estrutura AB1 e as estruturas bidimensionais do Tipo 1 são vantajosas, requerendo N e N^2 elementos processadores respectivamente.

Na figura 5.2 representa-se graficamente a variação do tempo de processamento (T^*) com k. Neste caso, é visível a vantagem das estruturas bidimensionais sobre as estruturas unidimensionais. Para além disso, verifica-se que a maioria das arquitecturas descritas apresenta um tempo de processamento crescente com a área de pesquisa considerada. A excepção a este comportamento é verificada com a estrutura do Tipo 2 que, usando um elemento processador para calcular a medida de similaridade correspondente a cada um dos macroblocos candidatos considerados, apresenta um tempo total de processamento constante e independente da área de pesquisa considerada. As restantes arquitecturas bidimensionais apresentam um comportamento bastante semelhante entre si, sendo a arquitectura AS2 a mais adequada para aplicações com áreas de pesquisa elevadas.

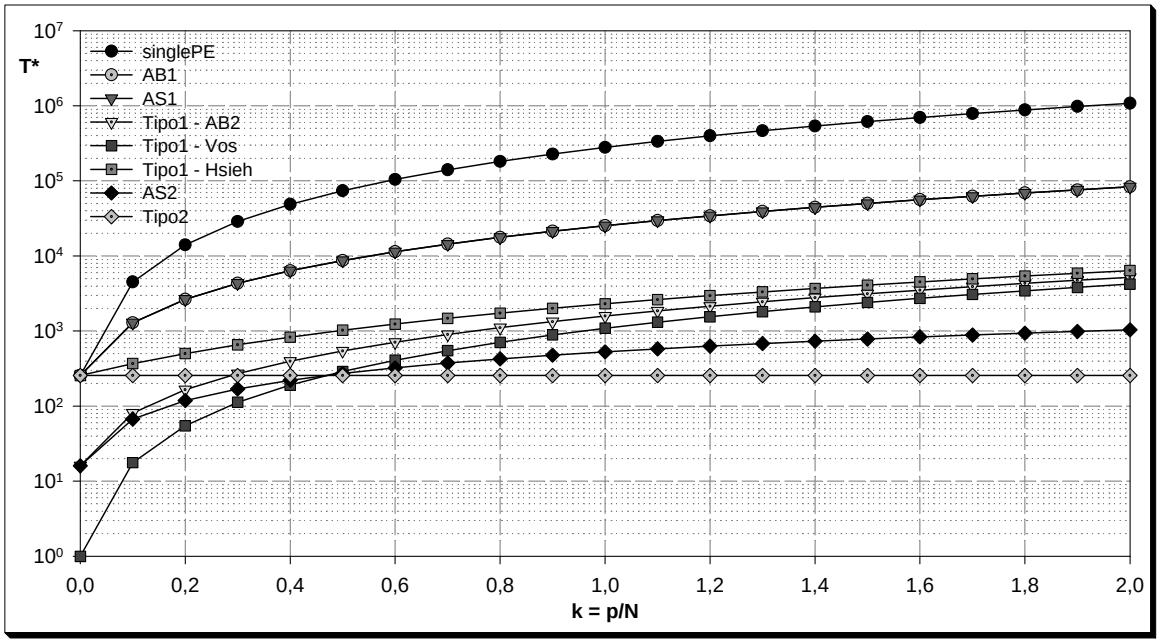


Figura 5.2: Variação do tempo de processamento T^* com o factor k.

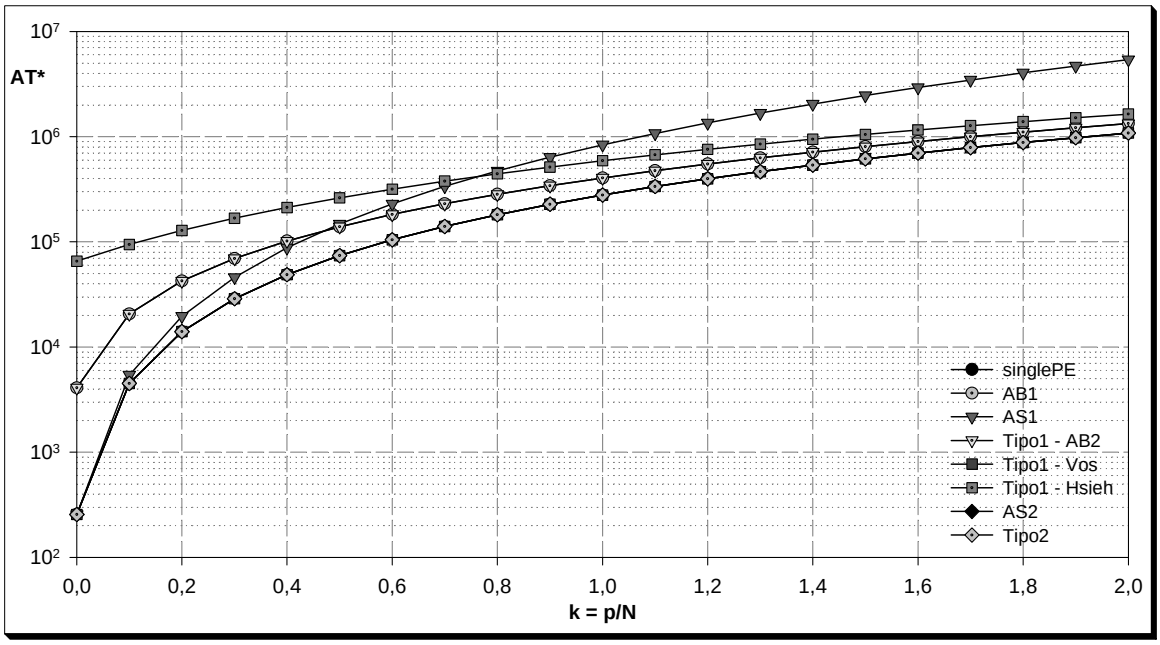


Figura 5.3: Variação do produto área de implementação - tempo de processamento AT^* com o factor k.

Finalmente, na figura 5.3 apresenta-se a variação do produto área de implementação - tempo de processamento (AT^*) quando a relação entre a dimensão da área de pesquisa (p) e a dimensão do macrobloco de referência (N) varia entre 0 e 2. Neste gráfico é

visível que, no conjunto das oito arquitecturas consideradas, existem apenas quatro tipos de comportamento distintos. Assim, e visto que um dos objectivos fundamentais é o de diminuir o produto área - tempo, verifica-se que as arquitecturas SinglePE, Tipo1-Vos, AS2 e Tipo2 apresentam o melhor desempenho, de acordo com esta medida. Verifica-se, também, que o conjunto constituído pelas arquitecturas AB1 e AB2 apresenta características um pouco piores, e que a arquitectura AS1 apresenta, para valores do parâmetro k maiores do que 1, o pior comportamento de entre todas as arquitecturas consideradas nesta análise.

Assim, e de acordo com os resultados apresentados, verifica-se que a arquitectura AB1 é, de entre todas as arquitecturas unidimensionais consideradas, aquela que oferece um melhor compromisso entre área de implementação requerida e tempo total de processamento.

No que se refere às estruturas bidimensionais, o estudo efectuado revela que a arquitectura do Tipo 1 proposta por *Vos* [9] apresenta o conjunto de características mais adequado para aplicações que requerem elevadas taxas de processamento, utilizando áreas de pesquisa de média e elevada dimensão. Demonstra, por isso, um desempenho adequado para processamento em tempo real em codificadores de vídeo de média e alta resolução. Para além disso, e conforme se pode observar nas figuras 5.1 e 5.2, esta arquitectura apresenta a vantagem de oferecer um baixo tempo de processamento, utilizando recursos de *hardware* que se podem considerar significativamente menores quando comparados com os requeridos por outras estruturas bidimensionais. É por estas razões que esta arquitectura é considerada como uma das mais eficientes no que se refere ao equilíbrio entre tempo de processamento (T) e área de semicondutor requerida (A).

5.2 Arquitectura apresentada por *Vos*

Face às considerações proferidas no final da secção anterior, optou-se por desenvolver uma arquitectura de um processador dedicado para estimação de movimento com um princípio de funcionamento baseado na estrutura do Tipo 1 proposta por *Vos* e Stegherr [9]. De facto, embora esta estrutura apresente um reduzido tempo de processamento e uma boa eficiência de utilização dos recursos de *hardware*, há um conjunto de características da arquitectura, ainda não exploradas, que podem ser utilizadas para melhorar a eficiência dos processadores, nomeadamente, no que diz respeito à utilização dos recursos de *hardware* e ao nível do paralelismo utilizado.

Na presente secção, composta por duas subsecções distintas, far-se-á uma discussão exaustiva do princípio de funcionamento da arquitectura que serviu de base ao trabalho

apresentado nesta dissertação, passando-se, depois, à proposta da nova arquitectura para estimação de movimento.

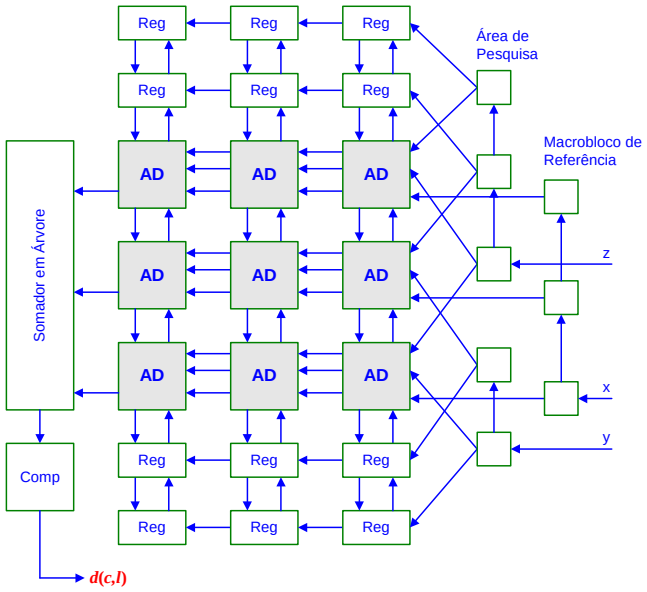
5.2.1 Estrutura

Conforme foi descrito na secção 4.4.1, o processador proposto por Vos e Stegherr [9] é composto por uma estrutura bidimensional do Tipo 1, onde cada pixel do macrobloco de referência é atribuído a um dos N^2 elementos processadores (ver figura 5.4(a)). Cada elemento processador é constituído por uma unidade aritmética e por vários registos, necessários para armazenar os valores correspondentes aos pixels do macrobloco actual e da área de pesquisa. Para além do agregado de elementos processadores, o processador é ainda composto por duas matrizes constituídas por $2p \times N$ registos, utilizadas para armazenar os dados correspondentes à área de pesquisa e que são interligadas com as fronteiras inferior e superior da matriz de elementos processadores (ver figura 5.4(a)). Os pixels correspondentes ao macrobloco actual são transferidos para o processador através do porto de entrada x, enquanto que os pixels da área de pesquisa são transferidos através das entradas y e z, cobrindo, assim, toda a área de pesquisa considerada.

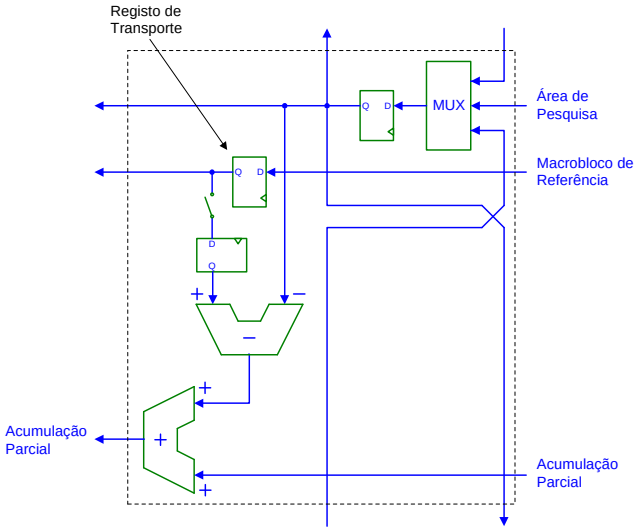
Dentro da matriz de elementos processadores, os dados correspondentes à área de pesquisa podem mover-se segundo três direcções distintas: para cima, para baixo e para a esquerda (ver figura 5.4). Assim, considere-se, a título de exemplo, que num determinado ciclo de relógio uma coluna constituída por $2p + N$ pixels da área de pesquisa é introduzida na estrutura através das $2p + N$ entradas superiores da cadeia de registos vertical. Simultaneamente, todos os pixels da área de pesquisa presentes na estrutura são deslocados uma posição para a esquerda. Nos $2p + 1$ ciclos de relógio seguintes, os dados da área de pesquisa são deslocados para baixo uma posição por ciclo de relógio. Desta forma, em cada ciclo de relógio os valores correspondentes aos pixels de diferentes macroblocos candidatos são, assim, introduzidos nos vários elementos processadores e considerados no cálculo da medida de similaridade correspondente. Decorridas $2p+1$ operações de deslocamento no sentido descendente processa-se, então, uma nova operação de deslocamento para a esquerda, sendo, uma vez mais, introduzida uma nova coluna de pixels nas entradas à direita do processador. Contudo, esta nova coluna de pixels é, agora, introduzida nas $2p + N$ posições da parte inferior da cadeia de registos vertical de entrada. Esta alternância entre as posições de entrada repete-se, assim, ao longo do processo de busca. Nos $2p+1$ ciclos de relógio seguintes os dados correspondentes à área de pesquisa são, agora, deslocados no sentido ascendente de uma forma análoga ao que foi descrito anteriormente, sendo, depois de $2p+1$ ciclos de relógio, deslocados uma vez mais uma posição para a esquerda. Desta forma, os deslocamentos necessários para efectuar o

processo de pesquisa podem resumir-se através do seguinte esquema de processamento:

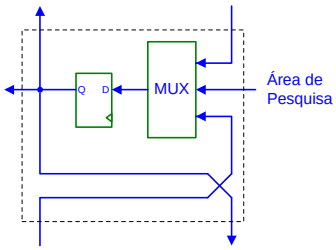
$$
\begin{gathered}
2p+1 \text{ ciclos para cima, 1 ciclo para a esquerda;} \\
2p+1 \text{ ciclos para baixo, 1 ciclo para a esquerda;} \\
\vdots \\
2p+1 \text{ ciclos para baixo, 1 ciclo para a esquerda.}
\end{gathered}
$$



(a) Matriz de elementos processadores interligada nas fronteiras superior e inferior a $2p \times N$ registos.



(b) Estrutura interna do elemento processador.



(c) Registo para deslocamento dos pixels da área de pesquisa.

Figura 5.4: Estrutura Tipo 1 proposta por *Vos* [9], considerando $N = 3$ e $p = 1$.

É precisamente este esquema de processamento que faz, desta arquitectura, uma das que permite o processamento mais rápido. De facto, ao contrário de todas as outras arquitecturas descritas no capítulo 4, o esquema de processamento em *zig-zag* desta estrutura evita ciclos de relógio extra entre duas linhas ou macroblocos candidatos consecutivos. Estes ciclos extra seriam necessários para efectuar o preenchimento ou a retirada de dados correspondentes a zonas da área de pesquisa desnecessárias para o cálculo da medida de similaridade do macrobloco candidato actual. Este facto é particularmente evidente se compararmos o esquema de processamento desta estrutura com o esquema de processamento da arquitectura proposta por Hsieh, descrita na secção 4.4.1.

A entrada dos pixels do macrobloco actual é feita por intermédio do registo de deslocamento vertical constituído por N posições. Decorridos N ciclos de relógio, é feita a introdução, na matriz de elementos processadores, de uma nova coluna de pixels do macrobloco actual, por intermédio do porto x situado na fronteira direita desta matriz. Simultaneamente, todos os dados do macrobloco actual presentes nesta matriz são deslocados uma posição para a esquerda, para o elemento processador vizinho. Decorridos N^2 ciclos de relógio existirá, assim, um novo macrobloco da imagem actual completo na matriz. Este novo macrobloco poderá, então, ser transferido dos registos de transporte para os registos de processamento, através de um interruptor existente em cada elemento processador, que liga o barramento de transporte de dados com o registo reservado a armazenar o pixel do macrobloco actual sob processamento (ver figura 5.4(b)).

Em cada elemento processador é calculado o valor absoluto da diferença entre dois pixels: um do macrobloco actual e outro do macrobloco candidato. Estes valores são, então, acumulados segundo linhas de elementos processadores, obtendo-se, assim, N valores parciais resultantes da acumulação parcial das diferenças. Estes valores parciais são depois somados para obter o valor final, através de um somador em árvore de N entradas localizado junto da fronteira esquerda da matriz de elementos processadores. O valor obtido é, então, transferido para um comparador, cuja função é seleccionar e reter o valor absoluto da diferença e as coordenadas do vector de movimento correspondente ao macrobloco candidato com maior medida de similaridade.

É de notar que a arquitectura até agora descrita pode ser facilmente sujeita a operações de sistolização, através da utilização de procedimentos de re-temporização e fazendo uso de registos de *pipeline* ao longo das várias partições temporais efectuados na estrutura [21], obtendo-se, assim, uma arquitectura completamente sistólica.

5.2.2 Elemento processador

Na figura 5.5 encontra-se representada a estrutura interna do elemento processador utilizado na matriz anteriormente descrita, tendo-se introduzido os registos de *pipeline* necessários para formar uma estrutura completamente sistólica. A introdução dos valores dos pixels correspondentes à área de pesquisa é feita por intermédio de um multiplexador, que selecciona o valor a processar de entre o conjunto de três valores disponíveis nos três elementos processadores vizinhos localizados em cima, em baixo ou à direita do elemento processador considerado. Em contraste, o valor do pixel do macrobloco actual é obtido a partir do registo de transporte presente no elemento processador, sempre que se dá início ao processamento de um novo macrobloco actual.

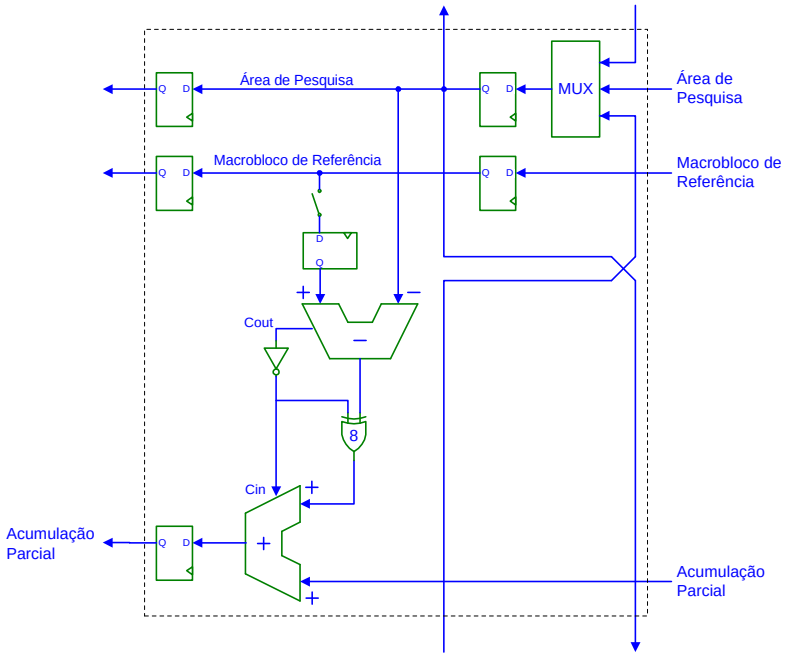


Figura 5.5: Diagrama de blocos do elemento processador utilizado na estrutura Tipo 1 proposta por *Vos* [9].

O valor absoluto da diferença é calculado por intermédio de um subtractor de 8 bits juntamente com um bloco composto por 8 portas XOR, que efectua a negação de todos os bits do resultado obtido quando, da operação de subtracção, resulta um número negativo. O resultado, assim obtido, é acumulado com o resultado parcial obtido no elemento processador vizinho localizado à direita do elemento processador considerado. Desta forma, obtém-se na extremidade esquerda da matriz um conjunto de N resultados parciais, que serão colocados à entrada do somador em árvore para calcular o valor absoluto da diferença do macrobloco candidato sob processamento.

5.3 Arquitectura proposta

Para facilitar a descrição da arquitectura proposta neste trabalho, optou-se por utilizar uma nomenclatura um pouco diferente da utilizada por *Vos*. Assim, se se compararem os dois tipos de blocos integrantes do processador de *Vos* (apresentados na figura 5.4), facilmente se constata que apenas os blocos anteriormente designados por *"Elementos Processadores"* (apresentados na figura 5.4(b)) efectuam, realmente, o processamento da medida de similaridade. Todos os outros blocos que integram as duas matrizes de registos funcionam, apenas, como elementos de memória, utilizados nas operações de deslocamento dos pixels correspondentes aos macroblocos candidatos da área de pesquisa. A partir de agora, designam-se estes dois tipos de elementos por elementos processadores *"activos"* e *"passivos"*, respectivamente. Designando cada uma das matrizes de elementos processadores por bloco, a estrutura ficará, assim, constituída por um *"bloco activo"* intercalado entre dois *"blocos passivos"* (ver figura 5.6). Por outro lado, adopta-se uma representação gráfica alternativa da arquitectura, correspondente a uma rotação de 90° do diagrama da estrutura apresentado na figura 5.6 (ver figura 5.7). Assim, a descrição anteriormente apresentada para o movimento dos pixels da área de pesquisa no sentido ascendente e descendente será, agora, efectuada através da indicação de movimento da esquerda para a direita e da direita para a esquerda.

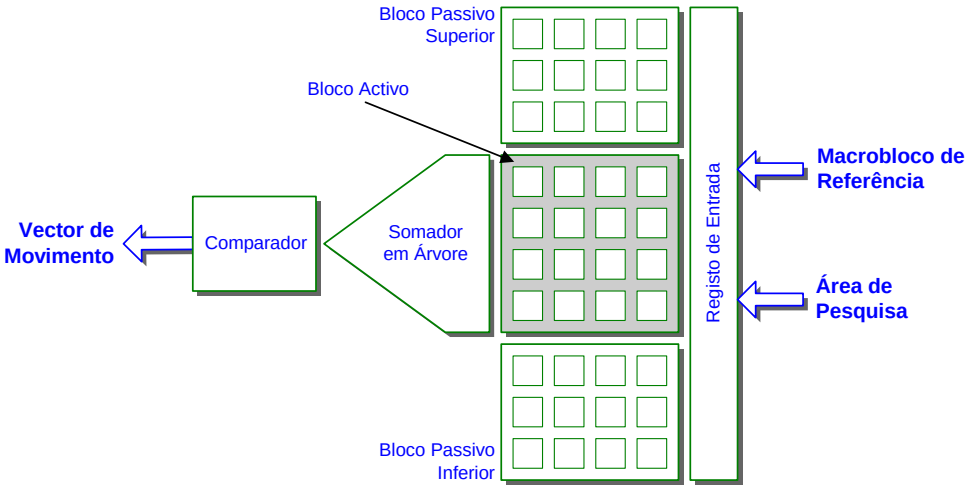


Figura 5.6: Estrutura do processador proposto por *Vos* [9], constituído por um *"bloco activo"* intercalado entre dois *"blocos passivos"*.

5.3.1 Estrutura

O esquema de processamento da arquitectura proposta por *Vos* e descrita na secção 5.2.1 pode ser representado por intermédio da sequência de estados apresentada na figura 5.8,

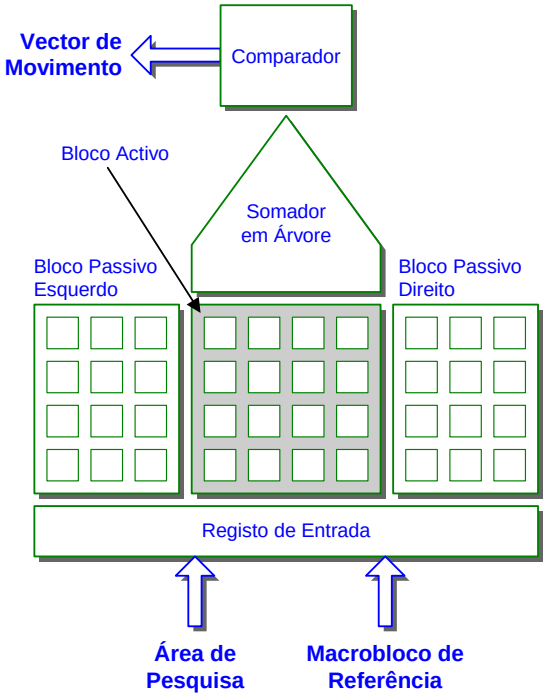


Figura 5.7: Representação alternativa da estrutura do processador proposto por *Vos* [9].

para $N = 4$ e $p = 2$. Convém notar que nesta figura optou-se por utilizar uma representação simplificada do processador, omitindo-se o registo de entrada, o somador em árvore e o comparador de saída, ilustrando-se, apenas, o esquema de processamento para a pesquisa. No caso considerado, a área de pesquisa é constituída por $(2p + N - 1)^2$ pixels, e o vector de movimento é seleccionado de entre um conjunto de $(2p)^2$ vectores de movimento candidatos (MV_c) com coordenadas definidas, em cada direcção, no intervalo $[-(p-1)\ ;\ p]$. Nesta figura representou-se a traço contínuo a fracção da área de pesquisa sob processamento e a traço interrompido as fracções da área de pesquisa já processadas ou em vias de ser processadas.

Na figura 5.8(a) encontra-se representada a situação correspondente ao processamento do macrobloco candidato com coordenadas $(-1, -1)$, correspondente à fracção da área de pesquisa representada a traço contínuo e que se encontra sob processamento no bloco activo do processador (representado a cinzento), no instante $t = T_0$. A traço interrompido encontra-se representada a fracção da área de pesquisa seguinte, que começará a ser processada no ciclo de relógio seguinte.

No instante $t = T_0 + 1$ verifica-se a saída da fracção da área de pesquisa utilizada nos ciclos de relógio anteriores e a entrada, no processador, de uma nova fracção da área de pesquisa, dando-se, assim, início à pesquisa na linha de macroblocos candidatos com coordenada horizontal $l = 0$. Neste ciclo de relógio e durante os três ciclos de relógio

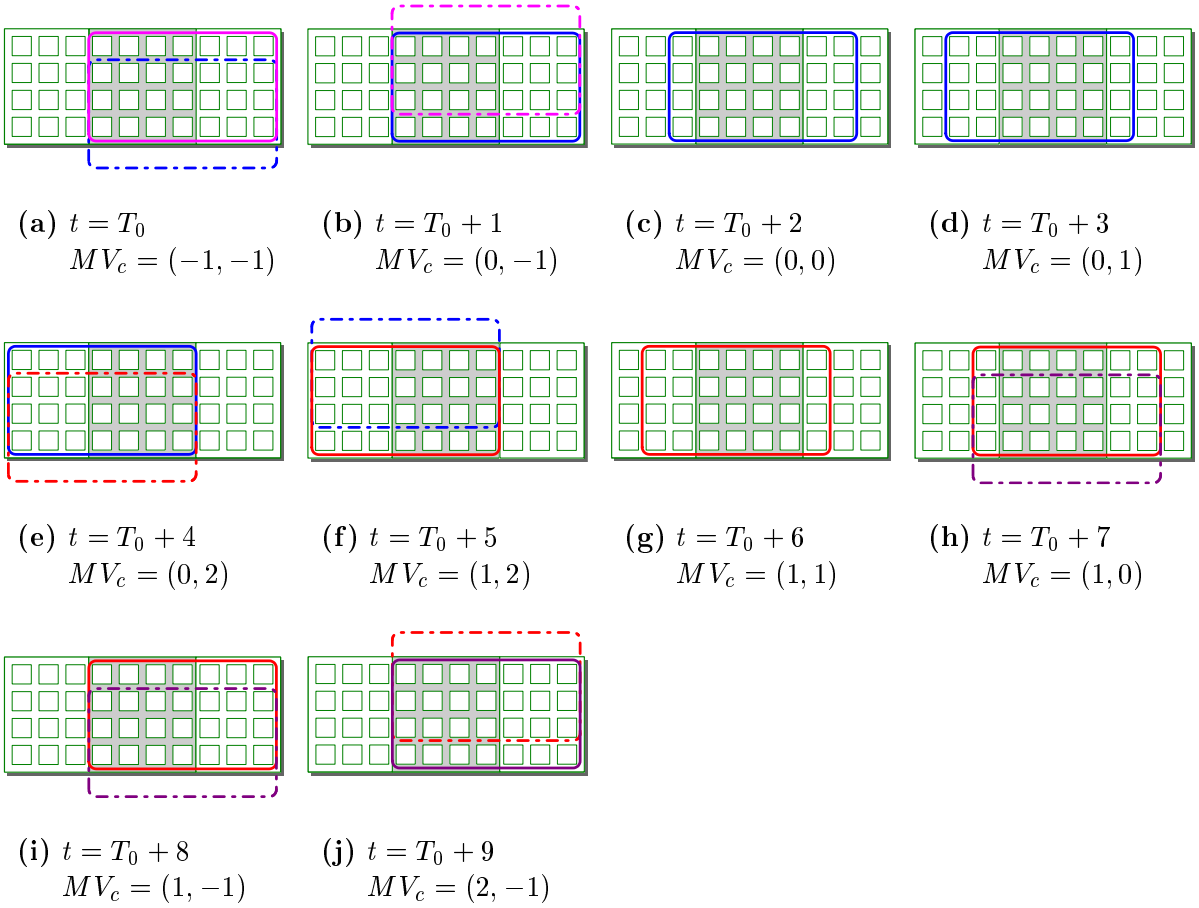


Figura 5.8: Esquema de processamento em *"zig-zag"* proposto por *Vos*, considerando $N = 4$ e $p = 2$, obtendo-se um sistema de coordenadas definidas no intervalo $[-1, 2]$.

seguintes procede-se, então, ao cálculo das medidas de similaridade correspondentes às coordenadas $(0, -1)$, $(0, 0)$, $(0, 1)$ e $(0, 2)$, fazendo passar cada um dos macroblocos candidatos sobre o bloco activo do processador. Durante este tempo, o deslocamento dos pixels é feito no sentido da direita para a esquerda.

No instante $t = T_0 + 5$ regista-se a saída da zona da área de pesquisa anteriormente utilizada no processo de procura e a entrada de uma nova região de pesquisa, correspondente aos macroblocos candidatos com coordenada horizontal $l = 1$. Procede-se, ao mesmo tempo, ao cálculo da medida de similaridade correspondente à coordenada $(1, 2)$. O processamento desta linha termina no ciclo de relógio $t = T_0 + 8$, tendo-se, então, considerado os macroblocos candidatos com coordenadas $(1, 2)$, $(1, 1)$, $(1, 0)$ e $(1, -1)$. Durante este período o deslocamento dos pixels é feito no sentido da esquerda para a direita.

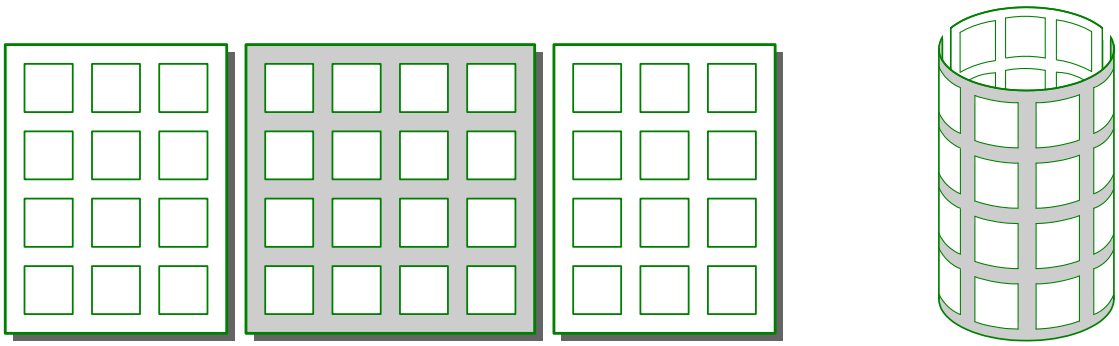
Na figura 5.8(j) representou-se a situação correspondente à entrada de uma nova

fracção da área de pesquisa com coordenada horizontal $l = 2$. Esta situação, correspondente ao cálculo da medida de similaridade do macrobloco candidato com coordenada $(2, -1)$, é em tudo idêntica à representada na figura 5.8(b), pelo que se completa o ciclo de descrição do funcionamento do processador.

É de destacar que, ao longo do processamento de toda a área de pesquisa não há, em qualquer circunstância, a necessidade de utilizar ciclos de relógio adicionais para retirar ou inserir dados nos blocos de processamento constituintes do processador. Esta característica deve-se à utilização do esquema de processamento em *"zig-zag"* anteriormente descrito e à utilização de registos de transporte no interior dos elementos processadores (ver figura 5.4).

Contudo, e tal como se pode concluir a partir da descrição do esquema de processamento anterior e da figura 5.8, o processador descrito não utiliza todos os recursos de *hardware* em cada ciclo de relógio. De facto, verifica-se que, em qualquer um dos estados do processamento, existe sempre uma quantidade de elementos processadores passivos sem qualquer informação útil. Esta situação acontece sempre que a zona da área de pesquisa sob processamento se desloca para fora da zona coberta pelo bloco activo da arquitectura. Assim, considerando que o processador é composto por N^2 elementos processadores activos e por $2 \times N(2p-1)$ elementos processadores passivos, e que as faixas da área de pesquisa são constituídas por $N \times (N + 2p - 1)$ pixels, verifica-se que existem sempre $N \times (2p-1)$ elementos processadores passivos que não se encontram a efectuar trabalho útil. Contudo, estes elementos processadores são necessários sempre que se deslocam os pixels da área de pesquisa para a zona por eles coberta.

Uma forma de resolver este problema consiste em começar por constatar que o número de elementos processadores sem informação útil é, em qualquer instante, constante e igual ao número de elementos processadores passivos que integram cada um dos blocos passivos presentes em cada um dos lados do bloco activo. Assim, coloca-se, então, a questão da necessidade de utilizar estes dois blocos, visto que, em qualquer instante, apenas o equivalente a um deles se encontra a realizar trabalho útil. A resposta a esta questão pode ser facilmente encontrada se dispusermos o plano do processador segundo a superfície de um cilindro, tal como se ilustra na figura 5.9. Note-se que, neste caso, os dois blocos passivos encontram-se sobrepostos, pelo que há apenas a necessidade de utilizar um único bloco passivo para efectuar o deslocamento dos pixels da área de pesquisa. De notar ainda que este deslocamento é, tal como anteriormente, efectuado segundo o esquema de processamento em *"zig-zag"* em torno da superfície cilíndrica. Contudo, verifica-se agora com esta estrutura que todos os elementos processadores se encontram, em qualquer instante, a efectuar trabalho útil.



(a) (b)

Figura 5.9: Alteração da estrutura do processador para melhorar a eficiência de utilização dos recursos de *hardware*: **(a)** - Plano do processador constituído por um bloco activo intercalado entre dois blocos passivos; **(b)** - Disposição do plano do processador sobre a superfície de um cilindro, com a sobreposição dos dois blocos passivos.

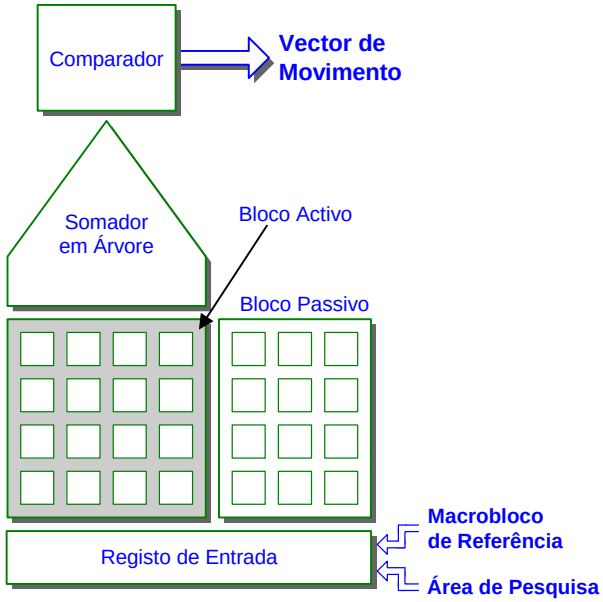


Figura 5.10: Diagrama de blocos do processador agora proposto, optimizado em termos da utilização dos recursos de *hardware*.

A estrutura do processador proposto assume a forma apresentada na figura 5.10. Para simplificar a representação, optou-se, uma vez mais, por se considerar uma estrutura planar e constituída por um bloco activo e por um único bloco passivo. Porém, assume-se que os elementos processadores localizados na fronteira direita do bloco passivo encontram-se interligados com os elementos processadores da fronteira esquerda do bloco activo, tal como se ilustrou na figura 5.9(b). Comparando a figura 5.10 com a figura 5.7 é clara a

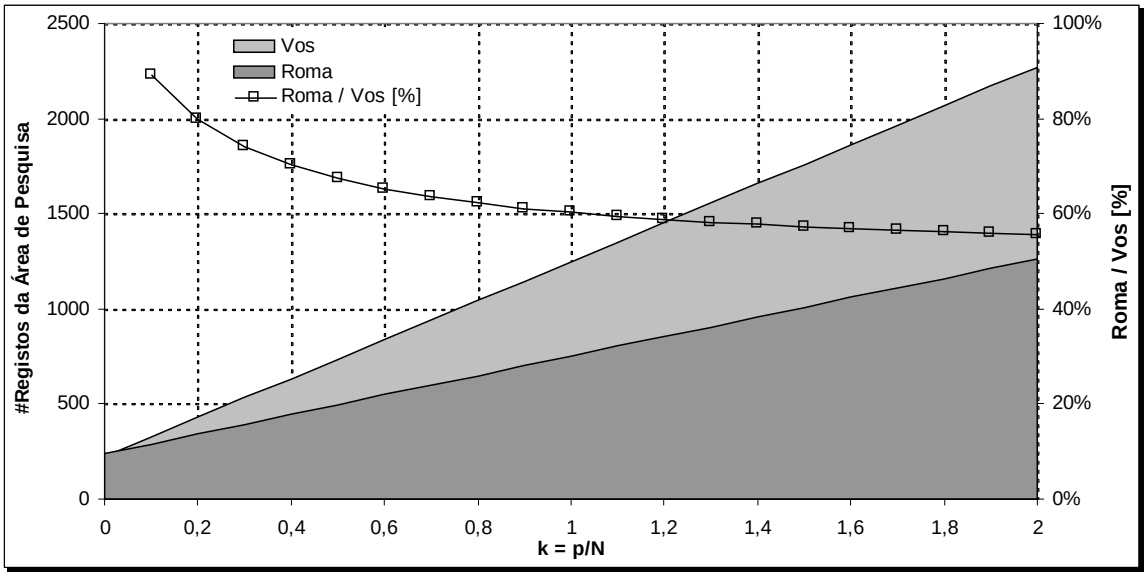


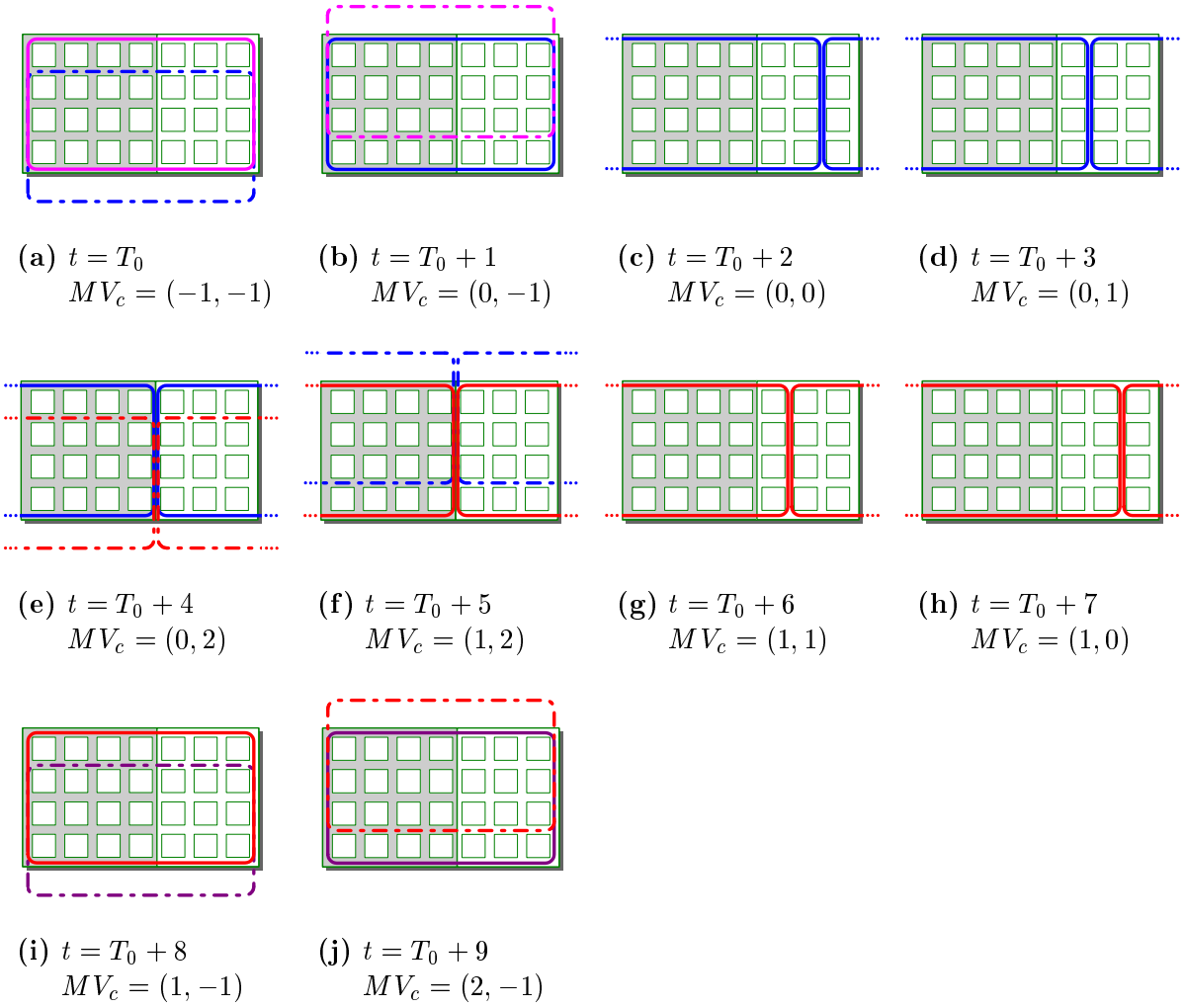
Figura 5.11: Variação do número de registos utilizados para efectuar o deslocamento dos pixels da área de pesquisa com k ($k = p/N$).

redução conseguida no que diz respeito ao número de elementos processadores passivos e ao comprimento e complexidade do registo de entrada.

Na figura 5.11 encontra-se representado o gráfico com a variação do número de registos utilizados na arquitectura proposta por *Vos* [9] e na arquitectura proposta nesta dissertação, que utiliza, apenas, um bloco passivo para efectuar o deslocamento dos pixels da área de pesquisa. Tal como se ilustrou nas figuras 5.4(b) e 5.4(c), tanto o elemento processador activo como o elemento processador passivo requerem, apenas, um único registo de 8 bits para propagarem os pixels da área de pesquisa. Assim, verifica-se que o processador proposto por *Vos* requer um total de $[N + 2 \times (2p - 1)] \times N$ registos, enquanto que a arquitectura agora proposta requer, apenas, $[N + (2p - 1)] \times N$ registos. No gráfico apresentado considerou-se, como dimensão do macrobloco de referência, o valor $N = 16$, tendo-se feito variar a relação entre a dimensão do macrobloco de referência e o deslocamento máximo na área de pesquisa ($k = p/N$) entre 0[1] e 2. A curva representada no gráfico com o símbolo "□" ilustra a relação entre o número de registos requerido pelas duas arquitecturas. Esta relação atinge valores bastante significativos, sendo cerca de 60% para $k = 1$ ($p = N$) e de 55% para $k = 2$ ($p = 2N$).

Na figura 5.12 apresenta-se a descrição do esquema de processamento da arquitectura agora proposta. Tal como na figura 5.8, parametrizou-se o processador com a dimensão do macrobloco de referência $N = 4$ e a área de pesquisa para um deslocamento máximo, em cada direcção, de $p = 2$, sendo constituída por $(2p + N - 1)^2$ pixels. Obtém-se,

[1] Correspondente a considerar-se, apenas, um único macrobloco candidato com coordenadas $(0, 0)$.



(a) $t = T_0$ $MV_c = (-1, -1)$ **(b)** $t = T_0 + 1$ $MV_c = (0, -1)$ **(c)** $t = T_0 + 2$ $MV_c = (0, 0)$ **(d)** $t = T_0 + 3$ $MV_c = (0, 1)$

(e) $t = T_0 + 4$ $MV_c = (0, 2)$ **(f)** $t = T_0 + 5$ $MV_c = (1, 2)$ **(g)** $t = T_0 + 6$ $MV_c = (1, 1)$ **(h)** $t = T_0 + 7$ $MV_c = (1, 0)$

(i) $t = T_0 + 8$ $MV_c = (1, -1)$ **(j)** $t = T_0 + 9$ $MV_c = (2, -1)$

Figura 5.12: Esquema de processamento em *"zig-zag"* utilizado no processador proposto nesta dissertação, considerando $N = 4$ e $p = 2$, obtendo-se um sistema de coordenadas definidas no intervalo $[-1, 2]$.

assim, um conjunto constituído por $(2p)^2$ macroblocos candidatos, distribuídos segundo um sistema de coordenadas definido, em cada direcção, no intervalo $[-(p-1)\ ;\ p]$.

Na figura 5.12(a) encontra-se representada a situação correspondente ao processamento do macrobloco candidato com coordenadas $(-1, -1)$, pertencente à fracção da área de pesquisa representada a traço contínuo e que se encontra sob processamento no bloco activo do processador. A traço interrompido representou-se a fracção da área de pesquisa que começará a ser processada no ciclo de relógio seguinte.

No instante $t = T_0 + 1$ verifica-se a saída da região da área de pesquisa utilizada nos ciclos de relógio anteriores e a entrada, no processador, de uma nova região da área de pesquisa, dando-se, assim, início ao cálculo da medida de similaridade correspondente ao macrobloco candidato com coordenadas $(0, -1)$. Durante este ciclo e nos três ciclos de

relógio seguintes é feito o processamento de todos os macroblocos candidatos correspondentes à coordenada horizontal $l = 0$: $(0, -1)$, $(0, 0)$, $(0, 1)$ e $(0, 2)$. Este procedimento é realizado fazendo-se passar cada um dos macroblocos candidatos sobre o bloco activo do processador, efectuando-se um deslocamento dos pixels da área de pesquisa segundo um movimento de rotação no sentido dos ponteiros do relógio.

No instante $t = T_0 + 5$ verifica-se a saída da região da área de pesquisa anteriormente processada e a entrada de uma nova fracção, correspondente ao conjunto de macroblocos candidatos com coordenada horizontal $l = 1$. Em simultâneo, efectua-se, também, o cálculo da medida de similaridade correspondente ao macrobloco candidato com coordenada $(1, 2)$. O processamento desta linha termina no ciclo de relógio $t = T_0 + 8$, tendo-se, então, considerado o conjunto de macroblocos candidatos correspondentes às coordenadas $(1, 2)$, $(1, 1)$, $(1, 0)$ e $(1, -1)$. Durante este período, o deslocamento dos pixels da área de pesquisa é feito, agora, segundo um movimento de rotação no sentido contrário ao dos ponteiros do relógio.

Por fim, no instante $t = T_0 + 9$, processa-se a entrada de uma nova fracção da área de pesquisa no processador, tendo agora como coordenada horizontal o valor $l = 2$. Em simultâneo, efectua-se, também, o processamento do macrobloco com coordenadas $(2, -1)$. Esta situação é em tudo idêntica à correspondente ao instante $t = T_0 + 1$, pelo que se completa o ciclo de descrição do esquema de funcionamento do processador.

5.3.2 Transferência de dados

Para minimizar o número de ciclos de relógio necessários para realizar o algoritmo de estimação de movimento, desenvolveu-se um sistema de transporte e transferência dos pixels da área de pesquisa e do macrobloco actual para o processador de *"dupla camada"*, implementado através dos vários elementos processadores que constituem a matriz de processamento. Este sistema permite a transferência e o transporte, em simultâneo, de dados correspondentes a dois macroblocos de referência e a duas áreas de pesquisa distintos. Desta forma, torna-se possível efectuar o pré-carregamento, em modo transparente, dos vários registos do processador com os dados correspondentes ao próximo macrobloco de referência e à próxima área de pesquisa. Este procedimento é realizado em simultâneo com a execução do processamento correspondente ao macrobloco de referência actual (ver figura 5.13). No final do processamento do macrobloco de referência actual efectua-se, então, a transferência imediata dos dados presentes nos registos do nível de transporte para os registos do nível principal do processador.

Esta estratégia de transporte de dados permite, assim, eliminar a necessidade de utilizar ciclos de relógio adicionais para retirar e introduzir informação na matriz de pro-

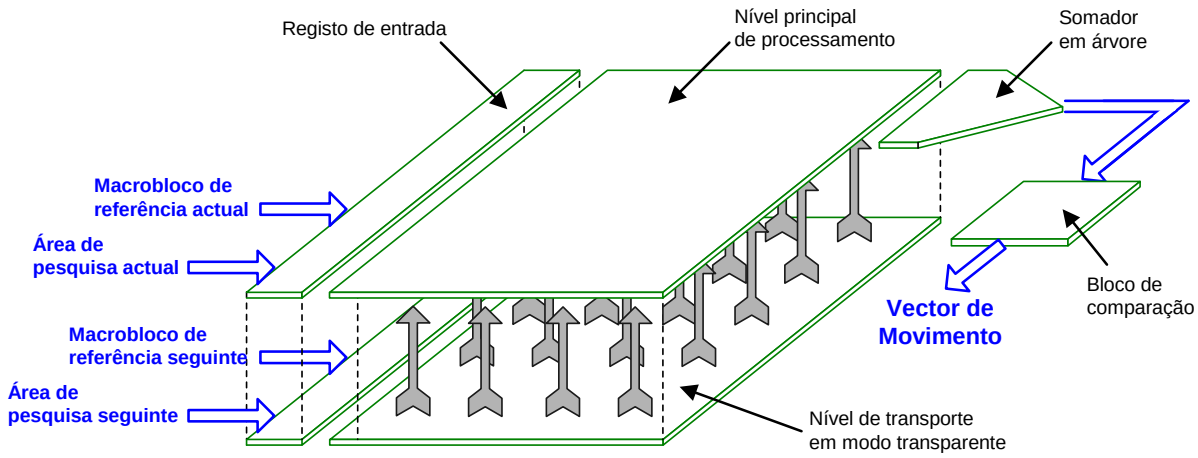


Figura 5.13: Sistema de processamento de duas camadas, permitindo o pré-carregamento dos registos internos dos elementos processadores em modo transparente.

cessamento entre macroblocos de referência consecutivos. Este facto contrasta significativamente com o modo de funcionamento das arquitecturas propostas por vários autores e descritas no capítulo 4, onde uma fracção significativa do tempo de processamento é reservada a efectuar este tipo de tarefa.

Com este sistema é possível, assim, obter um nível de eficiência do processador muito próximo dos 100%, pois em cada ciclo de relógio obtém-se, à entrada do bloco de comparação, um novo valor da medida de similaridade correspondente ao macrobloco candidato sob processamento.

Contudo, convém notar que este nível óptimo de eficiência é apenas conseguido à custa de um aumento da quantidade de *hardware* utilizada, nomeadamente, no que se refere aos circuitos correspondentes aos registos de transporte do macrobloco de referência e da área de pesquisa em modo transparente. De facto, a partir do que se descreveu, o processador desenvolvido requer um total de $2 \times [[N + (2p - 1)] \times N]$ registos, enquanto que o processador proposto por *Vos* requer $[N + 2 \times (2p - 1)] \times N$ registos. Comparando as duas expressões, constata-se que a diferença do número de registos é de N^2. Contudo, considerando, por exemplo, uma implementação com $N = 16$ e $k = 1$ ($p = N = 16$), verifica-se que esta diferença corresponde a um aumento de cerca de $20,5\%$, valor que, em face dos ganhos obtidos em termos de desempenho, se pode considerar aceitável.

Para além do acréscimo dos recursos de *hardware* verificado, é conveniente notar, ainda, que este sistema de transporte apenas cumpre o objectivo proposto se o sistema exterior de codificação de vídeo e, em particular, o banco de memória de vídeo a ele associado, permitir dispor de uma largura de banda suficiente para fornecer, em simultâneo, a quantidade de dados requerida pelo processador.

5.4 Arquitecturas multi-processador propostas

Considerando a arquitectura proposta anteriormente, verifica-se que, para além das melhorias introduzidas ao nível da utilização dos recursos de *hardware* é ainda possível alterar o esquema de processamento inicialmente proposto para aumentar o desempenho dos processadores. As alterações agora propostas incidem no aumento do nível de paralelismo explorado, através da utilização de múltiplos blocos activos na estrutura do processador. Desta forma, é possível realizar o cálculo de diversas medidas de similaridade em simultâneo, correspondentes a diferentes macroblocos candidatos.

O processador para estimação de movimento proposto na secção 5.3 desloca um bloco activo contendo os pixels do macrobloco de referência sobre a área de pesquisa, calculando-se o valor absoluto da diferença entre cada um dos $(2p)^2$ macroblocos candidatos e o macrobloco de referência actual (ver figura 5.14). De acordo com a descrição anterior do funcionamento do processador, o processamento é realizado fazendo passar, gradualmente, os pixels da área de pesquisa pelos blocos activo e passivo que constituem o processador que são, no total, constituídos por $[N + (2p - 1)] \times N$ elementos processadores (ver figura 5.15).

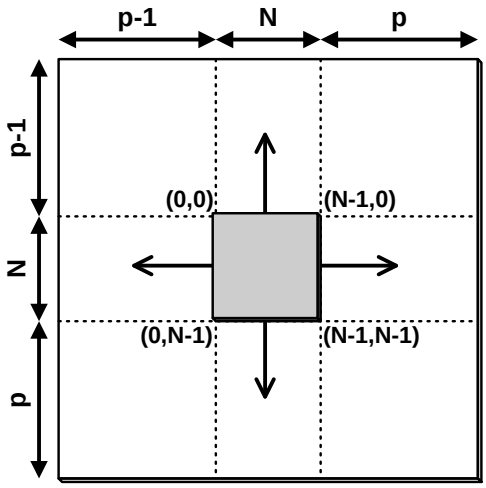


Figura 5.14: Princípio de funcionamento do processador proposto.

Uma vez que não existe qualquer tipo de dependência de dados no cálculo das medidas de similaridade dos vários macroblocos candidatos, torna-se possível acelerar o processamento através do cálculo, em simultâneo, de várias medidas de similaridade, correspondentes a diferentes macroblocos candidatos. Cada elemento processador activo desempenha, do ponto de vista da transferência dos pixels da área de pesquisa, uma função em tudo idêntica à desempenhada pelos elementos processadores passivos. Assim, pode-se substituir um ou mais agregados de $N \times N$ elementos processadores passivos por

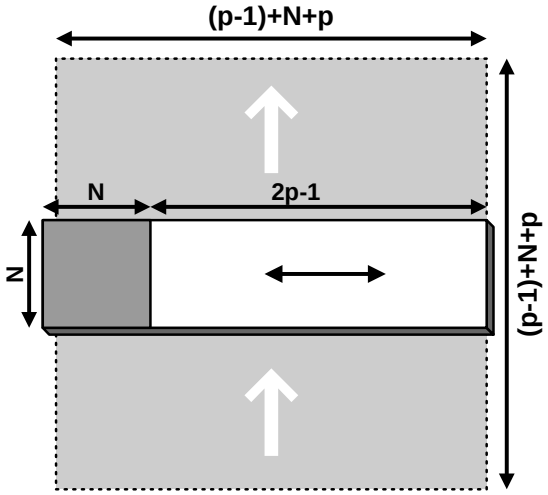


Figura 5.15: Procedimento de pesquisa do macrobloco candidato óptimo, fazendo passar os pixels da área de pesquisa pelos blocos activo e passivo.

elementos processadores activos, conforme se ilustra na figura 5.16 para o caso de uma implementação constituída por dois blocos activos. Uma consequência imediata desta alteração será a necessidade de introduzir, também, múltiplos somadores em árvore, para calcular as várias medidas de similaridade produzidas em cada instante.

O bloco passivo inicial constituído por $(2p-1)\times N$ elementos processadores será, agora, decomposto em dois blocos passivos de igual dimensão, localizados junto dos respectivos blocos activos, e ainda num bloco passivo adicional designado de *"bloco passivo de ligação"*, que desempenhará, igualmente, funções de deslocamento de pixels, conforme se explica mais à frente no texto.

Para além da introdução de múltiplos blocos activos no processador, verifica-se, ainda, a possibilidade de efectuar o processamento da medida de similaridade de cada macrobloco candidato de uma forma faseada, isto é, o cálculo da diferença não tem de ser efectuado de uma só vez por cada bloco activo, podendo ser feito *at posteriore* e repartido por vários ciclos de relógio. Esta característica permitirá, por exemplo, diminuir a dimensão dos blocos activos existentes no processador, tornando assim possível adaptar a arquitectura do processador às características da tecnologia utilizada na sua implementação. Esta solução apresenta custos adicionais no que se refere ao nível de complexidade dos restantes blocos que constituem o processador, nomeadamente, dos somadores em árvore e do bloco de comparação. É também possível diminuir a dimensão do sistema constituído pelos blocos activos e passivos para apenas $[(2p-1)+N]\times h$ elementos processadores, $(0 < h \leq N)$. Da mesma forma, também a largura de cada bloco activo poderá ser adaptada às características pretendidas, pelo que esta largura (ℓ) poderá variar, também,

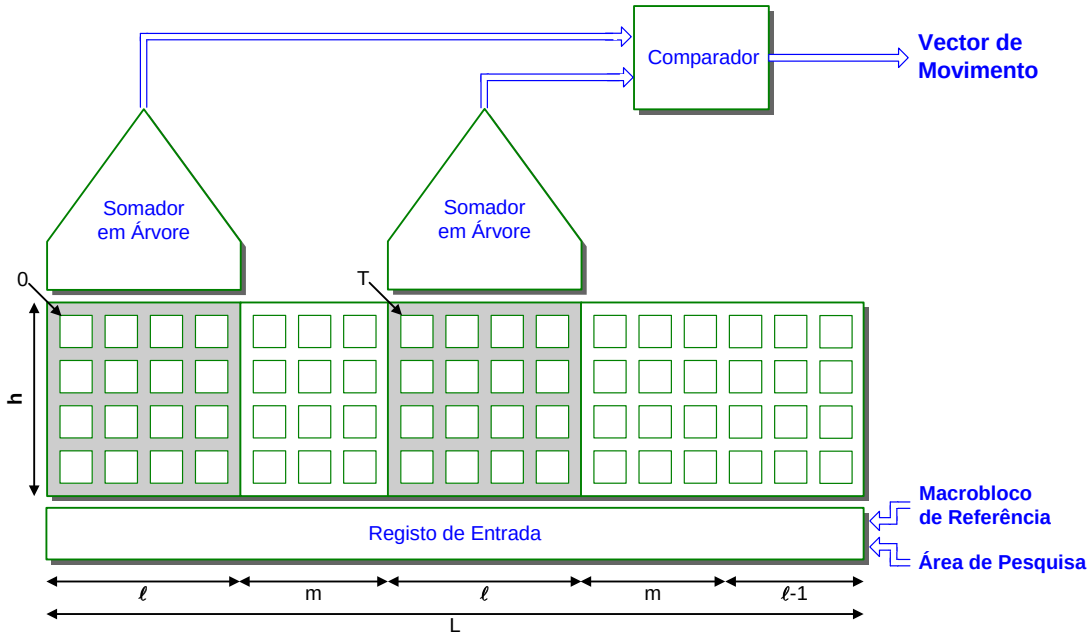


Figura 5.16: Estrutura de um processador para estimação de movimento com dois blocos activos ($NC = 2$).

entre 0 e N. No seguimento da descrição, designar-se-á o número total de blocos activos que integram o processador pela grandeza NC[2].

5.4.1 Estrutura geral

De acordo com a figura 5.16, verifica-se que o sistema, composto pelos blocos activos e passivos, dispõe de $h \times L$ elementos processadores, em que:

$$L = 2p + N - 1. \tag{5.4}$$

O número de macroblocos candidatos (MV_c) em cada linha da área de pesquisa, constituída por L pixels, será, então, dado por:

$$\#MV_c/linha = L - N + 1 = 2p. \tag{5.5}$$

Dividindo este valor pelo número de blocos activos do processador obtém-se, para o valor de T (ver figura 5.16):

$$T = \frac{\#MV_c/linha}{NC} = \left\lfloor \frac{2p}{NC} \right\rfloor \tag{5.6}$$

[2]Do Inglês *Number of Cores*.

Como o resultado anterior terá de ser, na prática, um valor inteiro, torna-se evidente a possibilidade de sobrarem vectores correspondentes aos últimos $2p \bmod (NC)$ macroblocos candidatos, localizados à direita na área de pesquisa.

Como consequência, verifica-se que para distribuir de uma forma equilibrada o número total de macroblocos candidatos pelos NC blocos activos, torna-se necessário considerar, apenas, os primeiros V macroblocos candidatos localizados na parte esquerda da área de pesquisa, a que correspondem os vectores de movimento aqui designados por *"vectores úteis"*:

$$V = \#MV_c\_úteis = NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor = NC \cdot T \tag{5.7}$$

A área de pesquisa será, assim, constituída por um total de $V \times V$ macroblocos candidatos, e o número total de macroblocos processados por cada bloco activo é dado pelo valor de T. O valor obtido a partir da equação 5.7 é igual ao apresentado na equação 5.6. Logo, o número total de elementos processadores passivos entre cada bloco activo deverá ser dado por:

$$m = T - \ell = \left\lfloor \frac{2p}{NC} \right\rfloor - \ell \tag{5.8}$$

e o número total de elementos processadores activos e passivos que constituem o sistema é dado por $h \times L'$, em que:

$$L' = V + (\ell - 1) = NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + (\ell - 1), \tag{5.9}$$

designando-se à fracção constituída pelo agregado composto por $(\ell - 1) \times h$ elementos processadores passivos localizados na extremidade direita do processador por *"bloco passivo de ligação"*. Convém recordar que a coluna de elementos processadores localizadas à direita deste bloco encontra-se interligada com a coluna de elementos processadores da face esquerda do primeiro bloco activo, formando, assim, a estrutura cilíndrica anteriormente descrita (ver figura 5.9).

Na figura 5.17 apresenta-se o exemplo de um processador de estimação de movimento considerando $N = 4$, $p = 10$ e utilizando 3 blocos activos ($NC = 3$):

EXEMPLO:

Neste caso, e partindo do princípio de que se pretende obter o valor da medida de similaridade de cada macrobloco candidato num único ciclo de relógio ($h = \ell = N$), o sistema deverá ser constituído por $h \times L'$ elementos processadores, sendo $h = N = 4$ e:

$$L' = NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + (\ell - 1) = 3 \cdot \left\lfloor \frac{20}{3} \right\rfloor + (4 - 1) = 21. \tag{5.10}$$

O número de macroblocos candidatos em cada linha da área de pesquisa correspondentes a vectores de movimento úteis (V) será dado por:

$$V = 3 \cdot \left\lfloor \frac{20}{3} \right\rfloor = 18, \tag{5.11}$$

pelo que cada bloco activo irá processar, em cada linha, T macroblocos candidatos:

$$T = \left\lfloor \frac{20}{3} \right\rfloor = 6. \tag{5.12}$$

De acordo com a equação 5.8, o número de elementos processadores passivos entre cada bloco activo será de:

$$m = T - \ell = 6 - 4 = 2. \tag{5.13}$$

De acordo com os cálculos apresentados, o processador pretendido tomará a forma apresentada na figura 5.17.

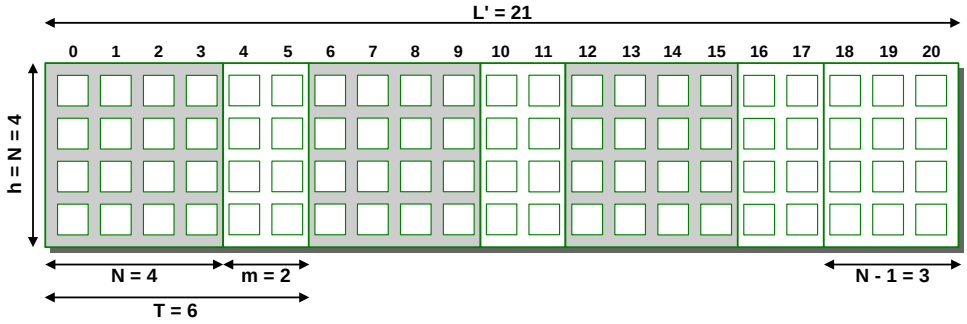


Figura 5.17: Exemplo da estrutura de um processador para estimação de movimento com arquitectura multi-processador, considerando a utilização de três blocos activos ($NC = 3$), $N = 4$ e $p = 10$.

Para garantir o correcto funcionamento das arquitecturas multi-processador há que respeitar algumas condições. De facto, seria possível imaginar um caso limite onde todos os elementos processadores passivos seriam substituídos por elementos processadores activos[3]. No entanto, apenas se garante o correcto funcionamento do esquema proposto no caso de não haver sobreposição dos blocos activos, pois caso contrário tornar-se-ia necessário haver, também, sobreposição de blocos somadores em árvore, para além de se aumentar significativamente a complexidade da estrutura interna de cada elemento processador.

[3]Partindo do princípio de que a tecnologia utilizada para implementar o processador garante a possibilidade de alocar todos os requisitos de *hardware* requeridos.

Assim, apenas se garante a não existência de sobreposição entre blocos activos *sse*:

$$\frac{L' - (\ell - 1)}{NC} > \ell \tag{5.14a}$$

$$\Leftrightarrow \frac{NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + (\ell - 1) - (\ell - 1)}{NC} > \ell \tag{5.14b}$$

$$\Leftrightarrow \left\lfloor \frac{2p}{NC} \right\rfloor > \ell. \tag{5.14c}$$

Atendendo a que $x \geq \lfloor x \rfloor$ vem que:

$$\frac{2p}{NC} \geq \left\lfloor \frac{2p}{NC} \right\rfloor > \ell \quad \Rightarrow \quad p \geq \frac{NC \cdot \ell}{2}. \tag{5.15}$$

Esta relação permite, assim, estabelecer a condição necessária e suficiente para que, usando NC blocos activos, cada um processando um conjuntos de $(h \times \ell)$ pixels do macrobloco de referência, e considerando uma área de pesquisa parametrizada por p, não exista sobreposição dos blocos activos que constituem o sistema.

A arquitectura multi-processador proposta pode dar origem a diferentes estruturas, designadas por topologias, com desempenhos e recursos de *hardware* distintos em função do nível de paralelismo pretendido. Nas próximas secções proceder-se-á ao estudo de três topologias reversíveis de estimação de movimento, comparando-se as suas características em termos do desempenho e da quantidade de *hardware* requeridos, utilizando-se, como termo de comparação, uma arquitectura semelhante mas não reversível.

5.4.2 Topologia do tipo A

Esta topologia, apresentada na figura 5.18, corresponde ao funcionamento mais simples da arquitectura agora proposta, consistindo na utilização de um único bloco activo ($NC = 1$) para efectuar a pesquisa do vector de movimento óptimo de entre o conjunto composto pelos $(2p)^2$ vectores de movimento candidatos em cada linha da área de pesquisa.

De acordo com a descrição efectuada nas secções anteriores, este tipo de topologia reversível requer um total de

$$T_A = V \cdot T = NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor \times \left\lfloor \frac{2p}{NC} \right\rfloor \tag{5.16a}$$

$$= 2p \times 2p \tag{5.16b}$$

$$= (2p)^2 \tag{5.16c}$$

ciclos de relógio para efectuar o processamento completo dum macrobloco de referência.

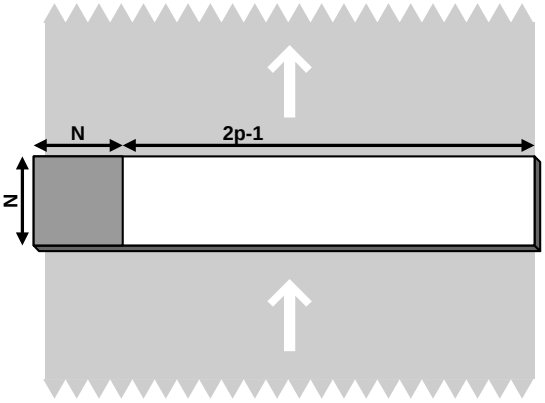


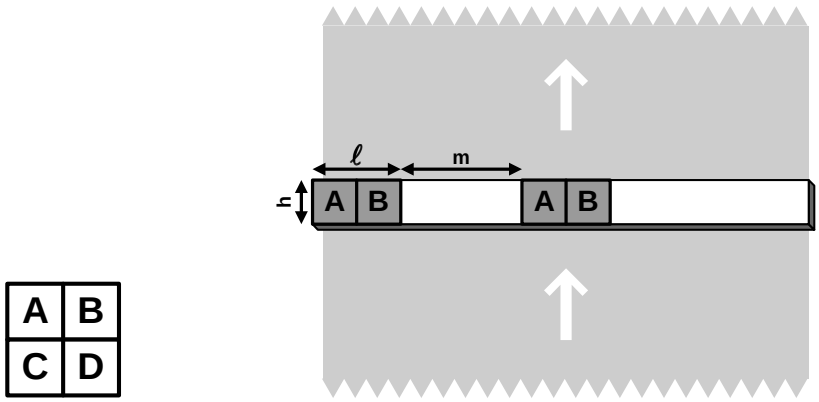
Figura 5.18: Esquema de processamento da topologia do tipo A.

5.4.3 Topologia do tipo B

Esta topologia, apresentada na figura 5.19, tem a vantagem de permitir reduzir os custos de *hardware* requeridos, em particular, o número de elementos de memória necessários para efectuar o deslocamento dos pixels da área de pesquisa, utilizando, apenas, $h = \frac{N}{2}$ linhas de elementos processadores. Para além disso, e para evitar uma diminuição do seu nível de desempenho, utiliza dois blocos activos ($NC = 2$). O macrobloco de referência é, por isso, dividido em duas secções rectangulares, correspondentes às secções AB e CD, ilustradas na figura 5.19(a). Assim, o processamento do macrobloco é, agora, dividido em duas fases, que são executadas alternadamente, dependente do sentido de deslocamento dos pixels da área de pesquisa:

- numa primeira fase, correspondente ao deslocamento dos pixels da área de pesquisa no sentido dos ponteiros do relógio, são colocados nos registos correspondentes aos pixels do macrobloco de referência dos elementos processadores activos os valores correspondentes à secção AB, efectuando-se o cálculo do valor absoluto das diferenças entre estes valores e os respectivos pixels da área de pesquisa.
- na segunda fase, correspondente ao deslocamento dos pixels da área de pesquisa em sentido contrário, são colocados nos registos dos elementos processadores activos os valores correspondentes à secção CD do macrobloco de referência.

A excepção a este tipo de funcionamento é verificada durante as primeiras e últimas $h = \frac{N}{2}$ linhas de pesquisa, pois enquanto que durante as primeiras linhas apenas é necessário calcular as diferenças correspondentes à secção AB, nas últimas $\frac{N}{2}$ linhas é, apenas, necessário calcular as diferenças correspondentes à secção CD do macrobloco de referência.



(a) (b)

Figura 5.19: Esquema de processamento da topologia do tipo B. **(a)** - Subdivisão do macrobloco em 2 secções: AB e CD; **(b)** - Esquema de processamento.

No entanto, este tipo de topologia terá, como custos da redução de *hardware* verificada ao nível do sistema constituído pelos blocos activos e passivos, um aumento da complexidade dos blocos correspondentes aos somadores em árvore e ao comparador. O número de ciclos de relógio requerido por esta topologia será o correspondente a três parcelas distintas:

$$T_B = T_{B1} + T_{B2} + T_{B3}, \tag{5.17}$$

em que T_{B1} e T_{B3} correspondem ao processamento das primeiras $\frac{N}{2}$ linhas e das últimas $\frac{N}{2}$ linhas, respectivamente, e T_{B2} representa o número de ciclos de relógio necessário para processar as restantes linhas:

$$T_{B1} = T_{B3} = h \times T = \frac{N}{2} \times \left\lfloor \frac{2p}{NC} \right\rfloor = \frac{Np}{2} \tag{5.18}$$

$$\begin{aligned} T_{B2} &= 2 \times (V - h) \times T && (5.19a)\\ &= 2 \times \left(NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor - h \right) \times \left\lfloor \frac{2p}{NC} \right\rfloor && (5.19b)\\ &= 2 \times \left(2 \cdot \left\lfloor \frac{2p}{2} \right\rfloor - \frac{N}{2} \right) \times \left\lfloor \frac{2p}{2} \right\rfloor && (5.19c)\\ &= 2p \left(2p - \frac{N}{2} \right) && (5.19d) \end{aligned}$$

$$\begin{aligned} T_B &= T_{B1} + T_{B2} + T_{B3} && (5.20a)\\ &= \frac{Np}{2} + 2p \left(2p - \frac{N}{2} \right) + \frac{Np}{2} && (5.20b)\\ &= (2p)^2 && (5.20c) \end{aligned}$$

A equação 5.20c permite concluir que esta topologia requer, exactamente, o mesmo número de ciclos de relógio do que a topologia do tipo A, considerada anteriormente. Este era o resultado esperado, pois embora a estrutura tenha apenas metade dos registos utilizados para o deslocamento dos pixels da área de pesquisa, esta topologia tem precisamente o mesmo número de elementos processadores activos que a topologia do tipo A.

5.4.4 Topologia do tipo C

A topologia do tipo C é, à excepção do número de blocos activos (NC), em tudo idêntica à topologia do tipo A. Neste caso, considerou-se uma implementação constituída por $NC = 2$ blocos activos, requerendo, por isso, o dobro dos requisitos de *hardware* do que a topologia do tipo A (ver figura 5.20).

O número de ciclos necessário para efectuar a procura do macrobloco candidato óptimo é, neste caso, de:

$$T_C = V \times T \tag{5.21a}$$

$$= NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor \times \left\lfloor \frac{2p}{NC} \right\rfloor \tag{5.21b}$$

$$= 2 \cdot \left\lfloor \frac{2p}{2} \right\rfloor \times \left\lfloor \frac{2p}{2} \right\rfloor \tag{5.21c}$$

$$= \frac{(2p)^2}{2} \tag{5.21d}$$

Como seria de esperar, esta topologia requer, apenas, metade dos ciclos de relógio requeridos pela topologia do tipo A, justificando, assim, a utilização do dobro dos recursos de *hardware*.

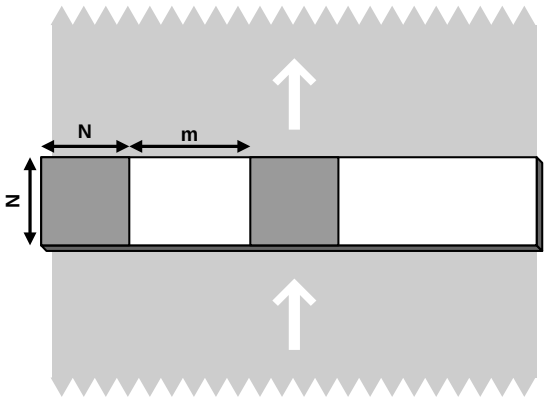


Figura 5.20: Esquema de processamento da topologia do tipo C.

5.4.5 Topologia não reversível

Para estabelecer um termo de comparação entre as topologias descritas anteriormente, optou-se por considerar o caso de uma topologia baseada na mesma arquitectura que as topologias anteriormente descritas mas não fazendo, neste caso, uso do esquema de processamento reversível em *"zig-zag"*. Assim, considera-se agora nesta topologia que os pixels da área de pesquisa são deslocados, apenas, num único sentido (ver figura 5.21). Para além disso, considerou-se o mesmo número de elementos processadores, integrados num único bloco activo ($NC = 1$).

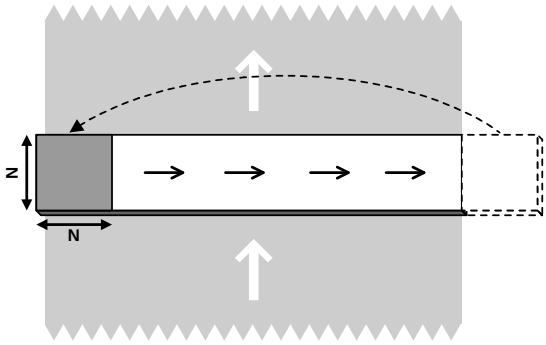


Figura 5.21: Esquema de processamento da topologia não reversível.

Na verdade, este tipo de topologia apresenta características muito semelhantes às da arquitectura proposta por Hsieh [22], descrita na secção 4.4.1. Como consequência, torna-se, também agora, necessário considerar a existência de ℓ ciclos de relógio adicionais para processar cada linha de macroblocos candidatos, gastos a transportar os pixels processados da área de pesquisa para a extremidade oposta da estrutura constituída pelos blocos activo e passivo.

Assim, o número total de ciclos de relógio requerido por esta topologia será de:

$$T_{nR} = V \times (T + \ell) \tag{5.22a}$$

$$= NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor \times \left(\left\lfloor \frac{2p}{NC} \right\rfloor + N \right) \tag{5.22b}$$

$$= 2p \times (2p + N) \tag{5.22c}$$

$$= (2p)^2 + 2pN \tag{5.22d}$$

5.4.6 Comparação das características das diferentes topologias

Na tabela 5.2 apresenta-se o número de ciclos requerido por cada uma das topologias propostas nas secções anteriores. Estes valores foram utilizados para traçar os gráficos correspondentes à relação entre o número de ciclos de relógio requerido por cada uma

Topologia	Número de ciclos
A	$(2p)^2$
B	$(2p)^2$
C	$\frac{1}{2}(2p)^2$
nR	$(2p)^2 + 2pN$

Tabela 5.2: Número de ciclos necessário por cada uma das topologias consideradas para processar um macrobloco de referência.

das topologias consideradas e a razão entre a dimensão do macrobloco de referência (N) e o deslocamento máximo considerado em cada direcção na área de pesquisa (p): $k = \frac{p}{N}$ (ver figura 5.22). É de notar que todos os valores apresentados são relativos ao valor requerido pela topologia não reversível, justificando, assim, o carácter porcentual dos valores obtidos.

Tal como se pode observar, os gráficos obtidos mostram que a diferença do número de ciclos para as várias topologias pode ser bastante significativa, sendo de 66% para $p = N$ nas topologias A e B e de 33% na topologia C.

Convém notar que na análise efectuada tomou-se como pressuposto a utilização do modo de transferência transparente dos dados correspondentes aos pixels da área de pesquisa descrito na secção 5.3.2, não tendo sido levado em consideração o tempo de pré-carregamento dos registos de deslocamento que compõem o sistema constituído pelos

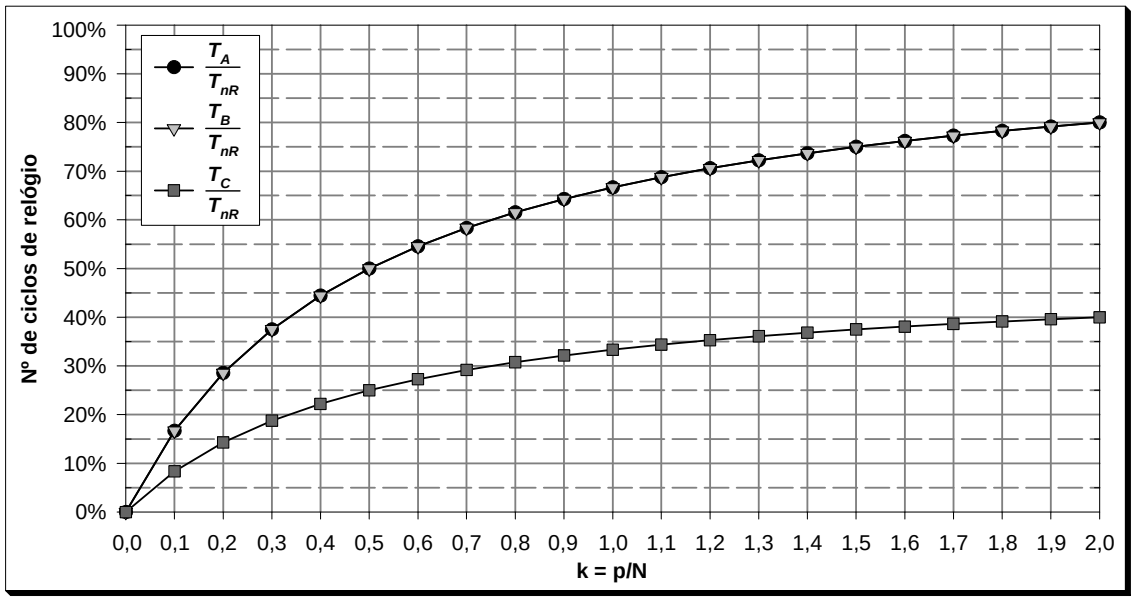


Figura 5.22: Relação entre o número de ciclos de relógio requerido por cada umas das topologias consideradas para processar um macrobloco de referência.

blocos activos e passivos. Consequentemente, os valores apresentados serão significativamente maiores se um sistema de transferência tradicional dos pixels da área de pesquisa for implementado no processador. Este tempo dependerá, naturalmente, do número de linhas de elementos processadores que constituem o sistema. Por isso, será de esperar que a topologia do tipo B seja menos penalizada do que as topologias do tipo A e C:

$$T_{pré\_A} = T_{pré\_C} = T_{pré\_nR} = \left. h \times L' \right|_{h=N} \tag{5.23a}$$

$$= N \times \left[NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + (\ell - 1) \right] \tag{5.23b}$$

$$= N \times (2p + N - 1) \tag{5.23c}$$

$$T_{pré\_B} = \left. h \times L' \right|_{h=\frac{N}{2}} \tag{5.24a}$$

$$= \frac{N}{2} \times \left[NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + (\ell - 1) \right] \tag{5.24b}$$

$$= \frac{N}{2} \times (2p + N - 1) \tag{5.24c}$$

Neste caso, o número de ciclos de relógio requerido por cada topologia será o apresentado na tabela 5.3. Na figura 5.23 representou-se, uma vez mais, o número de ciclos requerido por cada uma das topologias consideradas, tomando agora em consideração o tempo de pré-carregamento. Os gráficos obtidos permitem concluir que as relações entre o número requerido de ciclos de relógio agora consideradas podem ser bastante diferentes das encontradas anteriormente. Tal verifica-se, sobretudo, nas situações correspondentes a áreas de pesquisa de pequena dimensão (k pequeno). Nos casos correspondentes a áreas de pesquisa mais extensas verifica-se um comportamento semelhante ao anteriormente encontrado. Esta variação justifica-se pelo facto da parcela correspondente ao pré-carregamento dos registos tomar um peso dominante para o caso de k tomar valores pequenos. Na situação correspondente a áreas de pesquisa mais extensas (k ele-

Topologia	Número de ciclos
A	$(2p)^2 + 2pN + N\ (N-1)$
B	$(2p)^2 + pN + \frac{1}{2} N\ (N-1)$
C	$\frac{1}{2}(2p)^2 + 2pN + N\ (N-1)$
nR	$(2p)^2 + 4pN + N\ (N-1)$

Tabela 5.3: Número de ciclos necessário por cada uma das topologias consideradas para processar um macrobloco de referência.

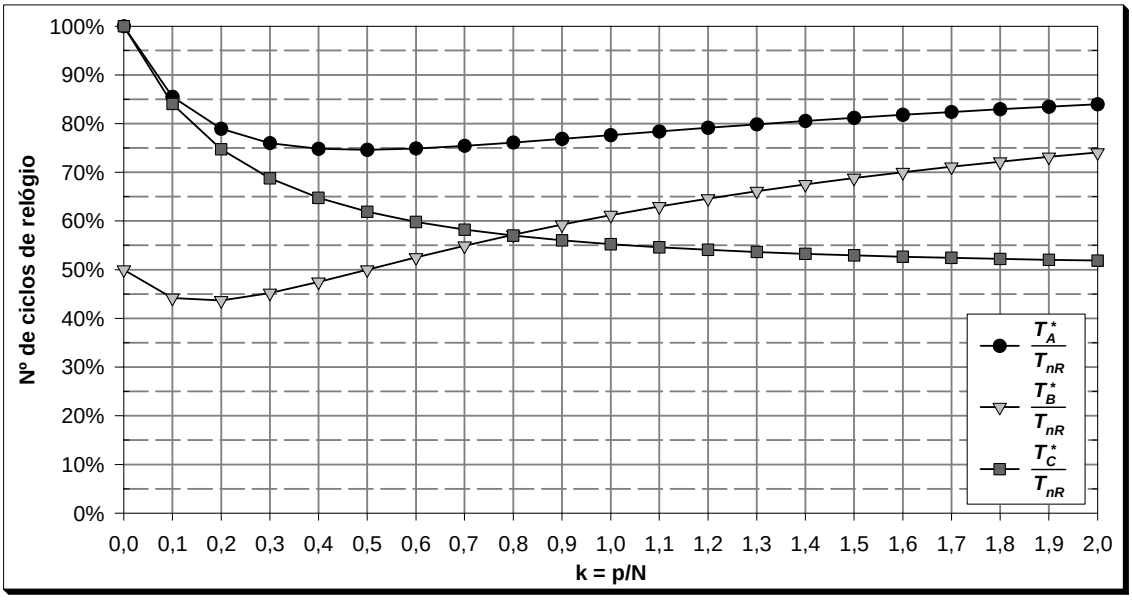


Figura 5.23: Relação entre o número de ciclos de relógio requerido por cada uma das topologias consideradas para processar um macrobloco de referência, tomando em consideração o tempo de pré-carregamento dos registos de deslocamento.

vado) verifica-se, pelo contrário, a dominância da parcela correspondente ao algoritmo de procura exaustiva propriamente dito.

Capítulo 6

Unidades Aritméticas

Conteúdo

6.1 Introdução

Conforme se referiu no capítulo 1, o objectivo fundamental do trabalho apresentado é o desenvolvimento de um processador dedicado para estimação de movimento com níveis elevados de desempenho, nomeadamente, no que diz respeito ao tempo de processamento. Para tal, torna-se determinante a utilização de unidades de processamento optimizadas para o tipo de operações requeridas pelos vários blocos de constituem o processador. As várias unidades aritméticas utilizadas no cálculo da medida de similaridade assumem, por isso, uma importância fundamental, nomeadamente, as estruturas somadoras e subtractoras utilizadas nos elementos processadores activos.

Decidiu-se, por isso, efectuar um estudo detalhado de várias arquitecturas rápidas de somadores, para avaliar qual o tipo de arquitectura que apresenta as características mais apropriadas. Foram consideradas arquitecturas do tipo *carry-lookahead* em árvore binária [27, 28, 29], baseadas em sistemas de numeração redundante [30, 31, 32] e *prefix-adder* [**?** 33], que se apresentam nas secções que se seguem. Para comparar as diferentes arquitecturas, optou-se por utilizar o modelo apresentado na tabela 6.1, que relaciona a área ocupada pelo circuito com o tempo de propagação de cada célula lógica [34**?**].

Célula lógica	**Símbolo**	**Área (A)**	**Tempo (T)**
Inversor, *buffer* [1]	INV BUFF	0	0
Portas básicas de 2 entradas	AND NAND OR NOR	1	1
Portas compostas de 2 entradas	XOR XNOR	2	2
Portas de m entradas (derivadas de portas lógicas mais simples)	ANDm NANDm ORm NORm XORm XNORm	$m-1$	$\lceil \log_2 m \rceil$
Ligações entre portas lógicas, decorrentes do mapeamento do circuito		*Não considerado*	

Tabela 6.1: Relação entre *área* ocupada e *tempo* de propagação de cada célula lógica, segundo o modelo utilizado [34].

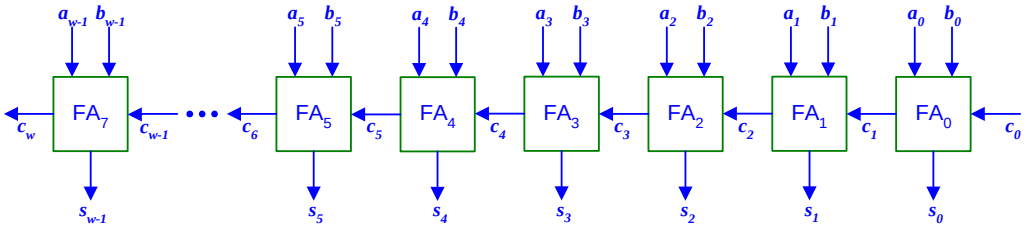
[1] Para efeitos de comparação de circuitos, estes dois elementos são ignorados.

Como referência, considerou-se o somador mais simples (mas também o mais lento), do tipo *carry-propagate*[2], que é o somador *ripple-carry*. Este somador, apresentado na figura 6.1(a), é constituído por um circuito com w somadores completos de bit (*full-adder*) ligados em série, efectuando o cálculo da soma de dois operandos de w bits: $A = a_{w-1}, \ldots, a_0$ e $B = b_{w-1}, \ldots, b_0$, com o sinal de entrada de *carry* (c_{in}), de acordo as seguintes expressões:

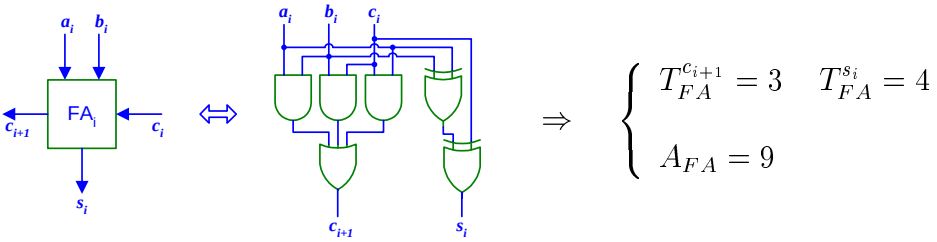
$$s_i = a_i \oplus b_i \oplus c_i \tag{6.1}$$

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i \tag{6.2}$$

Cada um dos blocos do tipo *full-adder* apresenta uma estrutura interna correspondente à apresentada na figura 6.1(b).



(a) Ligação em série de w blocos do tipo *full-adder*.



$$\Rightarrow \begin{cases} T_{FA}^{c_{i+1}} = 3 \quad T_{FA}^{s_i} = 4 \\ A_{FA} = 9 \end{cases}$$

(b) Estrutura interna do bloco do tipo *full-adder*.

Figura 6.1: Somador do tipo *ripple-carry*.

Utilizando o modelo descrito, este circuito tem um tempo de processamento:

$$\begin{cases} T^{c_{out}} = T^{c_w} = w \cdot T_{FA}^{c_{i+1}} = 3w \\ T^{s_w} = (w-1) \cdot T_{FA}^{c_{i+1}} + T_{FA}^{s_i} = 3(w-1) + 4 = 3w + 1 \approx 3w \end{cases} \tag{6.3}$$

e ocupa uma área:

$$A = w \cdot A_{FA} = 9w \tag{6.4}$$

[2]Do Inglês propagação do bit de transporte.

6.2 Somador do tipo *carry-lookahead*

O somador *carry-lookahead* (CLA) com uma estrutura em árvore binária apresenta, como principal característica, o facto de ser constituído por apenas $\lceil \log_2 w \rceil$ níveis hierárquicos (sendo w o número de bits dos operandos). Este facto contrasta, significativamente, com os $9w$ níveis de lógica requeridos pela arquitectura *ripple-carry*, apresentando, também, uma estrutura completamente regular, conforme se ilustra na figura 6.2(a).

O princípio de funcionamento deste tipo de somadores assenta no cálculo do vector w-dimensional constituído pelos w bits de *carry*, gerados e propagados ao longo da estrutura a partir do valor dos operandos de entrada A e B. O cálculo dos w bits de *carry* c_i baseia-se nas seguintes expressões:

$$c_1 = g_0 + p_0 c_0 \tag{6.5a}$$

$$c_2 = g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0 \tag{6.5b}$$

$$\vdots$$

$$c_w = g_{w-1} + p_{w-1} g_{w-2} + p_{w-1} p_{w-2} g_{w-3} + \cdots + p_{w-1} p_{w-2} \cdots p_0 g_0 + p_{w-1} p_{w-2} \cdots p_1 p_0 c_0 \tag{6.5c}$$

sendo $g_i = a_i.b_i$ e $p_i = a_i + b_i$.

Tal como se referiu anteriormente, esta estrutura pode ser implementada de uma forma hierárquica. De facto, designando por $G_{i,k}$ o valor do sinal de *carry* gerado entre as entradas i e k, e por $P_{i,k}$ o valor do sinal de propagação entre estas duas entradas, é possível mostrar que o vector de *carry* pode ser calculado de uma forma recursiva, utilizando as seguintes expressões:

$$c_{k+1} = G_{i,k} + P_{i,k} \cdot c_i \tag{6.6}$$

$$G_{i,k} = G_{j+1,k} + P_{j+1,k} \cdot G_{i,j} \tag{6.7}$$

$$P_{i,k} = P_{i,j} \cdot P_{j+1,k} \tag{6.8}$$

sendo $G_{i,i} = g_i$, $P_{i,i} = p_i$ e $i \leq j < k$.

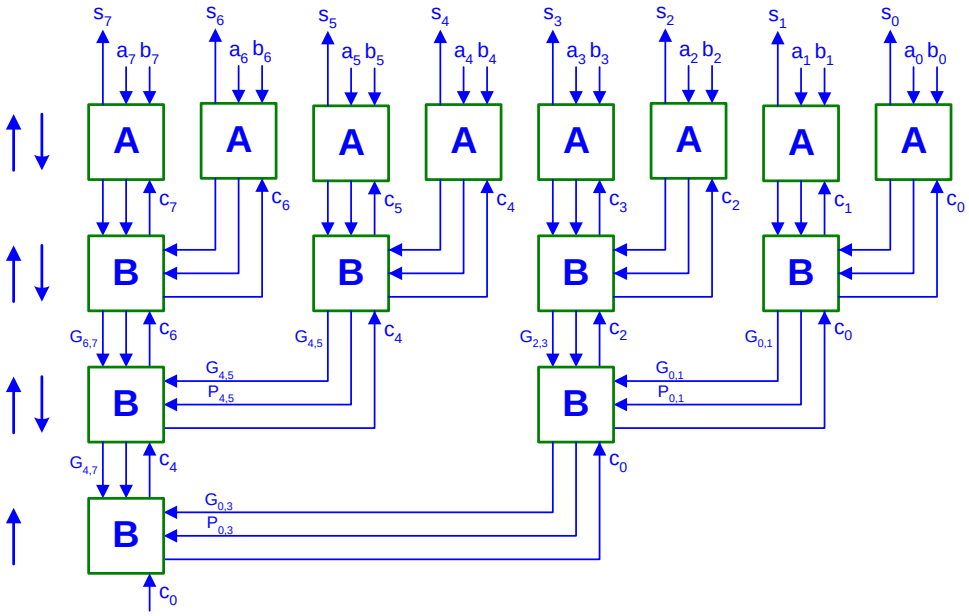
Calculado o vector w-dimensional constituído por todos os bits de *carry*, é possível obter facilmente o resultado da soma, atendendo a que:

$$s_i = a_i \oplus b_i \oplus c_i \tag{6.9}$$

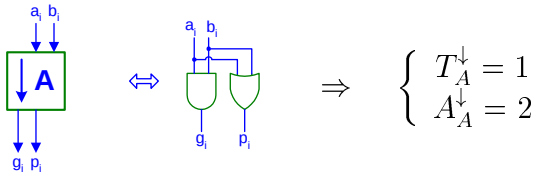
Tomando em consideração a formulação descrita, apresenta-se, na figura 6.2, a estrutura hierárquica do somador com estrutura em árvore. O somador tem duas fases de funcionamento:

- na primeira fase, é feito o cálculo dos vários valores de P e G a partir dos valores p_i e g_i, fazendo uso das equações 6.7 e 6.8;
- na segunda, utilizam-se estes valores de P e G para, através da equação 6.6, calcularem-se todos os bits do vector de *carry*.

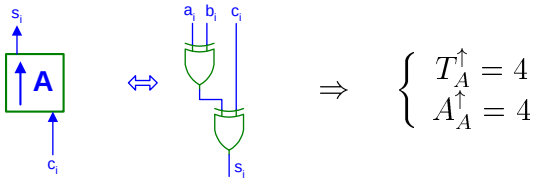
Durante a primeira parte do algoritmo, os valores dos bits dos operandos $a_{w-1}, \ldots, a_0$ e $b_{w-1}, \ldots, b_0$ são introduzidos nos blocos do tipo A na parte superior da estrutura em



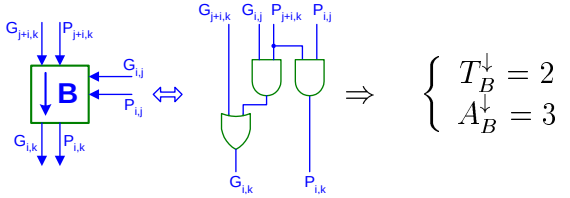
(a) Estrutura hierárquica do somador do tipo *carry-lookahead*.



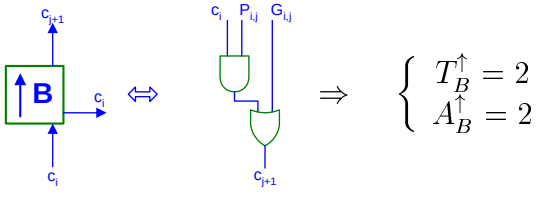
(b) Bloco A: circuito descendente.



(c) Bloco A: circuito ascendente.



(d) Bloco B: circuito descendente.



(e) Bloco B: circuito ascendente.

Figura 6.2: Diagrama de blocos do somador do tipo *carry-lookahead*, segundo uma implementação baseada em operandos de entrada com precisão de 8 bits.

árvore e convertidos nos valores p_i e g_i. Estes valores são, em seguida, utilizados para o cálculo dos vários valores de P e G nos blocos do tipo B, segundo um fluxo de dados com o sentido descendente na estrutura em árvore.

A segunda parte do cálculo inicia-se, então, com a introdução do bit de *carry* de entrada (c_0) no bloco do tipo B localizado no extremo inferior do somador. Este valor, juntamente com o par de valores (P, G) existente em cada bloco do tipo B, é utilizado para calcular o vector w-dimensional correspondente aos vários bits de *carry*, segundo um esquema de processamento com o sentido ascendente. Finalmente, o vector de bits de *carry* é utilizado nos vários blocos do tipo A para calcular o resultado da soma, de acordo com a equação 6.9. Conforme se pode verificar na figura 6.2(a), não existe uma saída com o sinal de $carry_{out}$ do bit mais significativo da soma. Contudo, este sinal pode ser facilmente calculado utilizando o valor dos bits mais significativos dos operandos de entrada e do bit de carry c_w, de acordo com a seguinte expressão:

$$c_w = a_{w-1} \cdot b_{w-1} + a_{w-1} \cdot c_{w-1} + b_{w-1} \cdot c_{w-1} \tag{6.10}$$

Esta expressão pode ser implementada com um circuito digital constituído por três portas lógicas do tipo AND e uma porta do tipo OR de 3 entradas, caracterizado por $T_{c_w} = 3$ e $A_{c_w} = 5$.

Assim, de acordo com os sentidos de processamento, o nível de blocos do tipo A e os $\lceil \log_2 w \rceil$ níveis de blocos do tipo B da árvore apresentada na figura 6.2(a) contribui para o tempo total de processamento com:

$$\begin{aligned} T_S &= T_A^{\downarrow} + (\lceil \log_2 w \rceil - 1) \times T_B^{\downarrow} + \lceil \log_2 w \rceil \times T_B^{\uparrow} + T_A^{\uparrow} && \text{(6.11a)} \\ &= 1 + (\lceil \log_2 w \rceil - 1) \times 2 + \lceil \log_2 w \rceil \times 2 + 4 && \text{(6.11b)} \\ &= 4. \lceil \log_2 w \rceil + 3 && \text{(6.11c)} \end{aligned}$$

O tempo de processamento do resultado do *carry* de saída é dado por:

$$\begin{aligned} T_{c_{out}} &= T_A^{\downarrow} + (\lceil \log_2 w \rceil - 1) \times T_B^{\downarrow} + \lceil \log_2 w \rceil \times T_B^{\uparrow} + T_{c_{w+1}} && \text{(6.12a)} \\ &= 1 + (\lceil \log_2 w \rceil - 1) \times 2 + \lceil \log_2 w \rceil \times 2 + 3 && \text{(6.12b)} \\ &= 4. \lceil \log_2 w \rceil + 2 && \text{(6.12c)} \end{aligned}$$

Convém notar que o número de entradas num somador em árvore deste tipo é uma potência inteira de 2, o que pode, por vezes, dar origem a um sub-aproveitamento dos recursos de *hardware* utilizados. Assim, ao longo da exposição que se segue, passar-se-á a denominar por $\hat{w}$ o número de entradas efectivamente requerido por este tipo de estruturas, sendo $\hat{w} = 2^{\lceil \log_2 w \rceil}$.

A área ocupada por este circuito somador será, então, dada por:

$$A = \hat{w} \times A_A + (\hat{w} - 1) \times A_B + A_{c_{w+1}} \tag{6.13a}$$
$$= \hat{w} \times (2 + 4) + (\hat{w} - 1) \times (3 + 2) + 5 \tag{6.13b}$$
$$= 11\hat{w} \tag{6.13c}$$

Conclui-se, assim, que de acordo com o modelo descrito na tabela 6.1, esta topologia apresenta um tempo total de processamento da ordem de $4.\lceil \log_2 w \rceil + 3$ níveis de lógica, o que representa uma melhoria substancial em relação a uma topologia do tipo *ripple-carry*, caracterizada por um total de $3w$ níveis de lógica. Esta melhoria será tanto maior quanto maior for o número de bits dos operandos (w). O custo a pagar por esta melhoria, em termos de rapidez, reflecte-se na área ocupada pelo circuito, pois enquanto que uma topologia do tipo *ripple-carry* requer uma área de apenas $9w$ unidades, esta arquitectura rápida requer uma área de $11\hat{w}$.

6.3 Somador baseado em aritmética redundante

Uma outra topologia de somador rápido considerada foi a baseada na utilização de sistemas de numeração redundante [30, 31, 32], tomando partido das características de rapidez do conversor de representação redundante para representação binária. Esta rapidez deve-se, como se demonstrará, à substituição dos blocos somadores completos de bit (*full-adder*) de um somador convencional por blocos mais simples e rápidos do tipo multiplexadores. Esta característica, aliada à utilização de estruturas de somadores em árvore permite, então, obter somadores rápidos.

Considere-se a operação $X = A + B$, em que $X = x_{w-1}, \ldots, x_0$, $A = a_{w-1}, \ldots, a_0$ e $B = b_{w-1}, \ldots, b_0$ representam números binários representados com w bits. Esta operação pode ser expressa de uma forma alternativa:

$$X = A + B = A - (-B) \tag{6.14a}$$
$$= A - \left(\overline{B} + 1\right) \tag{6.14b}$$
$$= \left(A - \overline{B}\right) + (-1) \tag{6.14c}$$

Na equação 6.14b, o valor simétrico do operando B é expresso através da soma do seu complemento para um ($\overline{B}$) e 1. Por sua vez, na equação 6.14c, o valor da parcela $\left(A - \overline{B}\right)$ pode ser interpretado como um número representado em numeração redundante, onde cada dígito $\left(a_i, \overline{b_i}\right)$ assume o valor $\left(a_i - \overline{b_i}\right)$. Da mesma forma, a parcela (-1) pode, também, ser interpretada como um número representado no mesmo sistema de numeração

redundante, com a representação $(0,1)$. De acordo com o descrito, qualquer número pode ser representado utilizando este sistema de numeração da forma $\left(a_i, \overline{b_i}\right)$ [31, 35, 36], sendo a sua conversão para o sistema de numeração binário em complemento para dois efectuada da seguinte forma: $x = x^+ - x^-$, sendo $x^+ = a_i$ e $x^- = \overline{b_i}$.

Nesta representação, os valores $(1,1)$ e $(0,0)$ representam, assim, o número 0. Contudo, mostra-se que se a representação $(1,1)$ não for permitida, é possível obter circuitos mais rápidos e mais simples [31, 35]. Uma forma simples de se eliminar a representação $(1,1)$ consiste na substituição por $(0,0)$, antes de se efectuar a conversão. Este processo pode ser facilmente implementado utilizando o circuito apresentado na figura 6.3.

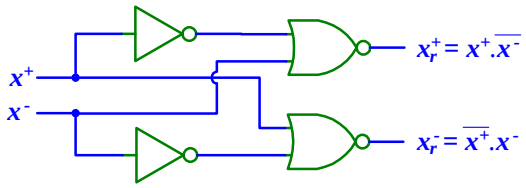


Figura 6.3: Conversão dos dígitos (x^+, x^-) em (x_r^+, x_r^-), para substituir a representação $(1,1)$ por $(0,0)$.

A conversão de um número representado no sistema de numeração redundante com w dígitos (x^+, x^-) num número representado em complemento para dois (x) pode ser realizada com w somadores completos de bit, obtendo-se à saída destes somadores o valor $x = x^+ - x^-$ (ver figura 6.4(a)). Cada célula calcula, então, o valor $x_i = x_i^+ - x_i^- - c_i = 2c_{i+1} + s_i$. Os bits de *carry* e de soma são dados por (ver figura 6.4(b)):

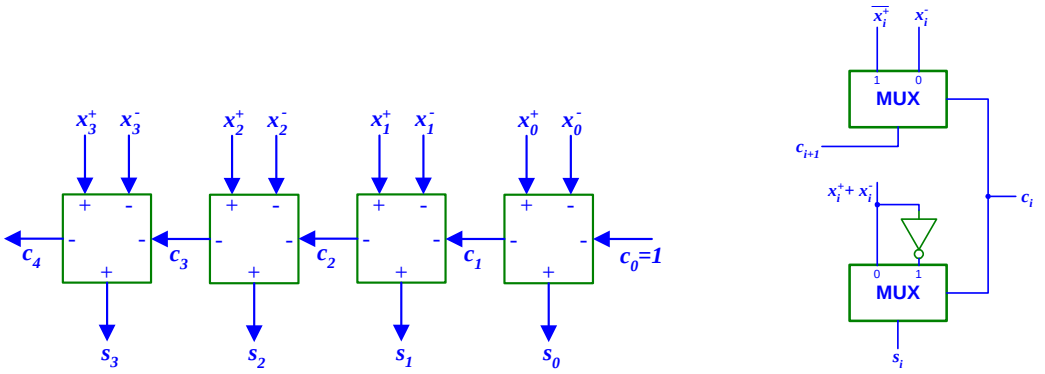
$$c_{i+1} = c_i \cdot \overline{x_i^+} + \overline{c_i} \cdot x_i^- \tag{6.15}$$

$$s_i = c_i \cdot \left(\overline{x_i^+ + x_i^-}\right) + \overline{c_i} \cdot \left(x_i^+ + x_i^-\right) \tag{6.16}$$

As duas equações anteriores permitem uma implementação directa por intermédio de dois multiplexadores tendo, como sinal de selecção, o valor do sinal c_i. Tal é possível devido ao pré-condicionamento apresentado na figura 6.3, que permite explorar situações do tipo *Don't Care*[3] na dedução das equações anteriores. Na figura 6.4(a) apresenta-se a implementação destas equações através de células somadoras do tipo MMP[4]. A função lógica de cada uma destas células pode ser implementada com dois multiplexadores, como se apresenta na figura 6.4(b). O valor inicial do bit de *carry* de entrada deverá ser $c_0 = 1$, correspondente à soma do valor (-1) (ver equação 6.14c).

Convém notar que o circuito apresentado na figura 6.4 tem um funcionamento correcto apenas se se garantir que os bits das entradas são convenientemente modificados através

[3] Do Inglês Indiferente.
[4] Do Inglês *Minus-Minus-Plus*.



(a) Ligação em série de células MMP.

(b) Estrutura duma célula MMP.

Figura 6.4: Somador de 4 bits baseado no sistema de numeração redundante.

do circuito de pré-condicionamento apresentado na figura 6.3. Para além disso, a geração do bit de *carry* é de fundamental importância, pois tal como acontece nas topologias do tipo *carry-propagate* [37**?**], os bits de soma podem apenas ser calculados à medida que se calcula o valor dos bits de *carry*. Consequentemente, têm sido propostas soluções que aplicam a este tipo de topologias os métodos para o cálculo rápido dos bits de *carry* utilizados nos somadores binários tradicionais [31, 35].

Como a conversão entre a representação redundante e a representação binária em complemento para dois apresenta semelhanças evidentes com a operação de soma binária, torna-se possível conceber conversores rápidos baseados em técnicas utilizadas no somador do tipo *carry-lookahead*, utilizando, agora, circuitos simples do tipo dos apresentados na figura 6.4(b). Este tipo de aproximação torna-se, assim, bastante mais eficiente, pois as células apresentadas na figura 6.4(b) (multiplexadores) são mais simples do que as células constituintes de um somador binário tradicional. Contudo, devido à estrutura em árvore binária utilizada, torna-se necessário efectuar a extensão do número de bits dos operandos sempre que w não é uma potência inteira de 2. Consequentemente, nas expressões apresentadas deve ser utilizado o valor $\hat{w}$ sempre que se refere o número de bits dos operandos.

Para ilustrar a implementação de um somador baseado na representação redundante, considere-se, a título de exemplo, um somador em árvore com operandos com 4 bits de precisão. O cálculo dos bits de *carry* pode ser apresentado de uma forma recursiva, fazendo uso, apenas, de blocos do tipo multiplexador (ver figura 6.4(b)), com base na seguinte expressão:

$$c_{i+1} = c_i \cdot \overline{x_i^+} + \overline{c_i} \cdot x_i^- \tag{6.17}$$

Designando por $\overline{f_0(i)} = \overline{x_i^+}$ e $g_0(i) = x_i^-$, obtém-se:

$$
\begin{aligned}
c_{i+2} &= c_{i+1} \cdot \overline{x_{i+1}^+} + \overline{c_{i+1}} \cdot x_{i+1}^- && \text{(6.18a)}\\
&= \left[c_i \cdot \overline{x_i^+} + \overline{c_i} \cdot x_i^-\right] \cdot \overline{x_{i+1}^+} + \left[c_i \cdot x_i^+ + \overline{c_i} \cdot \overline{x_i^-}\right] \cdot x_{i+1}^- && \text{(6.18b)}\\
&= c_i \cdot \underbrace{\left[\overline{x_i^+} \cdot \overline{x_{i+1}^+} + x_i^+ \cdot x_{i+1}^-\right]}_{\overline{f_1(i+1)}} + && \text{(6.18c)}\\
&\quad \overline{c_i} \cdot \underbrace{\left[x_i^- \cdot \overline{x_{i+1}^+} + \overline{x_i^-} \cdot x_{i+1}^-\right]}_{g_1(i+1)}\\
&= c_i \cdot \overline{f_1(i+1)} + \overline{c_i} \cdot g_1(i+1) && \text{(6.18d)}
\end{aligned}
$$

em que:

$$
\overline{f_1(i+1)} = \overline{f_0(i)} \cdot \overline{f_0(i+1)} + f_0(i) \cdot g_0(i+1) \quad \text{(6.19)}
$$

$$
g_1(i+1) = g_0(i) \cdot \overline{f_0(i+1)} + \overline{g_0(i)} \cdot g_0(i+1) \quad \text{(6.20)}
$$

Da mesma forma obtém-se:

$$
\begin{aligned}
c_{i+3} &= c_{i+2} \cdot \overline{x_{i+2}^+} + \overline{c_{i+2}} \cdot x_{i+2}^- && \text{(6.21a)}\\
&= \left[c_i \cdot \overline{f_1(i+1)} + \overline{c_i} \cdot g_1(i+1)\right] \cdot \overline{x_{i+2}^+} + && \text{(6.21b)}\\
&\quad \left[c_i \cdot f_1(i+1) + \overline{c_i} \cdot \overline{g_1(i+1)}\right] \cdot x_{i+2}^-\\
&= c_i \cdot \left[\overline{f_1(i+1)} \cdot \overline{x_{i+2}^+} + f_1(i+1) \cdot x_{i+2}^-\right] + && \text{(6.21c)}\\
&\quad \overline{c_i} \cdot \left[g_1(i+1) \cdot \overline{x_{i+2}^+} + \overline{g_1(i+1)} \cdot x_{i+2}^-\right]\\
&= c_i \cdot \underbrace{\left[\overline{f_1(i+1)} \cdot \overline{f_0(i+2)} + f_1(i+1) \cdot g_0(i+2)\right]}_{\overline{f_2(i+2)}} + && \text{(6.21d)}\\
&\quad \overline{c_i} \cdot \underbrace{\left[g_1(i+1) \cdot \overline{f_0(i+2)} + \overline{g_1(i+1)} \cdot g_0(i+2)\right]}_{g_2(i+2)}\\
&= c_i \cdot \overline{f_2(i+2)} + \overline{c_i} \cdot g_2(i+2) && \text{(6.21e)}
\end{aligned}
$$

De forma semelhante, obtém-se o valor de c_{i+4}:

$$c_{i+4} = c_{i+3} \cdot \overline{x^{+}_{i+3}} + \overline{c_{i+3}} \cdot x^{-}_{i+3} \tag{6.22a}$$

$$= \left[c_i \cdot \overline{f_2(i+2)} + \overline{c_i} \cdot g_2(i+2)\right] \cdot \overline{f_0(i+3)} + \tag{6.22b}$$

$$\left[c_i \cdot f_2(i+2) + \overline{c_i} \cdot \overline{g_2(i+2)}\right] \cdot g_0(i+3)$$

$$= c_i \cdot \underbrace{\left[\overline{f_2(i+2)} \cdot \overline{f_0(i+3)} + f_2(i+2) \cdot g_0(i+3)\right]}_{\overline{f_2(i+3)}} + \tag{6.22c}$$

$$\overline{c_i} \cdot \underbrace{\left[g_2(i+2) \cdot \overline{f_0(i+3)} + \overline{g_2(i+2)} \cdot g_0(i+3)\right]}_{g_2(i+3)}$$

$$= c_i \cdot \overline{f_2(i+3)} + \overline{c_i} \cdot g_2(i+3) \tag{6.22d}$$

Para reduzir o número de níveis da estrutura em árvore do somador, atendendo à equação 6.21d, é possível manipular a expressão anterior, obtendo-se:

$$c_{i+4} = c_i \cdot \left[\left[\overline{f_1(i+1)} \cdot \overline{f_0(i+2)} + f_1(i+1) \cdot g_0(i+2)\right] \cdot \overline{f_0(i+3)} + \tag{6.23a}$$

$$\left[\overline{f_1(i+1)} \cdot f_0(i+2) + f_1(i+1) \cdot \overline{g_0(i+2)}\right] \cdot g_0(i+3)\right] +$$

$$\overline{c_i} \cdot \left[\left[g_1(i+1) \cdot \overline{f_0(i+2)} + \overline{g_1(i+1)} \cdot g_0(i+2)\right] \cdot \overline{f_0(i+3)} + \right.$$

$$\left.\left[g_1(i+1) \cdot f_0(i+2) + \overline{g_1(i+1)} \cdot \overline{g_0(i+2)}\right] \cdot g_0(i+3)\right]$$

$$= c_i \cdot \left[\overline{f_1(i+1)} \cdot \underbrace{\left[\overline{f_0(i+2)} \cdot \overline{f_0(i+3)} + f_0(i+2) \cdot g_0(i+3)\right]}_{\overline{f_1(i+3)}} + \tag{6.23b}\right.$$

$$\left. f_1(i+1) \cdot \underbrace{\left[g_0(i+2) \cdot \overline{f_0(i+3)} + \overline{g_0(i+2)} \cdot g_0(i+3)\right]}_{g_1(i+3)}\right] +$$

$$\overline{c_i} \cdot \left[g_1(i+1) \cdot \underbrace{\left[\overline{f_0(i+2)} \cdot \overline{f_0(i+3)} + f_0(i+2) \cdot g_0(i+3)\right]}_{\overline{f_1(i+3)}} + \right.$$

$$\left. \overline{g_1(i+1)} \cdot \underbrace{\left[g_0(i+2) \cdot \overline{f_0(i+3)} + \overline{g_0(i+2)} \cdot g_0(i+3)\right]}_{g_1(i+3)}\right]$$

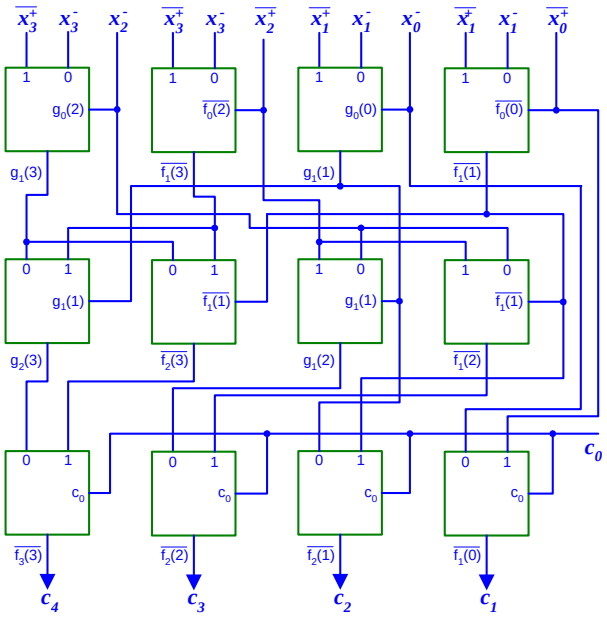
$$= c_i \cdot \left[\overline{f_1(i+1)} \cdot \overline{f_1(i+3)} + f_1(i+1) \cdot g_1(i+3)\right] + \tag{6.23c}$$

$$\overline{c_i} \cdot \left[g_1(i+1) \cdot \overline{f_1(i+3)} + \overline{g_1(i+1)} \cdot g_1(i+3)\right]$$

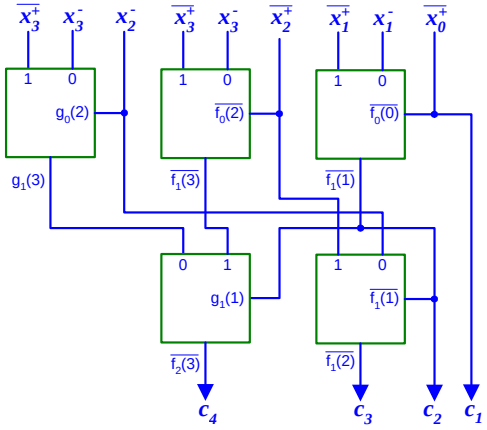
Verifica-se, assim, ser possível calcular todos os sinais de *carry* fazendo uso, apenas, de blocos do tipo multiplexador, através de uma estrutura constituída por $(\log_2 \hat{w}+1)$ níveis de processamento e utilizando um total de $\hat{w} \times (\log_2 \hat{w}+1)$ multiplexadores. O circuito de cálculo dos sinais de *carry* para o caso considerado ($\hat{w}=4$) encontra-se representado na figura 6.5(a).

Contudo, como no caso considerado o valor do bit de entrada de *carry* é conhecido ($c_0=1$), o circuito apresentado na figura 6.5(a) pode ainda ser simplificado, conforme se ilustra na figura 6.5(b). Neste caso, o circuito é composto por apenas 5 multiplexadores e apresenta, apenas, 2 níveis de processamento.

Combinando os dois circuitos apresentados na figura 6.5, pode-se obter um circuito com estrutura em árvore para uma precisão de 8 bits, conforme se ilustra na figura 6.6. Este conversor de 8 bits apresenta uma latência correspondente a 3 níveis de processamento e requer um total de 17 multiplexadores. Pode-se mostrar que, para a extensão de 16 bits, obtém-se um circuito de cálculo em árvore dos sinais de *carry* constituído por 4 níveis de processamento e utilizando um total de 49 multiplexadores [31]. No caso geral, esta arquitectura permite o cálculo de todos os $\hat{w}$ sinais de *carry* utilizando $(\log_2 \hat{w}-1).\hat{w}+1$ multiplexadores, através de uma estrutura com $\log_2 \hat{w}$ níveis [31].



(a) Circuito completo, constituído por $\lceil \log_2 4 \rceil + 1 = 3$ níveis.



(b) Circuito simplificado para o caso de $c_0 = 1$.

Figura 6.5: Circuito de cálculo dos sinais de *carry* para operandos com 4 bits com representação redundante.

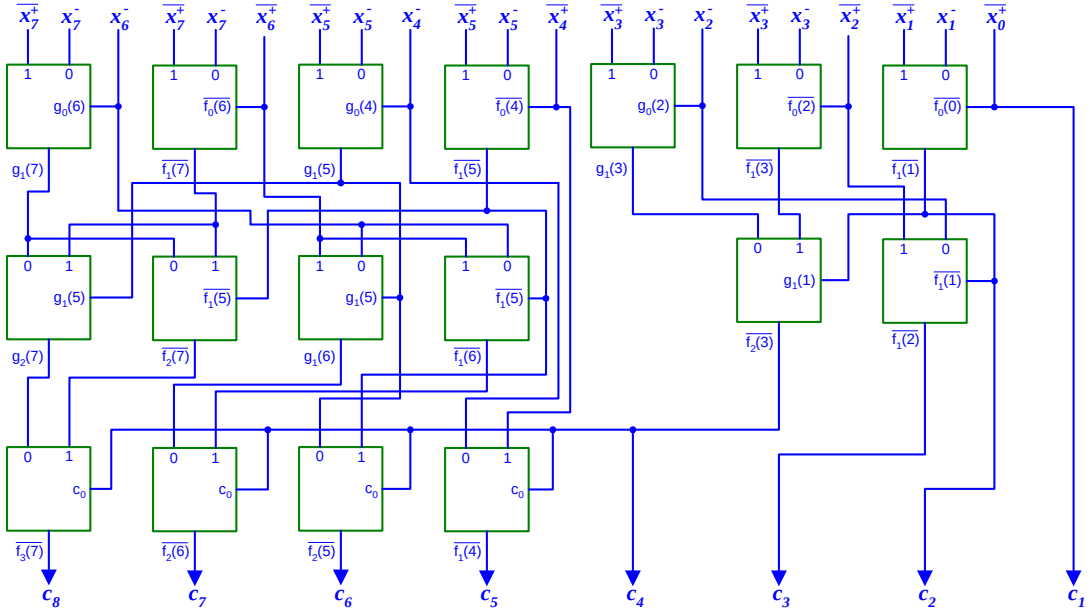


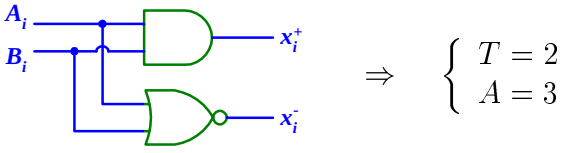
Figura 6.6: Circuito de cálculo dos sinais de *carry* para o caso de uma implementação de um somador com operandos com 8 bits de precisão e $c_0 = 1$.

O circuito somador responsável pelo cálculo da operação $X = A + B$ é constituído por três blocos distintos:

- Bloco de pré-condicionamento das entradas - Sendo $A = a_{w-1}, \ldots, a_0$ e $B = b_{w-1}, \ldots, b_0$ os operandos a adicionar, este bloco efectua as seguintes operações, correspondentes ao circuito apresentado na figura 6.3.

$$\text{①}: \begin{cases} a_i = A_i \\ b_i = \overline{B_i} \end{cases} \quad \text{②}: \begin{cases} x^+ = a_i \cdot \overline{b_i} \\ x^- = \overline{a_i} \cdot b_i \end{cases} \quad \text{① e ②} \Rightarrow \begin{cases} x^+ = a_i \cdot \overline{b_i} = A_i \cdot B_i \\ x^- = \overline{a_i} \cdot b_i = \overline{A_i} \cdot \overline{B_i} = \overline{A_i + B_i} \end{cases}$$

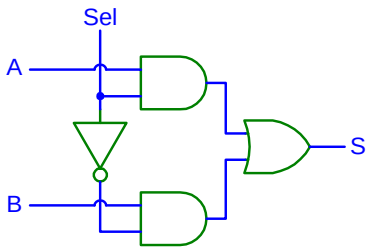
  Na figura 6.7 apresenta-se o circuito correspondente a estas duas operações, para cada um dos w bits dos operandos A e B. De acordo com o modelo descrito na tabela 6.1, obtém-se para este circuito, $T = 1$ e $A = 2\hat{w}$.



$$\Rightarrow \quad \begin{cases} T = 2 \\ A = 3 \end{cases}$$

Figura 6.7: Circuito de pré-condicionamento.

- Circuito de cálculo do vector de *carry* $c_w, \ldots, c_1$ - Considerando cada multiplexador deste circuito implementado com o circuito apresentado na figura 6.8, caracterizado



$$\Rightarrow \quad \begin{cases} T = 2 \\ A = 3 \end{cases}$$

Figura 6.8: Estrutura interna de cada um dos multiplexadores que constituem a estrutura do somador.

por $T = 2$ e $A = 3$, o circuito de cálculo do vector de *carry* apresentará uma estrutura semelhante à apresentada na figura 6.6 e será caracterizado por:

$$\begin{cases} T = 2 \log_2 \hat{w} \\ A = 3\left[(\log_2 \hat{w} - 1) \cdot \hat{w} + 1\right] \end{cases} \tag{6.24}$$

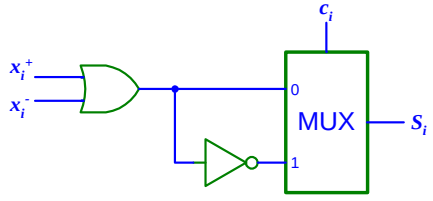
- Circuito de cálculo do resultado da soma - De acordo com a equação 6.16, o valor de cada bit do resultado da soma pode ser determinado utilizando um multiplexador e duas portas lógicas, tendo como sinal de controlo da selecção o bit de *carry* correspondente. Assim, o circuito completo é então constituído por $\hat{w}$ sub-blocos do tipo do apresentado na figura 6.9 e possui as seguintes características:

$$\begin{cases} T = 2 \\ A = 4\hat{w} \end{cases} \tag{6.25}$$

Convém notar que as portas lógicas do tipo OR e INV presentes à entrada do multiplexador não foram consideradas para efeitos de tempo de propagação, pois não se encontram no caminho crítico do circuito.

Assim, é possível determinar as características do circuito completo do somador, com base no modelo descrito na tabela 6.1:

$$\begin{cases} T = (1) + (2 \log_2 \hat{w}) + (2) \\ A = (2\hat{w}) + (3\left[(\log_2 \hat{w} - 1) \cdot \hat{w} + 1\right]) + 4\hat{w} \end{cases} \Leftrightarrow \begin{cases} T = 2 \log_2 \hat{w} + 3 \\ A = 3\hat{w} \log_2 \hat{w} + 3\hat{w} + 3 \end{cases} \tag{6.26}$$



$$\Rightarrow \quad \begin{cases} T = 3 \\ A = 4 \end{cases}$$

Figura 6.9: Circuito de cálculo do resultado da soma.

Conclui-se, assim, que este tipo de somador requer $(2\log_2 \hat{w} + 3)$ níveis de lógica, o que representa uma melhoria significativa em relação a uma topologia tradicional do tipo *ripple-carry*, caracterizada por um total de $3w$ níveis lógicos. Tal como no caso da estrutura do tipo *carry-lookahead*, o aumento da rapidez faz-se à custa de um aumento da área ocupada pelo circuito, pois enquanto que uma topologia do tipo *ripple-carry* requer uma área correspondente a $9w$, esta topologia apresenta uma ocupação da ordem de $3\hat{w}\log_2 \hat{w} + 3\hat{w} + 3$.

Antes de terminar a apresentação desta topologia, convém notar ainda que o facto de o somador apresentado anteriormente não possuir uma entrada externa de *carry* não impossibilita a realização de operações rápidas de subtracção, necessárias para o cálculo do valor absoluto da diferença. De facto, atendendo à equação 6.14a:

$$X = (A - B) = (A - B) + 0 \tag{6.27}$$

Esta equação é em tudo idêntica à equação 6.14c apresentada anteriormente, com excepção da entrada de *carry* do bloco com estrutura em árvore que deve tomar, agora, o valor $c_0 = 0$. Assim, o circuito condicionador de entrada deverá, neste caso, efectuar a seguinte operação:

$$①: \begin{cases} a_i = A_i \\ b_i = B_i \end{cases} \qquad ②: \begin{cases} x^+ = A_i \cdot \overline{B_i} \\ x^- = \overline{A_i} \cdot B_i \end{cases}$$

O circuito de cálculo do vector de *carry* será em tudo semelhante ao apresentado na figura 6.6, com excepção do bloco correspondente aos 4 bits menos significativos, que pode ser obtido através da simplificação do circuito apresentado na figura 6.5(a) para o caso particular de o bit de *carry* tomar o valor $c_0 = 0$. O circuito para o cálculo do resultado final é idêntico ao anteriormente apresentado. Assim, é possível concluir que, utilizando esta topologia, o circuito subtractor tem características idênticas ao circuito somador anteriormente descrito.

6.4 Somador do tipo *prefix-adder*

Uma outra topologia de somador rápido considerada baseia-se em estruturas do tipo *prefix-adder* [**?**] e, em particular, na topologia proposta por Sklansky [33]. Numa estrutura deste tipo, as saídas $(y_{w-1}, y_{w-2}, \ldots, y_0)$ são, em geral,calculadas a partir das

w entradas $(x_{w-1}, x_{w-2}, \ldots, x_0)$ utilizando um operador binário associativo $\star$:

$$y_0 = x_0 \tag{6.28a}$$
$$y_1 = x_1 \star x_0 \tag{6.28b}$$
$$y_2 = x_2 \star x_1 \star x_0 \tag{6.28c}$$
$$\vdots \tag{6.28d}$$
$$y_{w-1} = x_{w-1} \star x_{w-2} \star \cdots \star x_1 \star x_0 \tag{6.28e}$$

ou, representando na forma recursiva:

$$y_0 = x_0 \tag{6.29a}$$
$$y_i = x_i \star y_{i-1} \;\; ; \;\; i = 1, 2, \ldots, w-1 \tag{6.29b}$$

Verifica-se, assim, que numa topologia do tipo *prefix* uma saída depende das entradas de igual ou menor peso, e que as entradas influenciam todas as saídas de igual ou maior peso.

Devido à natureza associativa do operador $\star$, as operações podem ser efectuadas segundo uma ordem qualquer. Em particular, sequências de operações podem ser agrupadas para resolver o problema de uma forma parcial e em paralelo para grupos ou sequências de bits de entrada, resultando em variáveis de grupo do tipo $Y_{i,k}$. Nos níveis superiores, estas sequências de variáveis de grupo podem, então, ser calculadas, dando-se, assim, origem a m níveis intermédios de processamento. A variável $Y_{i,k}^{\ell}$ representa, assim, o resultado da operação utilizando os bits $(x_k, x_{k-1}, \ldots, x_i)$ do nível ℓ. As variáveis de grupo do último nível m terão de cobrir todos os bits de 0 a i $\left(Y_{0,i}^{m}\right)$, representando, assim, o resultado final:

$$Y_{i,i}^{0} = x_i \tag{6.30a}$$
$$Y_{i,k}^{\ell} = Y_{i,j}^{\ell-1} \star Y_{j+1,k}^{\ell-1} \;, \;\; i < j < k \;\; ; \;\; \ell = 1, 2, \ldots, m \tag{6.30b}$$
$$y_i = Y_{0,i}^{m} \;, \;\; i = 0, 1, \ldots, (w-1) \tag{6.30c}$$

Nos últimos anos têm sido propostos vários tipos de arquitecturas para resolver este tipo de problema [**?**], resultando, deste esforço, diferentes soluções em termos de dimensão, área e tempo de processamento.

Uma das primeiras propostas apresentadas foi publicada por Sklansky [33], cuja estrutura se apresenta na figura 6.10. Esta estrutura semi-regular minimiza o tempo de cálculo dos vários bits de saída a partir dos bits de entrada utilizando um modelo em árvore. Todos os sinais intermédios são calculados segundo uma estrutura mínima em árvore, e

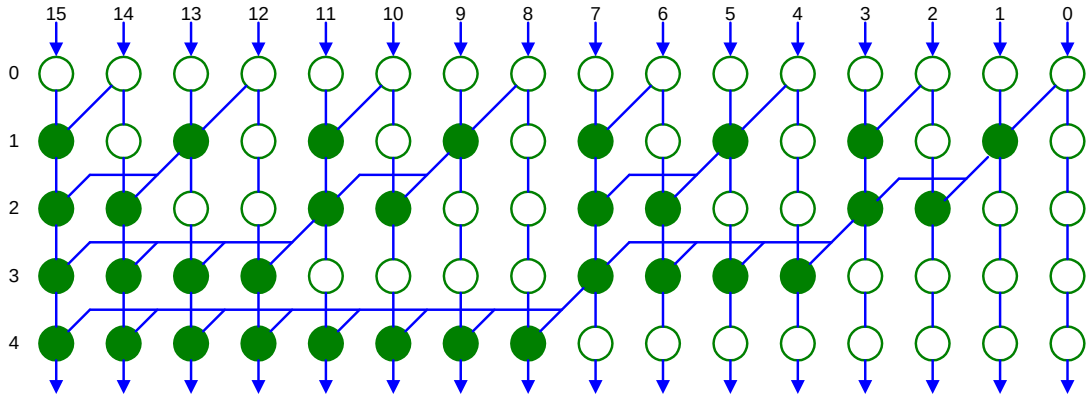


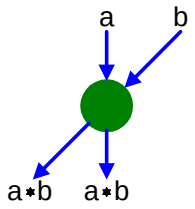
Figura 6.10: Estrutura de um somador do tipo *prefix* proposta por Sklansky [33].

distribuídos, em paralelo, para todos os bits de mais alto nível que requerem o referido sinal. Obtém-se, assim, uma estrutura que proporciona o cálculo rápido devido, sobretudo, à minimização do atraso entre os vários níveis e a um número reduzido de ligações intermédias. Contudo, esta estrutura apresenta algumas desvantagens, nomeadamente, o número de entradas ligadas à saída de um nó cresce consideravelmente com o número de bits (*fan-out*).

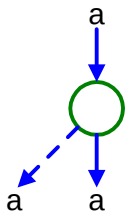
Na figura 6.11 representam-se os dois tipos de nós presentes na estrutura de processamento proposta por Sklansky:

- os nós escuros representam as unidades de cálculo da operação binária e associativa $\star$, tendo como operandos, os sinais a e b (figura 6.11(a)).
- os nós claros representam nós com ligação directa entre a entrada e a saída, não utilizando qualquer tipo de portas lógicas (figura 6.11(b)).

Como se observa na figura 6.10, cada uma das $\hat{w}$ colunas corresponde a uma dada posição de bit. Os nós escuros que operam em paralelo são representados na mesma linha,



(a) Unidade de cálculo da operação $\star$.



(b) Unidade neutra.

Figura 6.11: Nós da estrutura proposta por Sklansky.

enquanto que os nó escuros ligados em série são representados em linhas consecutivas. As saídas de uma dada linha ℓ representam, assim, as variáveis de grupo $Y_{i,k}^{\ell}$. Desta forma, o número total de linhas m ($\propto \log_2 \hat{w}$) corresponderá ao máximo número de níveis de lógica que terão de ser calculados em série, pelo que, o tempo de processamento é da ordem de $\mathcal{O}(\log_2 \hat{w})$.

A estrutura do tipo *prefix* utiliza, tal como a estrutura do tipo *carry-lookahead* descrita anteriormente, sinais intermédios do tipo *generate* e *propagate*. Estes sinais podem apresentar três significados distintos:

- Geração do sinal de *carry* a '0' (ou anulação do sinal de *carry* a '1');
- Geração do sinal de *carry* a '1';
- Propagação do sinal de *carry*.

Como consequência, estes sinais terão de ser codificados utilizando dois bits. Tal como para a estrutura do tipo *carry-lookahead*, utilizam-se as designações de "grupo de geração" $\left(G_{i,k}^{\ell}\right)$ e de "grupo de propagação" $\left(P_{i,k}^{\ell}\right)$. Estes sinais são utilizados para gerar o sinal composto de geração/propagação $Y_{i,k}^{\ell+1} = \left(G_{i,k}^{\ell+1}, P_{i,k}^{\ell+1}\right)$ no nível $\ell+1$. Os sinais do primeiro nível $\left(G_{i,i}^{0}, P_{i,i}^{0}\right)$ corresponderão, assim, ao bit de geração g_i e de propagação p_i, e são calculados a partir dos operandos num nível de pré-processamento denominado, a seguir, pela operação $\square$.

Assim, de acordo com as equações 6.7 e 6.8, os pares de sinais correspondentes ao nível ℓ serão calculados a partir dos sinais obtidos no nível $(\ell-1)$ utilizando a seguinte expressão:

$$\left(G_{i,k}^{\ell}, P_{i,k}^{\ell}\right) = \left(G_{j,k}^{\ell-1}, P_{j,k}^{\ell-1}\right) \star \left(G_{i,j}^{\ell-1}, P_{i,j}^{\ell-1}\right) \tag{6.31a}$$

$$= \left(G_{j,k}^{\ell-1} + P_{j,k}^{\ell-1} G_{i,j}^{\ell-1}, P_{j,k}^{\ell-1} P_{i,j}^{\ell-1}\right) \tag{6.31b}$$

Cada um dos bits de *carry* (c_i) é calculado a partir dos sinais obtidos no último nível $\left(G_{0,i}^{m}, P_{0,i}^{m}\right)$. Os bits de soma são obtidos a partir de uma operação de pós-processamento representada, a seguir, pelo símbolo $\diamond$. Para implementar um somador com entrada de *carry* (c_{in}), utiliza-se lógica adicional, conforme se descreverá a seguir. Partindo das expressões anteriores, é possível chegar às seguintes expressões:

$$g_i = a_i \cdot b_i \tag{6.32}$$

$$p_i = a_i \oplus b_i \ , \quad i = 0, 1, \ldots, (w-1) \tag{6.33}$$

$$\left(G^0_{i,i}, P^0_{i,i}\right) = (g_i, p_i) \tag{6.34}$$

$$\left(G^\ell_{i,k}, P^\ell_{i,k}\right) = \left(G^{\ell-1}_{j,k} + P^{\ell-1}_{j,k} G^{\ell-1}_{i,j}, P^{\ell-1}_{j,k} P^{\ell-1}_{i,j}\right), \quad i \leq j \leq k\ , \quad \ell = 1, 2, \ldots, m \tag{6.35}$$

$$c_{i+1} = G^m_{0,i} + P^m_{0,i} \cdot c_{in}\ , \quad i = 0, 1, \ldots, (w-1) \tag{6.36}$$

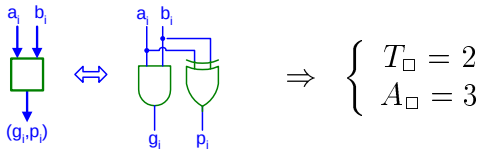
$$s_i = p_i \oplus c_i\ , \quad i = 0, 1, \ldots, (w-1)\ \ e\ \ c_0 = c_{in} \tag{6.37}$$

$$c_{out} = c_w \tag{6.38}$$

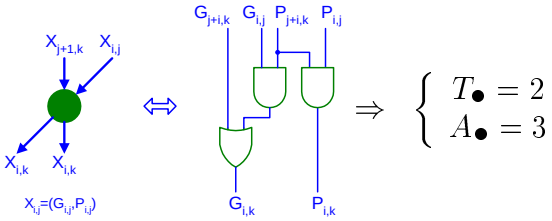
Cada um destes três conjuntos de expressões é calculado em cada um dos blocos apresentados na figura 6.12. Nesta figura apresentam-se, também, medidas de custo em termos de rapidez e de recursos de *hardware* requeridos, de acordo com o modelo apresentado na tabela 6.1.

Convém notar que os sinais de propagação p_i, calculados no nível de pré-processamento, terão de ser propagados pela estrutura por forma a calcular o sinal correspondente à soma s_i. Para além disso, devido à estrutura em árvore binária utilizada, torna-se necessário efectuar a extensão do número de bits dos operandos sempre que w não é uma potência inteira de 2. Consequentemente, nas expressões apresentadas deve ser utilizado o valor $\hat{w}$ sempre que se refere o número de bits dos operandos.

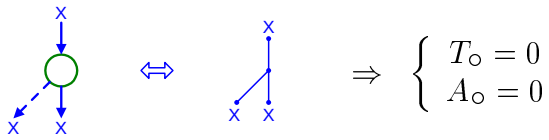
Uma das soluções possíveis para resolver o problema da implementação de um somador com entrada de *carry* consiste na utilização de um nível adicional de operadores



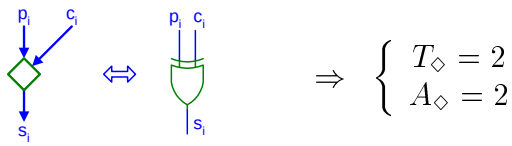
(a) Bloco de pré-processamento.



(b) Unidade de cálculo da estrutura *prefix*.



(c) Unidade neutra da estrutura *prefix*.



(d) Bloco de pós-processamento.

Figura 6.12: Estrutura interna dos blocos constituintes da estrutura de um somador proposta por Sklansky.

do tipo ●, tal como se ilustra na figura 6.13. O bit de *carry* entrará na estrutura apenas neste nível adicional. Esta característica permite, assim, o processamento de uma forma bastante rápida do sinal de *carry* de entrada do somador, sendo por isso designado de "somador Sklansky rápido". Consequentemente, esta estrutura apresenta características particularmente interessantes para a implementação de incrementadores em que o caminho crítico de circuito se encontra entre a entrada c_{in} e a saída c_{out}.

Assim, de acordo com o modelo utilizado, o tempo de cálculo da soma é dado por:

$$T_s^{a,b} = 1 \times T_{\square} + (\log_2 \hat{w} + 1) \times T_{\bullet} + 1 \times T_{\diamond} \tag{6.39a}$$
$$= 2 + 2\,(\log_2 \hat{w} + 1) + 2 \tag{6.39b}$$
$$= 2 \log_2 \hat{w} + 6 \tag{6.39c}$$

Partindo do princípio de que o processamento dos operandos da soma não se encontra incluído no caminho crítico, verifica-se que o tempo de cálculo do resultado da soma depende, apenas, do bit de entrada c_{in}, sendo dado por:

$$T_s^{c_{in}} = 1 \times T_{\bullet} + 1 \times T_{\diamond} \tag{6.40a}$$
$$= 2 + 2 \tag{6.40b}$$
$$= 4 \tag{6.40c}$$

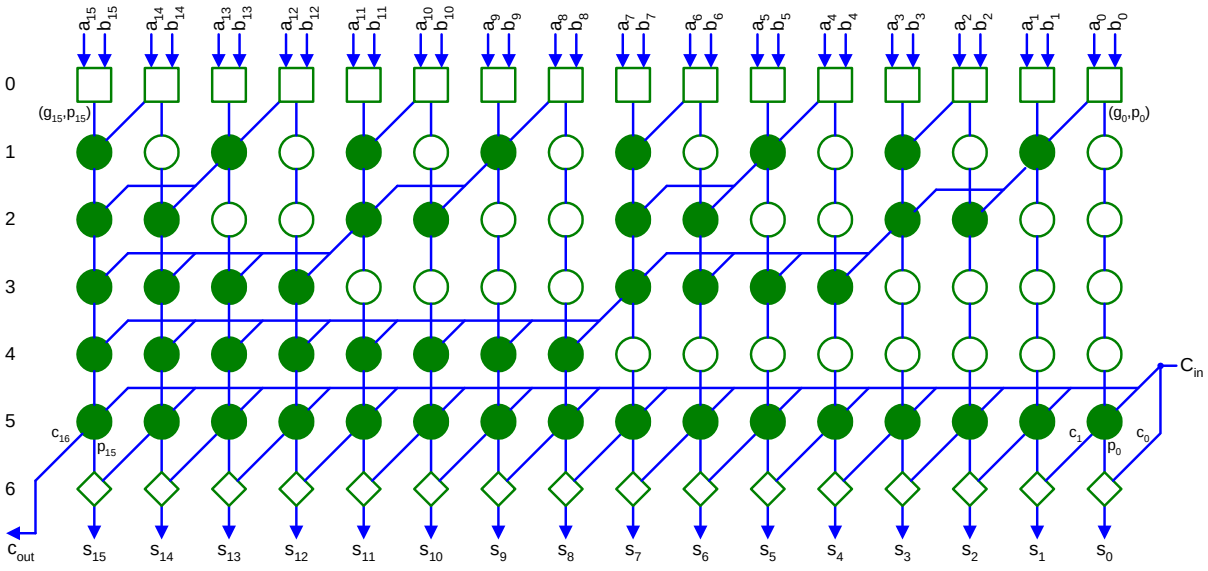


Figura 6.13: Somador Sklansky rápido.

A área do circuito é:

$$A = \hat{w} \times A_{\square} + \left(\frac{1}{2}\hat{w}\log_2 \hat{w} + \hat{w}\right) \times A_{\bullet} + \hat{w} \times A_{\diamond} \tag{6.41a}$$

$$= 3\hat{w} + 3\left(\frac{1}{2}\hat{w}\log_2 \hat{w} + \hat{w}\right) + 2\hat{w} \tag{6.41b}$$

$$= \frac{3}{2}\hat{w}\log_2 \hat{w} + 8\hat{w} \tag{6.41c}$$

Uma solução alternativa para o processamento do sinal de *carry* de entrada consiste na utilização de um elemento processador do tipo $\square$ especialmente adaptado para o processamento do bit menos significativo dos operandos e do sinal de *carry* de entrada, conforme se descreve na equação 6.42a. Esta solução permite, assim, manter toda a estrutura original apresentada na figura 6.10 inalterada[5] (ver figuras 6.14 e 6.15). Contudo, embora esta solução apresente vantagens em termos de área, ela tem associado um tempo de processamento do sinal de entrada de *carry* bastante maior, pois agora o caminho crítico atravessa toda a estrutura do somador. Por esta razão, esta solução é designada por "somador Sklansky lento".

$$g_0 = a_0 b_0 + a_0 c_{in} + b_0 c_{in} \tag{6.42a}$$

$$g_i = a_i b_i \quad , \quad i = 1, 2, \ldots, (\hat{w} - 1) \tag{6.42b}$$

$$\vdots$$

$$c_{i+1} = G^m_{i,0} \quad , \quad i = 0, 1, \ldots, (\hat{w} - 1) \tag{6.42c}$$

Assim, em termos de tempo de processamento, este circuito apresenta as seguintes características:

$$T_s^{a,b} = T_s^{c_{in}} = 1 \times T_{\square} + \log_2 \hat{w} \times T_{\bullet} + 1 \times T_{\diamond} \tag{6.43a}$$

$$= 3 + 2\log_2 \hat{w} + 2 \tag{6.43b}$$

$$= 2\log_2 \hat{w} + 5 \tag{6.43c}$$

A área requerida por esta estrutura é:

$$A = (\hat{w} - 1) \times A_{\square} + 1 \times A^*_{\square} + \left(\frac{1}{2}\hat{w}\log_2 \hat{w}\right) \times A_{\bullet} + \hat{w} \times A_{\diamond} \tag{6.44a}$$

$$= 2(\hat{w} - 1) + 7 + 3\left(\frac{1}{2}\hat{w}\log_2 \hat{w}\right) + 2\hat{w} \tag{6.44b}$$

$$= \frac{3}{2}\hat{w}\log_2 \hat{w} + 4\hat{w} + 5 \tag{6.44c}$$

[5]Neste caso deixa de ser necessária a utilização de um nível adicional de processamento.

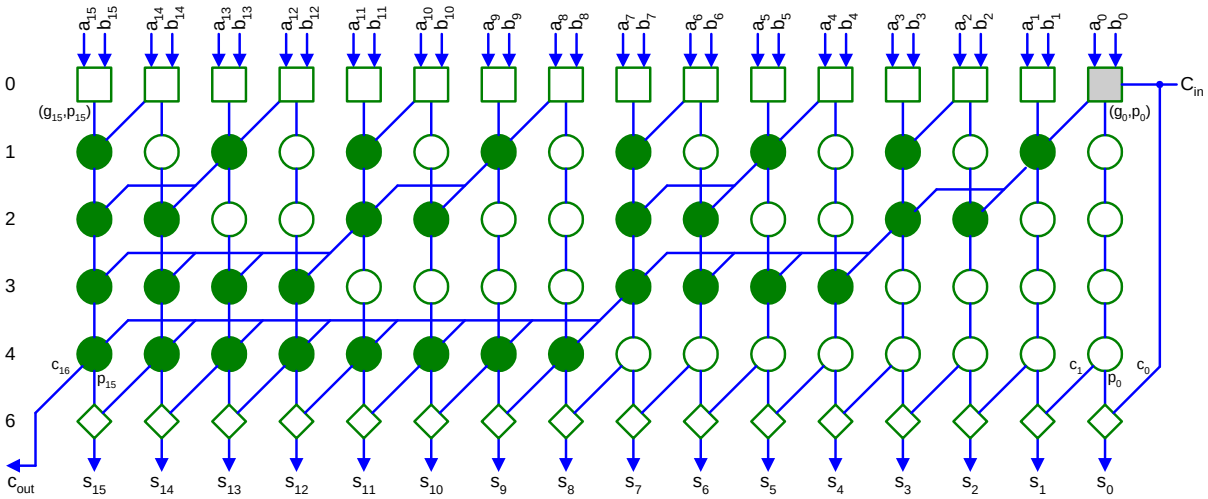


Figura 6.14: Somador Sklansky lento.

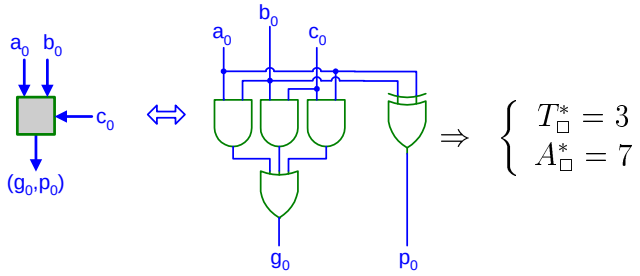


Figura 6.15: Elemento processador para o pré-processamento dos bits menos significativos dos operandos e do sinal de *carry* de entrada.

Tal como as estruturas anteriores, este tipo de somador apresenta um tempo de processamento bastante inferior ao dos somadores do tipo *ripple-carry* ($3w$ níveis de lógica). Contudo, verifica-se, uma vez mais, que o aumento da rapidez se reflecte ao nível da área ocupada pelo circuito, sendo agora da ordem de $\frac{3}{2}\hat{w}\log_2\hat{w} + 8\hat{w}$, em contraste com $9w$ requerido pela topologia do tipo *ripple-carry*.

6.5 Comparação das diferentes estruturas para soma

Na tabela 6.2 registaram-se os resultados obtidos para as estruturas de somadores analisadas, no que diz respeito ao *tempo* de processamento e à *área* requerida por cada circuito. Com base nestes valores, apresenta-se graficamente nas figuras 6.16, 6.17 e 6.18 o tempo de processamento (T), a área ocupada (A) e o produto área-tempo de processamento (AT) dos circuitos somadores em função do número de bits dos operandos (w). Em cada um destes gráficos representou-se, com uma área sombreada, a zona de interesse para o caso em estudo ($w = 8$ bits).

Estrutura	Tempo	Área
Ripple-Carry	$3w$	$9w$
Carry-Lookahead	$4\log_2 \hat{w} + 3$	$11\hat{w}$
Baseada em aritmética redundante	$2\log_2 \hat{w} + 3$	$3\hat{w}\log_2 \hat{w} + 3\hat{w} + 3$
"Sklansky rápido"(Incrementador)[6]	4	$\frac{3}{2}\hat{w}\log_2 \hat{w} + 8\hat{w}$
"Sklansky lento"	$2\log_2 \hat{w} + 5$	$\frac{3}{2}\hat{w}\log_2 \hat{w} + 4\hat{w} + 5$

Tabela 6.2: Comparação das estruturas de soma, com base no modelo da tabela 6.1.

O gráfico representativo da variação do tempo de processamento permite, de uma forma clara, verificar as diferenças em termos de desempenho das topologias rápidas estudadas, em contraste com a topologia mais simples do tipo *ripple-carry*. Esta diferença será tanto mais acentuada quanto maior o número de bits dos operandos. Verifica-se, ainda, que para além da estrutura "Sklansky rápido", que apresenta o melhor desempenho sob certas condições particulares, as estruturas baseadas em aritmética redundante e a "Sklansky lento" apresentam tempos de processamento relativamente baixos.

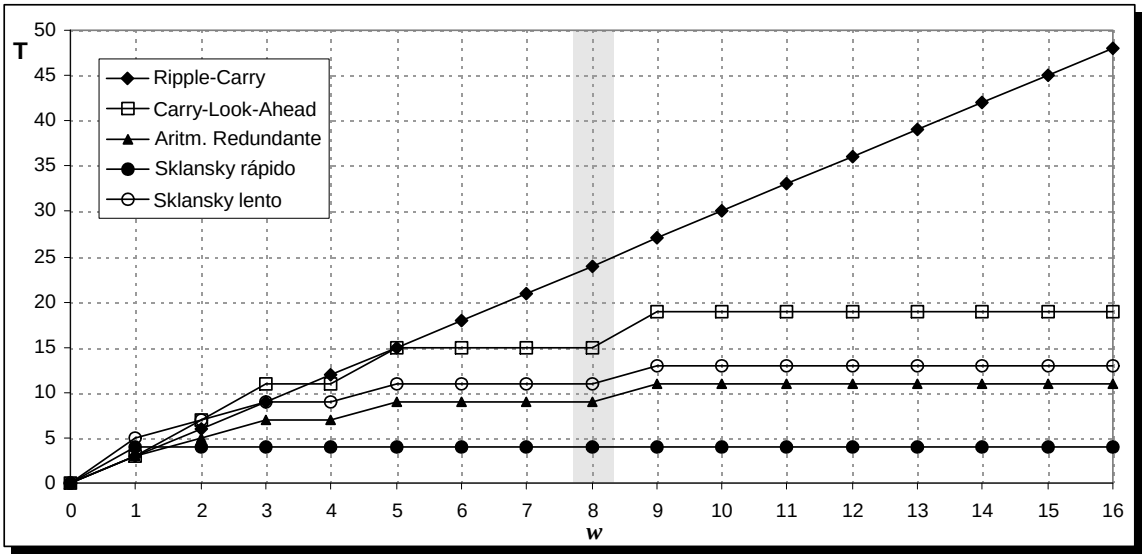


Figura 6.16: Comparação do tempo de processamento das várias estruturas de soma.

No que se refere à área do circuito, verifica-se que a estrutura do tipo *ripple-carry* é, tal como se esperava, a que requer menor área. Como consequência de utilizarem estruturas baseadas em árvores binárias, todas as outras apresentam uma variação em "degraus" entre potências inteiras de 2. Estes tipos de somadores rápidos apresentam

[6]Valores para o caso em que o processamento dos operandos da soma não se encontra incluído no caminho crítico, o que pode ser útil numa operação final de incremento.

áreas de circuito que se aproximam da área requerida pela estrutura do tipo *ripple-carry* sempre que a largura dos operandos é uma potência inteira de 2, pois nessas situações os circuitos lógicos são utilizados com a máxima eficiência. Em qualquer dos casos, verifica-se que a estrutura baseada no uso de aritmética redundante é a que apresenta maior área de circuito.

A figura 6.18 permite verificar que, no que respeita à variação do produto área-tempo de processamento, as estruturas para cálculo rápido apresentam vantagens significativas para operandos com um número de bits superior a 11.

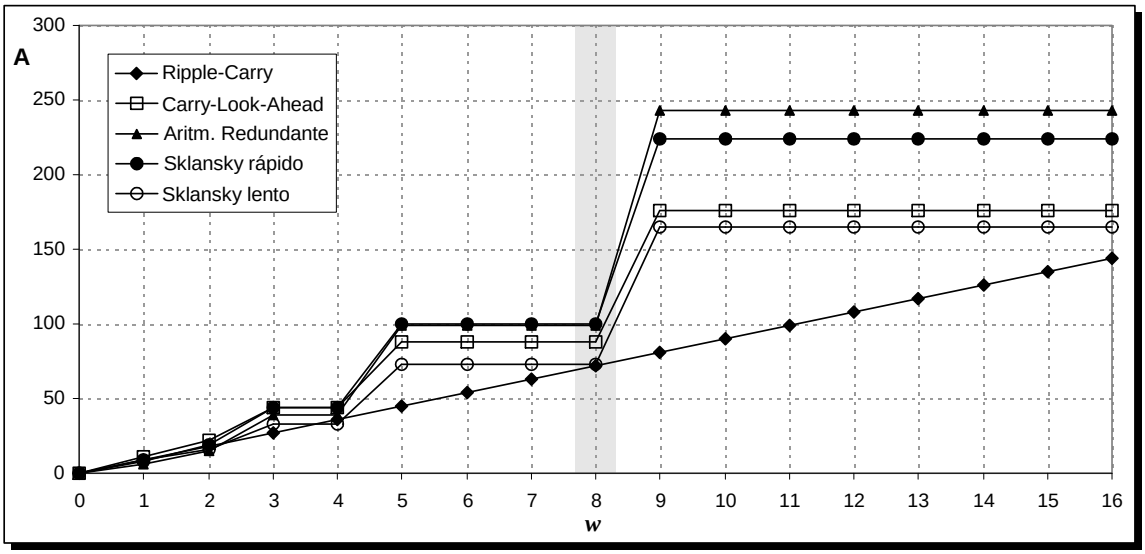


Figura 6.17: Comparação da área requerida pelas várias estruturas de soma.

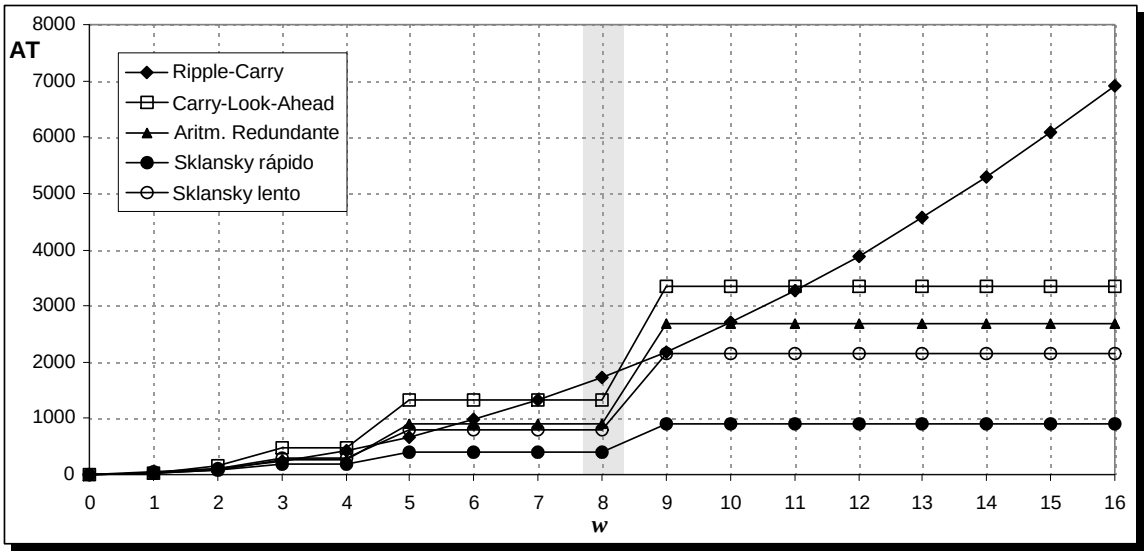


Figura 6.18: Comparação do produto área - tempo de processamento das várias estruturas de soma.

Capítulo 7

Implementação de um processador de procura exaustiva

Conteúdo

7.1 Introdução

No capítulo 5 foi proposta uma arquitectura para estimação de movimento com pesquisa exaustiva, tendo-se procurado obter processadores rápidos utilizando, apenas, os recursos de *hardware* indispensáveis.

Contudo, para além do algoritmo de pesquisa propriamente dito, do esquema de processamento utilizado e das várias melhorias introduzidas ao nível da utilização dos recursos, existem, ainda, outros factores relacionados com a implementação do processador que são fundamentais para a obtenção de processadores eficientes. São exemplos destes factores:

- as arquitecturas das unidades aritméticas e lógicas que integram as estruturas do processador;
- a forma como as estruturas são decompostas em elementos mais básicos, em particular, em portas ou células lógicas;
- a tecnologia seleccionada para implementar o processador.

Tal como se referiu na secção 5.3 para o caso de um processador com um único bloco activo na matriz de processamento ($NC = 1$), o processador agora proposto pode ser dividido em cinco grandes blocos presentes na figura 7.1, que são:

- os registos de entrada,
- a matriz de processamento,
- o somador em árvore,
- o bloco de comparação,
- a unidade de controlo.

Para o projecto e implementação destes blocos, consideraram-se os seguintes objectivos a cumprir:

- facilitar a integração do processador em sistemas de codificação de vídeo usuais, através da simplificação da interface de I/O e minimizando o número de sinais a trocar com o exterior; procurou-se, também, simplificar a interligação do processador a sistemas usuais de bancos de memória e a outras unidades de processamento que integram um sistema de codificação de vídeo;
- maximizar a frequência de operação do processador, recorrendo a técnicas de *pipelining* e de sistolização, com vista a minimizar o caminho crítico e a unidades lógicas e aritméticas rápidas;

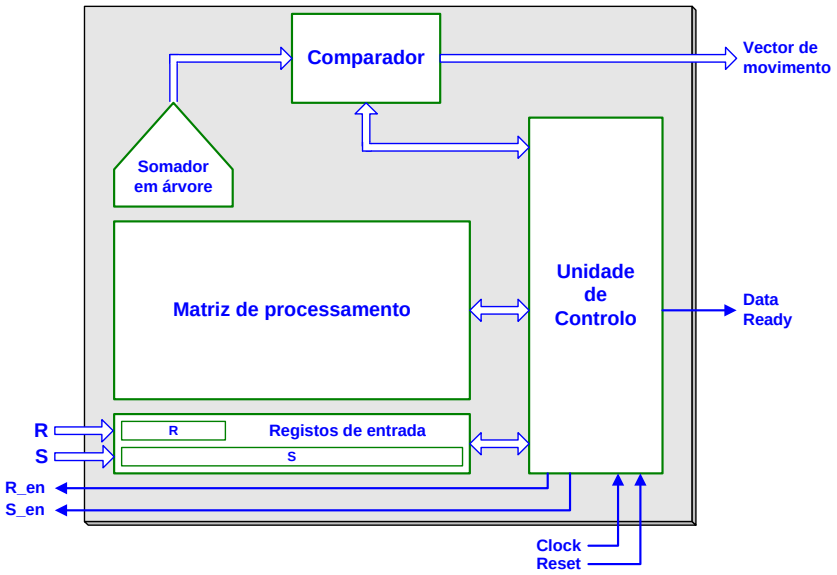


Figura 7.1: Diagrama de blocos do processador implementado utilizando um único bloco activo ($NC = 1$).

- minimizar a quantidade de recursos de *hardware* requerida, atendendo sempre ao compromisso entre área ocupada (A) e tempo de processamento (T);

- realizar uma descrição das estruturas de *hardware* totalmente parametrizável, tornando assim possível obter facilmente topologias de processamento diferentes através do ajuste directo de um número mínimo de parâmetros da arquitectura: N, p, h e NC.

Para a descrição do processador utilizou-se a linguagem de descrição de circuitos digitais VHDL [38, 39, 40, 41], pelas suas potencialidades e por ser uma das mais utilizadas hoje em dia. Para além disso, adoptou-se, de forma exaustiva, um tipo de descrição completamente estrutural das arquitecturas, para melhor atingir os objectivos atrás enunciados. Para o desenvolvimento do processador utilizou-se um conjunto de ferramentas de síntese e simulação de circuitos da Synopsys^{TM} [42, 43, 44].

Nas secções seguintes passar-se-á a descrever as estruturas principais do processador, nomeadamente, os elementos processadores, a matriz de processamento, o somador em árvore, o bloco de comparação e os vários circuitos para controlar o funcionamento do processador. Apresentam-se, também, as técnicas adoptadas para a implementação das várias estruturas apontando, sempre que necessário e relevante, as várias soluções equacionadas e justificando as opções que foram seguidas.

7.2 Elementos processadores

De acordo com a descrição efectuada na secção 5.3.1, existem dois tipos de elementos processadores:

- **Elemento processador activo**, cuja função principal é a de realizar o cálculo do valor absoluto da diferença entre cada pixel do macrobloco de referência e o pixel correspondente do macrobloco candidato. Realiza, ainda, a acumulação dos valores parciais da medida de similaridade do macrobloco candidato em consideração e o transporte e deslocamento dos pixels respectivos da área de pesquisa, permitindo, assim, a passagem dos vários macroblocos candidatos pelo bloco activo do processador.
- **Elemento processador passivo**, cuja única função é a de deslocar os pixels da área de pesquisa, transferindo-os para os blocos activos do processador.

De seguida, passar-se-á a descrever cada um dos elementos processadores implementados.

7.2.1 Elemento processador activo

Tal como se referiu anteriormente, este tipo de elementos processadores cumpre três tarefas principais, correspondentes ao:

- transporte e deslocamento dos pixels do macrobloco de referência e da área de pesquisa;
- cálculo do valor absoluto da diferença entre o pixel do macrobloco de referência e o pixel da área de pesquisa correspondente ao macrobloco candidato em consideração;
- acumulação do valor absoluto da diferença com o valor parcial da medida de similaridade calculado noutros elementos processadores activos.

Para realizar estas tarefas, decompõe-se a estrutura do elemento processador em três circuitos principais, correspondentes a cada uma das tarefas referidas, apresentados na figura 7.2:

- circuito de registos de transporte;
- circuito de cálculo do valor absoluto da diferença;
- circuito de cálculo da acumulação dos valores absolutos.

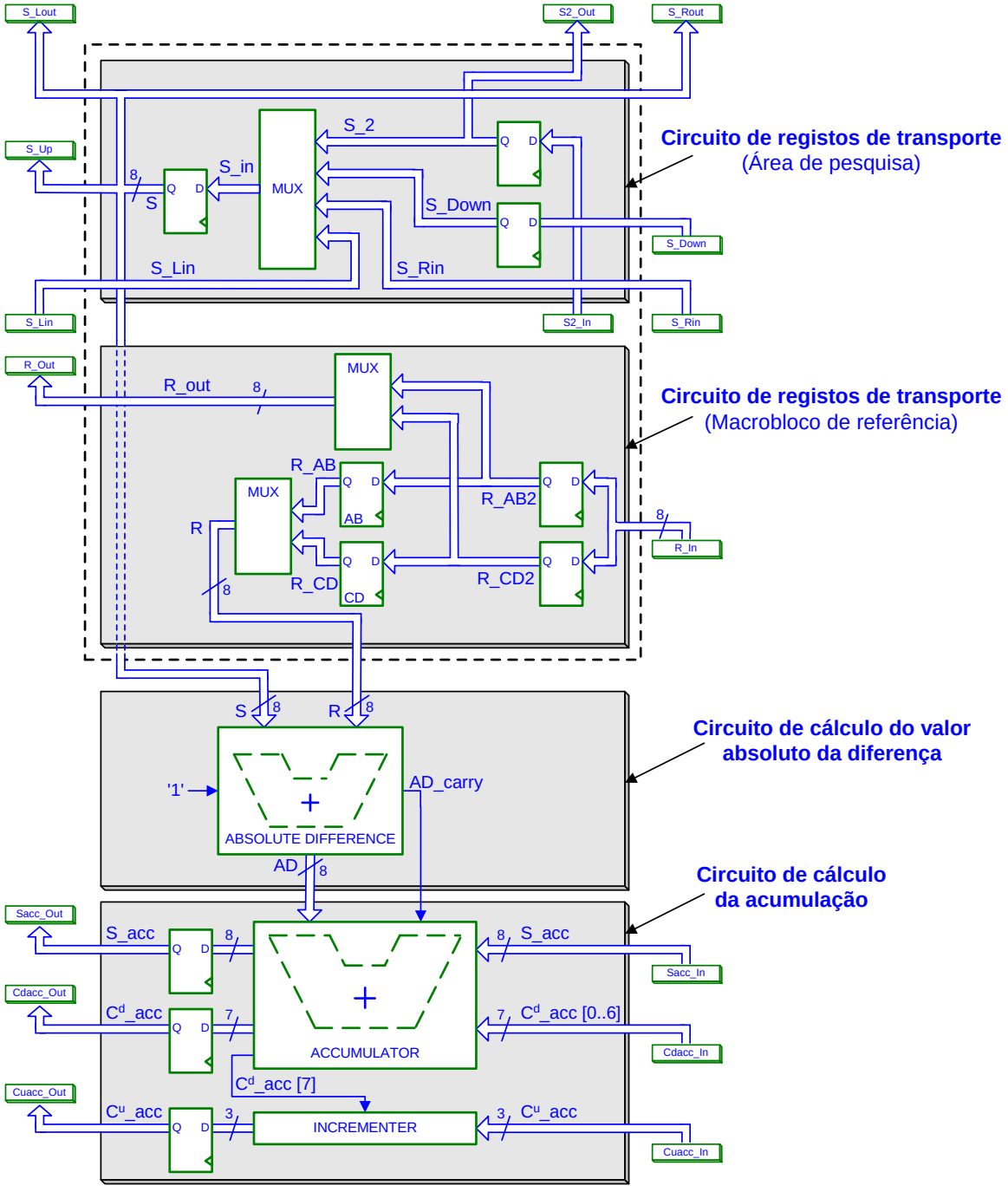


Figura 7.2: Estrutura interna do elemento processador activo.

Registos de transporte

O circuito correspondente aos registos de transporte pode ser decomposto em dois sub-circuitos independentes, para transporte dos pixels da área de pesquisa e para transporte dos pixels do macrobloco de referência.

Da descrição do esquema de funcionamento apresentado na figura 5.12, verifica-se que os dados relativos a cada pixel da área de pesquisa podem ser provenientes de três entradas principais, correspondentes ao elemento processador situado à esquerda, à direita e a baixo do elemento processador considerado (entradas S_Lin, S_Rin e S_Down, respectivamente). Contudo, face à descrição do esquema de transferência e transporte de pixels de dupla camada (figura 5.13), há ainda a considerar a entrada correspondente ao nível de transporte em modo transparente (entrada S2). O valor do pixel do macrobloco candidato é seleccionado através de um multiplexador de 4 entradas, sendo depois armazenado num registo cuja saída será utilizada para o cálculo do valor absoluto da diferença. O valor armazenado neste registo será, também, entregue aos elementos processadores localizados à esquerda, à direita e a cima do elemento processador considerado (saídas S_Lout, S_Rout e S_Up, respectivamente).

O registo associado à entrada S_Down é utilizado para garantir o funcionamento sistólico do circuito, aquando da mudança da linha de macroblocos candidatos sob processamento. O registo presente entre a entrada S2_in e a saída S2_out realiza a função desempenhada pelo nível de transporte em modo transparente.

O funcionamento do circuito de transporte dos pixels do macrobloco de referência é diferente ao anteriormente descrito. De facto, para obter topologias de processamento do tipo B (ver secção 5.4.3), é necessário ter armazenado, em qualquer instante, os valores do par de pixels correspondentes a cada uma das secções AB e CD (ver figura 5.19(a)): um pixel do bloco A e outro do bloco C ou um pixel do bloco B e outro do bloco D. Torna-se assim evidente a necessidade de utilizar dois registos distintos para armazenar este par de pixels: Registo R_AB e registo R_CD. O valor utilizado para o cálculo da medida de similaridade será então seleccionado através de um multiplexador, consoante o sentido de deslocamento dos pixels da área de pesquisa.

Da mesma forma, também o nível de transporte em modo transparente terá de ser constituído por dois registos independentes (R_AB2 e R_CD2). Contudo, o pré-carregamento dos pixels do macrobloco de referência em cada elemento processador requer, neste caso, o dobro do tempo requerido para o caso da topologia do tipo A, pois ter-se-ão de transferir os dados correspondentes aos dois pixels a armazenar em cada elemento processador. Esta operação é feita entre os vários elementos processadores de uma forma sequencial, transferindo os dados por intermédio de um multiplexador que selecciona qual dos valores presentes nestes registos, correspondentes ao nível de transporte em modo transparente, será colocado no porto de saída de cada elemento processador.

Circuito de cálculo do valor absoluto da diferença

O processador de estimação de movimento proposto baseia-se na medida de similaridade SAD, correspondente à soma dos valores absolutos das diferenças entre os pixels do macrobloco de referência e os pixels dos macroblocos candidatos, de acordo com a equação 7.1:

$$d(c,l) = \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} |R(u,v) - S(c+u, l+v)| \tag{7.1}$$

A operação elementar desta equação, o cálculo da soma do valor absoluto da diferença, pode ser decomposta em três operações elementares:

$$\textcircled{1}:\ X_t = R_t - S_t \qquad \textcircled{2}:\ Y_t = \begin{cases} X_t\,, & X_t \geq 0 \\ -X_t\,, & X_t < 0 \end{cases} \qquad \textcircled{3}:\ Z_t = Z_{t-1} + Y_t$$

Considerando uma representação binária em complemento para dois, onde o bit mais significativo (MSB[1]) representa o bit de sinal, a sequência anterior pode ser representada da seguinte forma, em que se designou por $\overline{S_t}$ a representação em complemento para um de S_t e porX^{Cout} o valor do bit de *carry* da saída da operação ①:

$$\textcircled{1}:\ X_t = R_t + \overline{S_t} + 1 \qquad \textcircled{2}:\ Y_t = \begin{cases} X_t\,, & X^{Cout} = 1 \\ \overline{X_t} + 1\,, & X^{Cout} = 0 \end{cases} \qquad \textcircled{3}:\ Z_t = Z_{t-1} + Y_t\,,$$

Verifica-se, assim, a necessidade de realizar três operações de adição para efectuar a acumulação do valor absoluto da diferença, sendo necessários, por isso, três somadores.

Contudo, através de um estudo mais atento da sequência de operações descrita é possível efectuar o cálculo desta operação recorrendo a circuitos mais simples. De facto, se o somador utilizado na fase de acumulação possuir uma entrada de *carry*, é possível evitar o somador correspondente à segunda etapa da sequência de operações anterior:

$$\textcircled{1}:\ X_t = R_t + \overline{S_t} + 1 \qquad \textcircled{2}:\ \begin{cases} Y_t = \begin{cases} X_t\,, & X^{Cout} = 1 \\ \overline{X_t}\,, & X^{Cout} = 0 \end{cases} \\ \\ c_t = \begin{cases} 0\,, & X^{Cout} = 1 \\ 1\,, & X^{Cout} = 0 \end{cases} \end{cases} \qquad \textcircled{3}:\ Z_t = Z_{t-1} + Y_t + c_t$$

Assim, o circuito de cálculo do valor absoluto da diferença tomará a forma apresentada na figura 7.3. Neste circuito optou-se por implementar o bloco correspondente à segunda etapa da sequência de operações anterior com um conjunto de 8 portas lógicas do tipo

[1]Do Inglês *Most Significant Bit*.

XOR, efectuando-se (ou não) a negação de todos os bits resultantes da etapa anterior, consoante o valor do sinal do bit de *carry*. O resultado desta operação será, então, entregue ao circuito de acumulação.

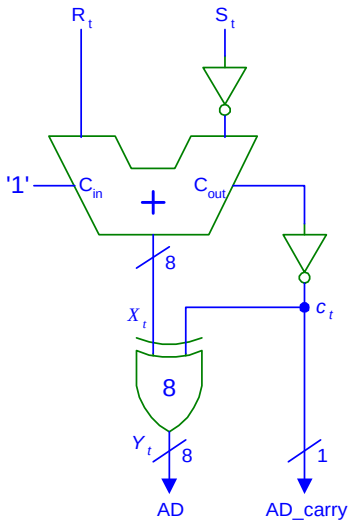


Figura 7.3: Circuito de cálculo do valor absoluto da diferença

Circuito de acumulação

O circuito de acumulação, presente em cada elemento processador activo, é responsável pela soma dos resultados parciais da medida de similaridade do macrobloco candidato em consideração. Este cálculo é realizado em paralelo e de uma forma distribuída pelas ℓ colunas de elementos processadores que constituem cada bloco activo (ver figura 5.16). Obtém-se assim, como resultado final, um conjunto de ℓ valores parciais que serão somados entre si no somador em árvore.

Tal como se pode verificar a partir do esquema apresentado na figura 7.2, este circuito efectua a soma entre o valor parcial recebido do elemento processador localizado na linha inferior e o valor absoluto da diferença calculado entre o pixel do macrobloco candidato e o pixel do macrobloco de referência. Constata-se, assim, que os dois somadores estabelecem o caminho crítico do circuito, pelo que se optou por reduzir o seu valor através da implementação da etapa de acumulação utilizando somadores rápidos do tipo *carry-free*. Assim, os somadores utilizados são do tipo *carry-save-adder* [**?** 28] que permite, através de uma representação redundante, reduzir o tempo de cálculo em relação aos somadores com propagação do sinal de *carry* (*carry-propagate*) [37**?**].

Na figura 7.4 apresentam-se os operandos que são entregues a este circuito de acumulação, que são: um vector de 8 bits correspondente ao valor absoluto da diferença calculado na fase anterior (vector AD[7..0]), um vector de bits de soma obtido na etapa de acumulação do elemento processador anterior (vector S_acc[7..0]) e um vector de bits de *carry*, também calculado nesse elemento processador (vector C_acc[8..1]).

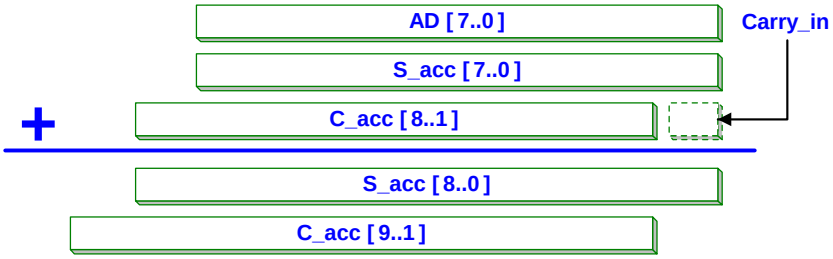


Figura 7.4: Acumulação dos resultados parciais da medida de similaridade, utilizando uma topologia do tipo *carry-save*.

Considerando o cálculo do valor absoluto da diferença utilizando uma representação com 8 bits, e admitindo a implementação do bloco activo do processador utilizando h linhas de elementos processadores (ver figura 5.16), no pior caso, o valor entregue pelo último elemento processador de cada coluna do bloco activo ao somador em árvore é no máximo:

$$Z_h = h \times 2^8 \tag{7.2}$$

Assim, é necessário utilizar $\log_2 h + 8$ bits de resolução para representar cada um dos resultados parciais a acumular no somador em árvore. Contudo, ao efectuar o dimensionamento deste circuito houve, ainda, a necessidade de tomar em consideração a seguinte observação relativamente à topologia do somador do tipo *carry-save*: o vector de *carry* do resultado obtido em cada nível da acumulação apresenta-se deslocado de uma posição para a esquerda relativamente ao vector de entrada correspondente, pelo que ambos os vectores do resultado desta operação requerem um bit adicional, relativamente aos vectores dos operandos de entrada correspondentes (ver figura 7.4).

Este facto induziria, de uma forma natural, a optar por uma das seguintes hipóteses:

a) Para garantir a regularidade do circuito, considera-se cada elemento processador com os vectores correspondentes ao resultado dimensionado para o pior caso, isto é, suportando $(\log_2 h + 8) + h$ bits. Os h bits adicionais serviriam, assim, para assegurar o crescimento dos vectores de *carry* entre cada um dos h níveis de acumulação.

b) Para reduzir os recursos de *hardware* requeridos, efectuar o dimensionamento do vector de *carry* de cada elemento processador dependendo da linha na qual este será colocado: $\#bits_{carry} = 8 + \log_2 h_i$, sendo h_i o número da linha correspondente ao elemento processador considerado. Conseguir-se-ia, assim, garantir a presença do número de bits requeridos em cada um dos h níveis do bloco activo, optimizando, também, os recursos de *hardware* utilizados para acumular os resultados parciais. Esta opção sacrificaria, no entanto, as características de regularidade do circuito, pois passaria a haver h arquitecturas diferentes de elementos processadores activos apresentando, cada uma delas, um tempo de propagação diferente.

A opção tomada foi num sentido um pouco diferente, evitando, assim, as desvantagens associadas às duas alternativas acima referidas. Assim, utiliza-se uma única arquitectura para o elemento processador activo, constituída por um somador do tipo *carry-save* de 8 bits e por um incrementador de $\log_2 h$ bits, conforme se ilustra na figura 7.5.

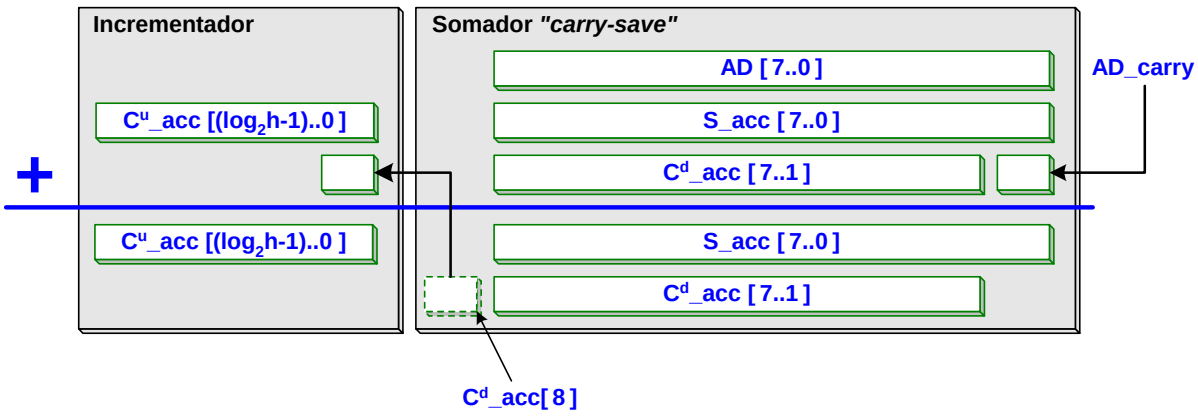


Figura 7.5: Acumulação dos resultados parciais da medida de similaridade, utilizando um somador do tipo *carry-save* e um incrementador.

De facto, ligando o bit mais significativo do vector de *carry* obtido à saída do somador do tipo *carry-save* à entrada do incrementador, obtém-se uma arquitectura híbrida para o somador pretendido: os 8 bits menos significativos são calculados de uma forma expedita utilizando a topologia do tipo *carry-save*, enquanto que os $\log_2 h$ bits mais significativos são, agora, calculados utilizando um somador com arquitectura *carry-propagate*. A entrada de *carry* do incrementador serve, assim, para controlar a operação de incrementação do resultado obtido a partir do elemento processador da linha anterior. Desta forma, consegue-se manter constante o número de bits necessário à representação dos valores correspondentes às somas parciais da medida de similaridade, não aumentando os recursos de *hardware* requeridos.

Assim, a estrutura dos vectores correspondentes às somas parciais do bloco de acumulação será constituída por três vectores distintos (ver figura 7.5):

- $\mathsf{S\_acc}\,[7\,..\,0]$ - **Vector de soma** - constituído pelos 8 bits do vector soma resultantes da acumulação dos valores parciais da medida de similaridade utilizando o somador do tipo *carry-save*;
- $\mathsf{C^d\_acc}\,[7\,..\,1]$ - **Vector de *carry* menos significativo** - constituído pelos 7 bits menos significativos do vector de *carry* obtido à saída do somador do tipo *carry-save*;
- $\mathsf{C^u\_acc}\,[(\log_2 \mathsf{h} - 1)\,..\,0]$ - **Vector de *carry* mais significativo** - constituído pelos $\log_2 h$ bits obtidos à saída do incrementador; este valor deverá ser incrementado sempre que se verifique a condição $\mathsf{C^d\_acc}\,[8] = 1$.

Na secção anterior foi referido que a acumulação dos valores absolutos das diferenças pode ser facilmente realizada utilizando, apenas, dois somadores. A única condição para atingir este objectivo é a de que o somador utilizado na fase de acumulação permita a utilização de uma entrada de 1 bit adicional, para completar a operação de inversão ($Y_t = -X_t$) iniciada na fase ②. Este objectivo foi agora concretizado observando que o vector de *carry* obtido na saída do somador do tipo *carry-save* apresenta-se sempre deslocado de uma posição para a esquerda, dando assim origem ao aumento do número de bits dos vectores da saída referido anteriormente, e que foi evitado através da utilização do incrementador. Este deslocamento pode ser convenientemente aproveitado precisamente para efectuar a adição do bit $\mathsf{AD\_carry}$, obtido na fase ② da operação descrita, conforme se encontra ilustrado na figura 7.5.

Desta forma, considerando, a título de exemplo, a implementação de um processador com $h = 16$ linhas de elementos processadores activos, o valor obtido à saída do bloco de acumulação de cada elemento processador será dado por:

$$\begin{aligned} Z_t &= 2^8 \times \left(\mathsf{C^u\_acc}\,[3\,..\,0] + \mathsf{C^d\_acc}\,[8]\right) \\ &+ 2 \times \left(\mathsf{C^d\_acc}\,[7\,..\,1]\right) \\ &+ \left(\mathsf{S\_acc}\,[7\,..\,0] + \mathsf{AD}\,[7\,..\,0] + \mathsf{AD\_carry}\right). \end{aligned} \tag{7.3}$$

7.2.2 Elemento processador passivo

Ao contrário dos elementos processadores activos, descritos na secção anterior, os elementos processadores passivos são, apenas, responsáveis pelo transporte e deslocamento dos pixels da área de pesquisa, fazendo passar cada um dos macroblocos candidatos sobre um dos blocos activos do processador.

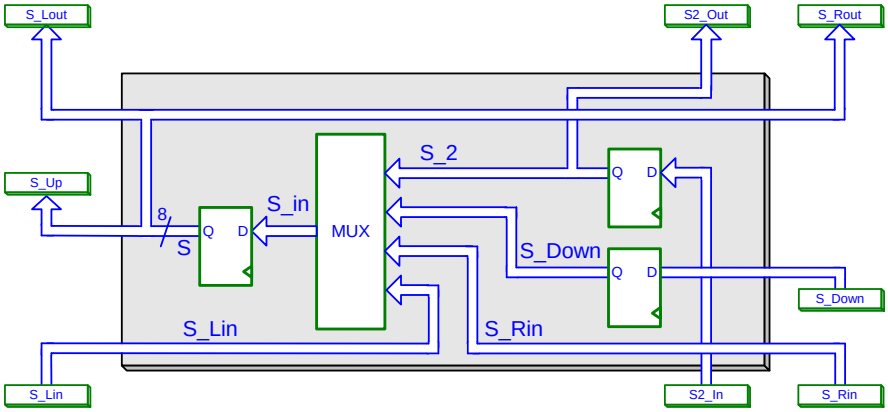


Figura 7.6: Estrutura interna do elemento processador passivo.

Como consequência, a estrutura interna deste tipo de elemento processador é em tudo idêntica ao circuito de registos de transporte da área de pesquisa do elemento processador activo, conforma se ilustra na figura 7.6.

O funcionamento deste tipo de elemento processador é, consequentemente, idêntico ao do circuito de registos de transporte do elemento processador activo, tanto no que diz respeito aos registos de transporte propriamente ditos como aos de transporte em modo transparente, não se repetindo, por isso, a descrição do funcionamento deste circuito.

7.2.3 Melhoria do desempenho

Conforme se referiu anteriormente, uma das preocupações fundamentais no projecto do processador para estimação de movimento foi a de conseguir obter uma arquitectura eficiente, maximizando a frequência de funcionamento. Contudo, tal é apenas conseguido através da utilização de unidades lógicas e aritméticas rápidas. Foi nesse sentido que se optou, tal como se descreveu anteriormente, pela utilização de somadores do tipo *carry-save* para proceder à acumulação dos valores parciais da medida de similaridade entre os vários elementos processadores.

Contudo, após um estudo mais detalhado da arquitectura do elemento processador activo apresentada, constatou-se que existe, ainda, uma importante componente do caminho crítico por optimizar. Na figura 7.7 apresenta-se um diagrama em que se assinala o caminho crítico do circuito. Verifica-se que este caminho crítico tem início no bloco de controlo, aquando da geração do sinal de selecção da saída do multiplexador responsável pela escolha do pixel do sub-bloco do macrobloco de referência a considerar. Atravessa, então, o somador para cálculo do valor absoluto da diferença e, através do sinal $\mathsf{AD}[7]$ (utilizado no cálculo do valor do sinal $\mathsf{C^d\_acc}[8]$), passa pelo somador do tipo *carry-save*,

pelo incrementador a partir do sinal de controlo, até ao bit mais significativo do sinal C^u_acc (bit $C^u\_acc[3]$ no caso de uma implementação com blocos activos constituídos por $h = 16$ linhas de elementos processadores).

Na impossibilidade de optimizar significativamente os blocos correspondentes ao multiplexador de selecção do pixel do macrobloco de referência e ao somador do tipo *carry-save* responsável pela acumulação, concluiu-se que o esforço de optimização se deve concentrar nos elementos somadores constituintes do bloco de cálculo do valor absoluto da

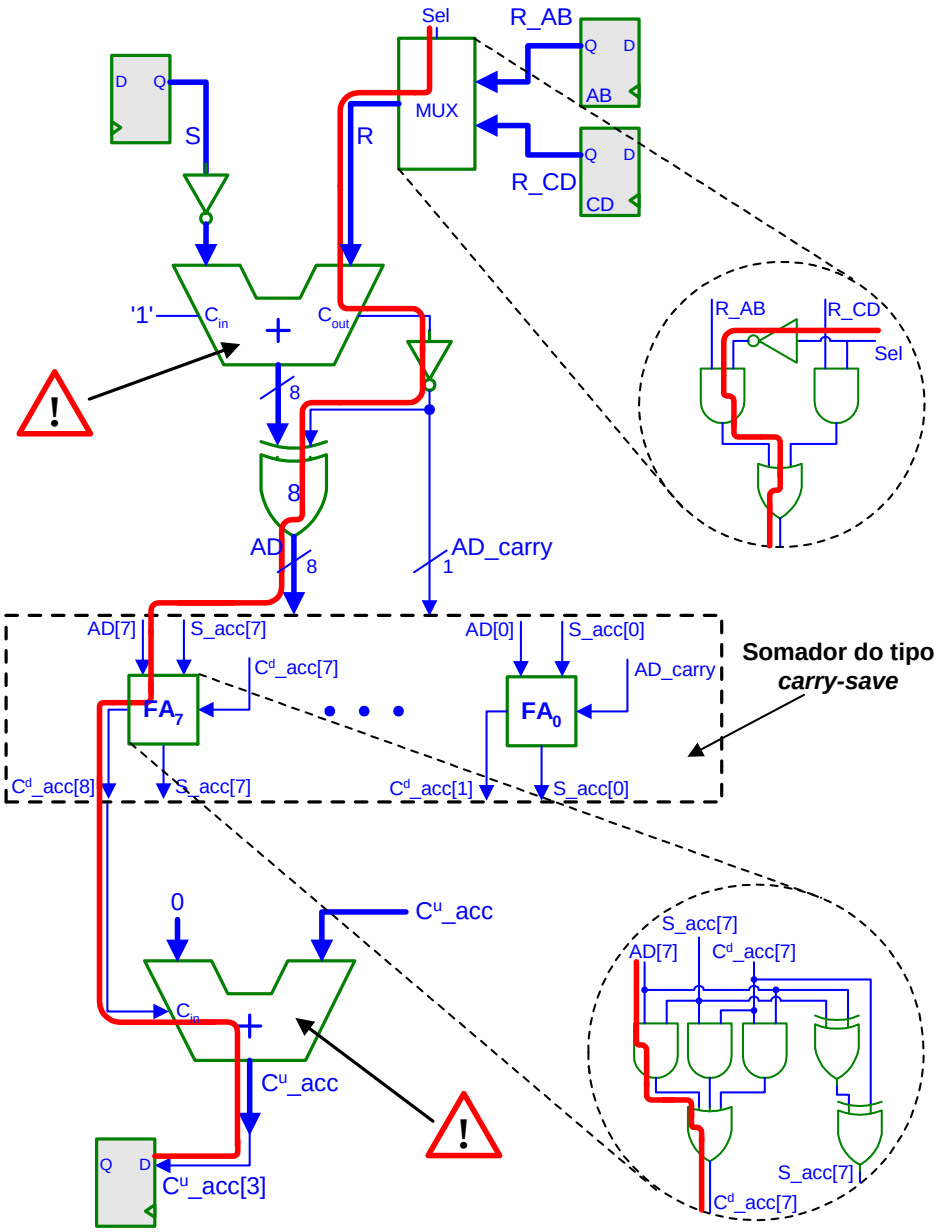


Figura 7.7: Caminho crítico da estrutura interna do elemento processador activo.

diferença e do incrementador. Recorreu-se, então, ao estudo efectuado no capítulo 6 sobre arquitecturas rápidas de somadores, para avaliar qual o tipo de somador que apresenta as características mais apropriadas para estes blocos.

Face aos resultados apresentados na tabela 6.2 e nas figuras 6.16, 6.17 e 6.18, e considerando o estudo do caminho crítico do elemento processador activo realizado (ver figura 7.7), optou-se por utilizar a estrutura correspondente ao somador do tipo "Sklansky rápido". Esta opção justifica-se pelo facto de este tipo de somador apresentar vantagens evidentes sempre que o caminho crítico entra no dispositivo pela entrada de *carry*, conforme se mostrou no capítulo 6. Como se referiu, esta característica é particularmente vantajosa no bloco de incrementação utilizado no circuito de acumulação do elemento processador activo (ver figura 7.7).

Assim sendo, apresenta-se na tabela 7.1 a contabilização do tempo correspondente ao caminho crítico do circuito, discriminando os vários dispositivos constituintes do elemento processador activo. O tempo total obtido foi de 23 unidades, o que corresponde ao mínimo alcançado a partir da análise detalhada das características das arquitecturas de somadores rápidos. Convém referir que o valor apresentado correspondente ao somador do bloco de cálculo do valor absoluto da diferença foi calculado utilizando a expressão correspondente ao tempo de propagação entre as entradas do dispositivo e a saída ($T_s^{a,b} = 2\log_2 \hat{w} + 6$), deduzida na equação 6.39c.

Circuito	**Dispositivo**	**Tempo**
Circuito de registos de transporte	Multiplexador	2
Circuito de cálculo do valor absoluto da diferença	Somador "Sklansky-rápido"	12
	INV	0
	XOR	2
Circuito de cálculo da acumulação	Somador *carry-save*	3
	Incrementador	4
TOTAL		23

Tabela 7.1: Tempo correspondente ao caminho crítico do circuito do elemento processador activo.

7.3 Matriz de elementos processadores

A matriz de elementos processadores proposta teve, como base, a estrutura descrita na secção 5.3 e apresentada na figura 5.16. Conforme se referiu aquando da apresentação desta estrutura, as características do processador desenvolvido dependem de um conjunto de parâmetros, nomeadamente (ver figura 5.16):

- a dimensão do macrobloco de referência (N);
- a dimensão do sub-bloco da área de referência sob processamento em cada deslocamento horizontal completo da área de pesquisa (ℓ);
- a dimensão da área de pesquisa (p);
- o número de blocos activos (NC);
- o número de linhas de elementos processadores (h).

A aplicação deste conjunto de valores às equações 5.4 a 5.9 permite definir as características da estrutura de elementos processadores a implementar.

Nesta matriz de elementos processadores com estrutura regular efectuam-se, então, todas as ligações necessárias entre os vários elementos processadores activos e passivos, por forma a garantir a transferência dos dados correspondentes ao macrobloco de referência, à área de pesquisa e dos vários valores parciais da soma dos valores absolutos das diferenças. Com será de esperar, a única excepção a esta característica de regularidade dar-se-á nas fronteiras da estrutura, nas quais se podem distinguir três casos distintos:

- nas extremidades laterais, onde efectua-se a ligação entre a coluna de elementos processadores passivos localizados na extremidade direita da matriz e a coluna de elementos processadores activos localizados na extremidade esquerda da matriz, por forma a obter a estrutura com características cilíndricas descrita na secção 5.3.1 e representada na figura 5.9(b);
- na extremidade inferior interligam-se os sinais obtidos a partir dos registos de entrada do macrobloco de referência (R) (ver figura 7.1) com os elementos processadores que integram a linha inferior de cada bloco activo do processador; também nesta extremidade se interligam os registos de entrada da área de pesquisa (S) com todos os elementos processadores da linha inferior da matriz de processamento; para além destas ligações, colocam-se a zero os valores parciais da soma dos valores absolutos das diferenças: vectores $\mathsf{S\_acc}[7..0]|_{h=15}$, $\mathsf{C^d\_acc}[7..1]|_{h=15}$ e $\mathsf{C^u\_acc}[3..0]|_{h=15}$, para o caso particular de uma implementação com $h = 16$ linhas de elementos processadores (linhas definidas no intervalo $[0, 15]$);
- na extremidade superior ligam-se os sinais resultantes da adição parcial dos valores absolutos das diferenças obtidos nos elementos processadores activos, ao somador em árvore respectivo.

As ligações descritas encontram-se ilustradas na figura 7.8, para o caso particular de uma implementação com um único bloco activo ($NC = 1$) e com $N = p = h = 4$.

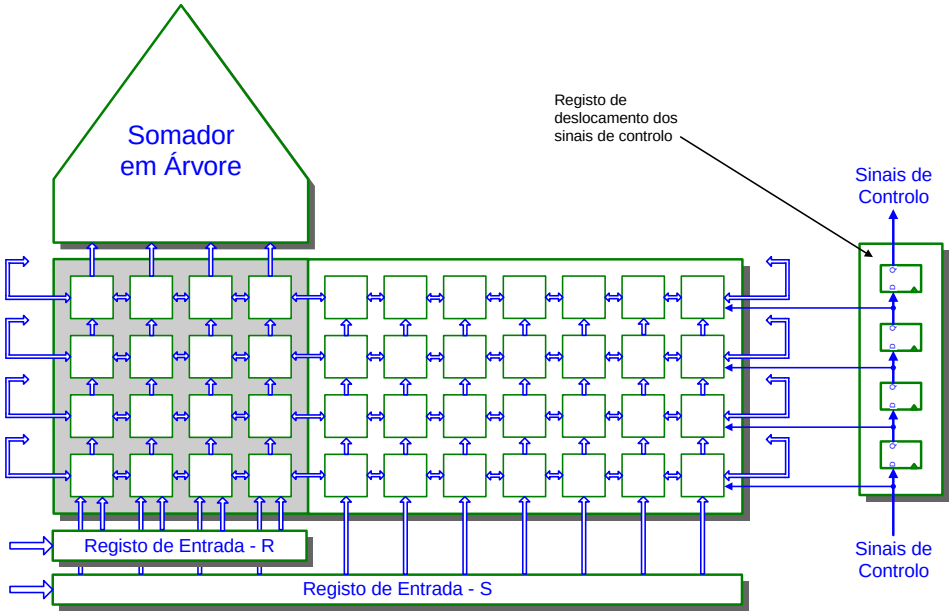


Figura 7.8: Matriz de processamento composta pelos vários elementos processadores activos e passivos interligados segundo uma estrutura regular, com $N = p = h = 4$ e $NC = 1$.

Nesta figura, encontra-se também representado um bloco tipo registo de deslocamento que transporta alguns dos sinais de controlo necessários à matriz de processamento e a outros blocos subsequentes, nomeadamente, o bloco somador em árvore e o bloco comparador. Desta forma, torna-se não só possível efectuar a sincronização dos sinais correspondentes ao valor da medida de similaridade do macrobloco candidato actual com o par de valores correspondentes às coordenadas do vector de movimento respectivo, como também dos sinais de controlo necessários à inicialização e bloqueio dos registos de comparação.

7.4 Somador em árvore

O bloco correspondente ao somador em árvore é responsável pela soma dos ℓ valores parciais da medida de similaridade do macrobloco candidato em consideração (ver figura 5.16). À sua saída obtém-se a medida de similaridade a usar como termo de comparação entre os vários macroblocos candidatos.

Antes de se proceder à descrição do somador em árvore, analisa-se o número mínimo de bits a utilizar. Conforme se descreveu na secção 7.2 e, em particular, na figura 7.4, a representação utilizada pelos elementos processadores caracteriza-se por três vectores distintos: $\mathsf{S\_acc}\,[7\,..\,0]$, $\mathsf{C^d\_acc}\,[7\,..\,1]$ e $\mathsf{C^u\_acc}\,[(\log_2 \mathsf{h} - 1)\,..\,0]$. Estes vectores permitem

representar o valor máximo eventualmente obtido em cada uma das ℓ colunas de elementos processadores da matriz de processamento através de $(\log_2 h + 8)$ bits, correspondente ao valor $h \times 2^8$ (ver equação 7.2).

Contudo, verifica-se que esta gama dinâmica pode, em alguns casos, ser insuficiente para representar o valor final da medida de similaridade obtido na saída do somador em árvore. De facto, o valor máximo que se pode obter à saída é agora $\ell \times h \times 2^8$, requerendo um total de $(\log_2 \ell + \log_2 h + 8)$ bits.

Os vectores que representam os valores parciais do bloco somador em árvore utilizam, tal como na matriz de processamento, uma notação do tipo da utilizada nos somadores do tipo *carry-save* (ver figura 7.9):

- Vector de soma - $\mathsf{S\_acc}\,[(\log_2 \mathsf{h} + 7)\,..0]$ - constituído, inicialmente, através da concatenação do vector de 8 bits $\mathsf{S\_acc}$ e do vector de $\log_2 h$ bits $\mathsf{C^u\_acc}$, obtidos à saída da matriz de processamento;
- Vector de *carry* menos significativo - $\mathsf{C^d\_acc}\,[(\log_2 \mathsf{h} + 7)\,..0]$ - constituído, inicialmente, através da concatenação do vector de 7 bits $\mathsf{C^d\_acc}$ obtido à saída da matriz de processamento com um vector nulo, constituído por $\log_2 h$ bits a zero;
- Vector de *carry* mais significativo - $\mathsf{C^u\_acc}\,[(\log_2 \ell - 1)\,..0]$ - inicialmente com todos os bits a zero.

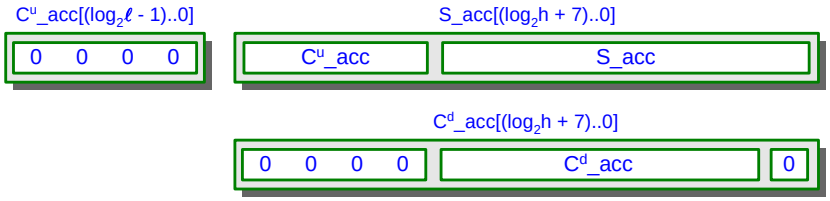


Figura 7.9: Vectores com os valores parciais da medida de similaridade no somador em árvore.

A arquitectura do somador em árvore foi projectada, fundamentalmente, com dois objectivos: *i)* reduzir ao mínimo o tempo de cálculo; *ii)* utilizar a mínima quantidade de recursos de *hardware* possível, garantindo a minimização do tempo de propagação.

Para atingir o primeiro objectivo utilizou-se uma estrutura em árvore binária, com um número total de níveis da ordem de $\mathcal{O}(\log_2 \ell)$. Para além disso, e à semelhança do que se fez no caso dos elementos processadores activos, procedeu-se a uma optimização exaustiva da estrutura interna dos elementos de processamento agora utilizados no somador em árvore, conforme se descreverá a seguir.

Para atingir o segundo objectivo, houve a necessidade de se realizar um esforço de projecto suplementar. De facto, reconhece-se facilmente que a solução mais simples consistiria em utilizar um somador em árvore do tipo binário para que acomodasse $2^{\lceil \log_2 \ell \rceil}$ entradas, com $\lceil \log_2 \ell \rceil$ níveis de soma. Contudo, o facto de existirem $2^{\lceil \log_2 \ell \rceil} - \ell$ entradas não utilizadas daria origem a uma utilização desnecessária de muitos recursos de *hardware*, para além de, em geral, requerer um nível suplementar de processamento.

Para o cumprimento dos dois objectivos acima referidos, optou-se por estabelecer um método optimizado de projecto automático de somadores em árvore com um número de entradas diferente de uma potência inteira de 2. Por isso, faz-se a descrição do somador em árvore em duas secções distintas, correspondentes à descrição do somador em árvore com um número de argumentos (ℓ) correspondente a uma potência inteira de 2 ($\ell = 2^n$) - secção 7.4.1, e ao somador com um número de argumentos (ℓ) diferente de uma potência inteira de 2 ($\ell \neq 2^n$) - secção 7.4.2.

7.4.1 Somador com $\ell = 2^n$ argumentos

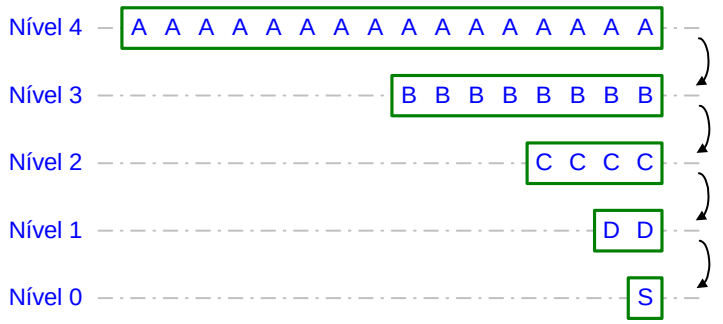
Este tipo de somador apresenta uma estrutura em árvore binária tradicional, somando conjuntos de 2^i valores parciais em cada um dos $\log_2 \ell$ níveis da estrutura em árvore: o nível i é composto por 2^{i-1} elementos de processamento, que efectuam, cada um, a soma de dois operandos da soma que é utilizada no nível seguinte $(i-1)$.

Na figura 7.10(a) apresenta-se o esquema de processamento utilizado pelo somador em árvore, para o caso particular de $\ell = 16$. Em cada um dos 4 níveis é feito o processamento de 2^i valores parciais da medida de similaridade.

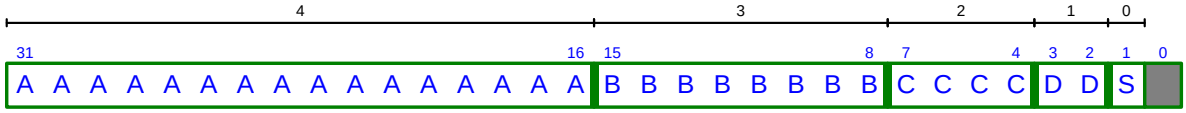
O número de somadores de duas entradas necessário é:

$$\sum_{t=1}^{n} 2^{t-1} = 2^n - 1 = \ell - 1 \tag{7.4}$$

Para armazenar os resultados parciais dos n níveis utilizam-se $2\ell - 1 = 2^{n+1} - 1$ elementos de memória. Na figura 7.10(b) encontra-se ilustrada a organização do espaço de memória para o caso particular de uma implementação com $\ell = 16$ operandos. Cada nível da árvore encontra-se, assim, armazenado no conjunto de posições entre 2^i e $2^{i+1} - 1$. Um dado elemento de processamento pe localizado no nível i, efectua a soma dos operandos armazenados mas posições $2^i + 2pe$ e $2^i + 2pe + 1$, e guarda o resultado da operação na posição $2^{i-1} + pe$. O resultado final é obtido na posição 1, após realizar os $n = \log_2 \ell$ níveis de processamento.



(a) Esquema de processamento hierárquico.



(b) Armazenamento em memória dos valores.

Figura 7.10: Processamento do somador em árvore com $\ell = 2^4$ operandos.

7.4.2 Somador com $\ell \neq 2^n$ argumentos

Tal como foi referido anteriormente, a estrutura deste tipo de somador em árvore é o resultado de um esforço significativo de investigação para minimizar a utilização desnecessária de *hardware* e o número de níveis de processamento do somador, continuando, no entanto, a garantir características de projecto e configuração automáticas.

A abordagem utilizada foi no sentido de dividir os ℓ operandos a somar em $\lceil \log_2 \ell \rceil$ sub-árvores, tendo cada uma um conjunto de 2^i operandos, com $i = 0 \ldots \lfloor \log_2 \ell \rfloor$. Na figura 7.11 encontra-se ilustrado o caso particular de uma implementação com $\ell = 15$ operandos. Assim, verifica-se que cada sub-árvore consiste, na realidade, num somador em árvore com um número de operandos correspondente a uma potência inteira de 2, obtendo-se, em cada um deles, o resultado da soma parcial respectiva: A_1, B_2, C_4 e D_8.

A forma como os ℓ operandos são distribuídos pelas $\lceil \log_2 \ell \rceil$ sub-árvores é feita no sentido de preencher completamente o maior número de sub-árvores possível, correspondente à situação de se ter, para uma dada sub-árvore de ordem i, a condição: $\ell \bmod (2^{i+1}) > \ell \bmod (2^i)$, com $i = \lfloor \log_2 \ell \rfloor \ldots 1, 0$ e começando-se por preencher, em primeiro lugar, as sub-árvores com maior número de operandos.

No último nível de cada sub-árvore de ordem i é apenas necessário efectuar a acumulação do resultado obtido com o valor calculado nas sub-árvores de menor peso (S_i), retornado o valor assim obtido (S_{i+1}) para a sub-árvore seguinte, de maior peso. Assim, no caso particular de uma dada sub-árvore de ordem i não possuir, para um dado valor

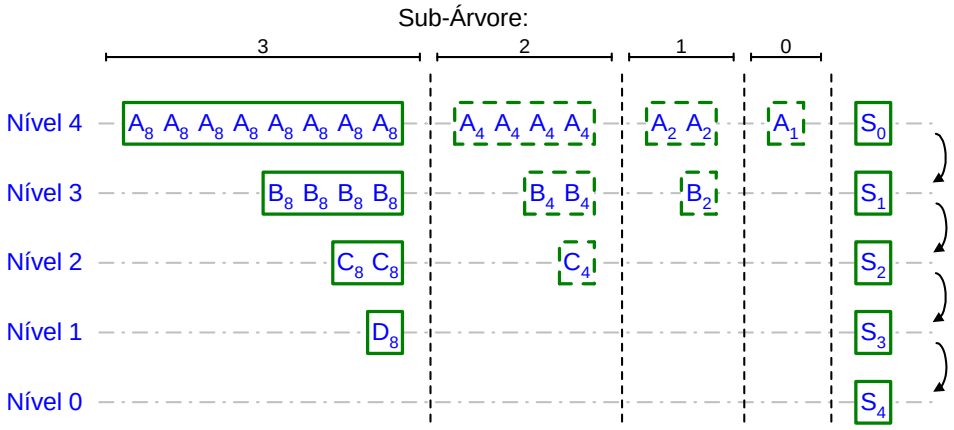


Figura 7.11: Esquema de processamento do somador em árvore com $\ell \neq 2^n$ argumentos.

de ℓ, quaisquer operandos, esta tem, apenas, de passar o resultado parcial obtido nas sub-árvores de menor peso (S_i) para a entrada (S_{i+1}) da sub-árvore seguinte, de maior peso. Deste modo, evita-se o desperdício de recursos de *hardware*. No dimensionamento dum somador de ℓ operandos em árvore, com $\lceil \log_2 \ell \rceil$ níveis de processamento, apenas a sub-árvore de maior peso ($i = \lfloor \log_2 \ell \rfloor$) tem implementação garantida, pelo que se representaram todas as outras a traço interrompido (ver figura 7.11).

Verifica-se assim que, no caso de todas as sub-árvores serem implementadas, correspondente a ter-se $\ell = 2^k - 1$, com $k \in \mathcal{N}$, são necessários um total de:

$$\sum_{n=0}^{\lfloor \log_2 \ell \rfloor} (2^n - 1) = \sum_{n=0}^{\lfloor \log_2 \ell \rfloor} 2^n - (\lfloor \log_2 \ell \rfloor + 1) \tag{7.5a}$$

$$= \frac{1 - 2^{\lfloor \log_2 \ell \rfloor + 1}}{1 - 2} - (\lfloor \log_2 \ell \rfloor + 1) \tag{7.5b}$$

$$= 2 \times 2^{\lfloor \log_2 \ell \rfloor} - (\lfloor \log_2 \ell \rfloor + 2) \tag{7.5c}$$

somadores de duas entradas. Para além disso, utilizam-se $2^{\lceil \log_2 \ell + 1 \rceil} - 1$ elementos de memória para armazenar os valores intermédios, com uma organização semelhante à apresentada na figura 7.12, correspondente ao caso particular de $\ell = 15$ operandos.

Nesta figura é possível verificar que todos os sinais de entrada são colocados, inicialmente, na metade superior deste vector, começando-se, assim, por garantir o preenchimento das sub-árvores de maior peso. Os valores finais obtidos em cada sub-árvore i são colocados nas posições $2^{\lceil \log_2 \ell \rceil - i}$, com $i = 0 \ldots \lfloor \log_2 \ell \rfloor$. Estes valores são, então, acumulados no último nível de cada sub-árvore, somando o valor correspondente à posição $2^{\lceil \log_2 \ell \rceil - i}$ com o valor S_i, obtido a partir da sub-árvore de menor peso e localizado na posição $2^{\lceil \log_2 \ell \rceil - (i-1)} - 1$. O valor obtido é, então, armazenado na posição $2^{\lceil \log_2 \ell \rceil - i} - 1$.

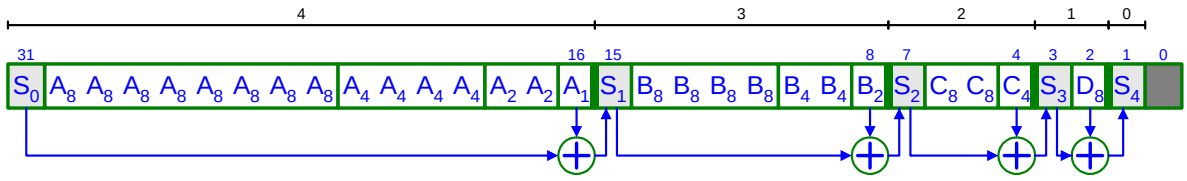


Figura 7.12: Distribuição dos valores intermédios pelo espaço de memória utilizado pelo somador em árvore com $\ell \neq 2^n$ argumentos.

Qualquer que seja o número de operandos (ℓ) é possível, utilizando este mecanismo, obter o resultado final na posição 1, utilizando, para isso, um total de $\lceil \log_2 \ell \rceil$ níveis de processamento.

7.4.3 Elementos de processamento

Efectuada a descrição do tipo de estruturas utilizadas para o somador em árvore, resta apenas descrever a estrutura interna dos elementos de processamento utilizados. Conforme se apresentou na figura 7.9, as operações de soma realizadas recebem, como argumento, os vectores de soma e de *carry* obtidos através de unidades do tipo *carry-save*. Assim sendo, constata-se facilmente da existência de dois grupos de valores a somar em cada elemento processador:

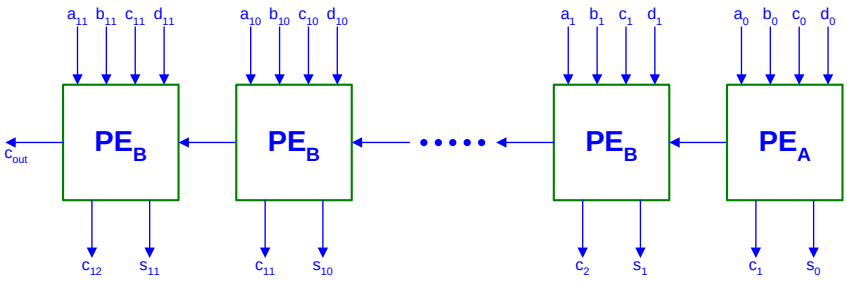
- um grupo constituído por 4 operandos, correspondentes aos bits localizados nas $\log_2 h + 8$ posições menos significativas: $\mathsf{S\_acc}_{2i}$, $\mathsf{C^d\_acc}_{2i}$, $\mathsf{S\_acc}_{2i+1}$ e $\mathsf{C^d\_acc}_{2i+1}$;
- um grupo constituído por 2 operandos, correspondentes aos bits localizados nas $\log_2 \ell$ posições mais significativas: $\mathsf{C^u\_acc}_{2i}$ e $\mathsf{C^u\_acc}_{2i+1}$.

A adição do primeiro grupo de operandos é efectuada usando, uma vez mais, técnicas de cálculo rápido do tipo *carry-free*, para reduzir o tempo de cálculo. Os somadores utilizados são do tipo *compressor (4,2)* [?], ilustrados na figura 7.13, que se caracterizam por receber 4 bits de entrada, eventualmente com uma entrada adicional de carry ($\mathrm{PE_B}$), e de fornecerem à saída um bit de soma e dois bits de *carry*: um correspondente ao vector de *carry* de saída (C), e outro que será propagado para o somador seguinte. A estrutura interna dos somadores utilizados encontra-se ilustrada nas figuras 7.13(b) e 7.13(c).

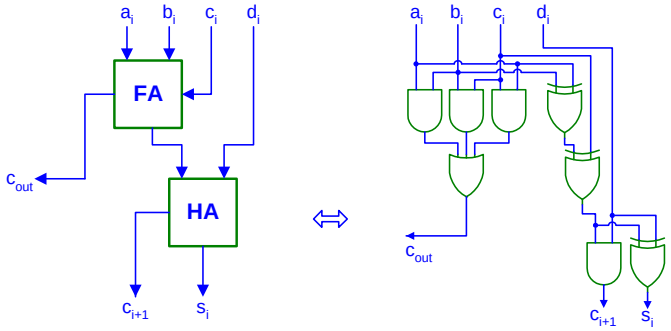
Conforme se pode verificar, este tipo de somador apresenta, como característica fundamental, o facto de não existir caminho de propagação do sinal de *carry* de entrada (c_{in}) para o sinal de *carry* de saída (c_{out}), tornando assim o tempo de computação da soma independente do número de bits dos operandos de entrada. Assim, considerando o modelo apresentado na tabela 6.1, cada um destes blocos apresenta os seguintes tempos

de propagação:

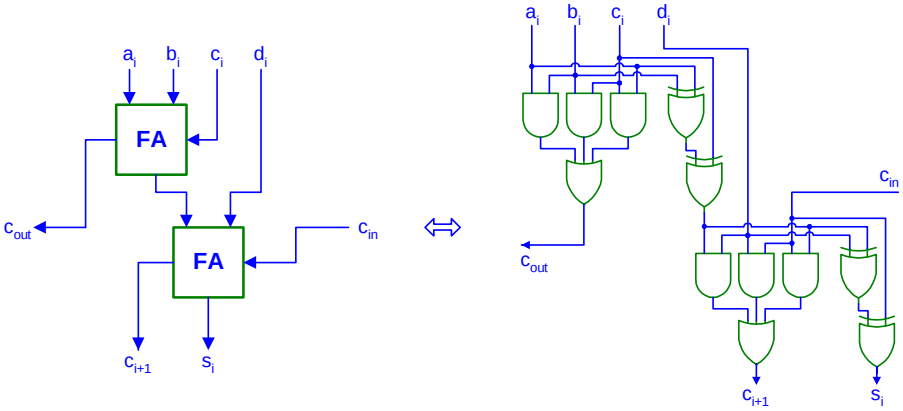
$$T\left(a,b,c,d \rightarrow c,s\right) = 8$$
$$T\left(a,b,c,d \rightarrow c_{out}\right) = 3$$
$$T\left(c_{in} \rightarrow c,s\right) = 4$$
$$T\left(c_{in} \rightarrow c_{out}\right) = 0$$



(a) Estrutura de somadores do tipo *compressor (4,2)*.



(b) Somador *compressor* do tipo A.



(c) Somador *compressor* do tipo B.

Figura 7.13: Estrutura do somador do tipo *compressor (4,2)* para o cálculo dos $(\log_2 h + 8)$ bits menos significativos, para o caso de $h = 16$ linhas de elementos processadores em cada bloco activo.

À saída deste tipo de somador obtêm-se os seguintes sinais:

- $\mathsf{S}\,[(\log_2 \mathsf{h}+7)\,..\,0]$ e $\mathsf{C}\,[(\log_2 \mathsf{h}+7)\,..\,1]$, correspondentes aos vectores de soma e de *carry*, respectivamente;
- $\mathsf{c_{out}}$ e $\mathsf{C}\,(\log_2 \mathsf{h}+8)$, correspondentes aos 2 bits de *carry* que serão enviados para o somador que irá processar os $\log_2 \ell$ bits mais significativos, a que correspondem tempos de propagação iguais a 3 e 8, respectivamente.

Desta forma, o somador correspondente aos $\log_2 \ell$ bits mais significativos deve receber, como argumentos, 2 bits correspondentes aos operandos de entrada e 2 vectores correspondentes aos dois sinais de *carry* obtidos do somador *compressor*, obtendo-se, à saída, o vector de soma correspondente. A estrutura de soma concebida para este somador é do tipo *carry-propagate* com duas linhas de propagação dos bits de *carry* [**?**], conforme se ilustra na figura 7.14 para o caso de uma implementação com $\ell = 16$ colunas de elementos processadores em cada bloco activo. O bit $\mathsf{c_{out}}$ obtido no andar anterior é somado e propagado no nível superior, constituído por blocos do tipo *full-adder*, ao passo que o bit $\mathsf{C}\,(\log_2 \mathsf{h}+8)$ é somado e propagado no nível inferior, constituído por blocos do tipo *half-adder*. Os sinais de soma obtidos à saída deste bloco correspondem aos bits do vector $\mathsf{C^u\_acc}\,[(\log_2 \ell + \log_2 \mathsf{h}+7)\ldots(\log_2 \mathsf{h}+8)]$, que serão, depois, passados para o nível seguinte do somador em árvore. Como o número de bits utilizado por este tipo de representação é suficiente para representar a gama dinâmica imposta pelo valor dos operandos, pode-se ignorar o valor dos dois bits de *carry* de saída deste bloco somador (ver figura 7.14).

Considerando os tempos de propagação dos sinais $\mathsf{c_{out}}$ e $\mathsf{C}\,(\log_2 \mathsf{h}+8)$ obtidos à saída do bloco *compressor (4,2)* correspondentes a 3 e 8, respectivamente, e considerando a estrutura interna dos blocos *full-adder* e *half-adder* ilustrada nas figuras 7.15(a) e 7.15(b),

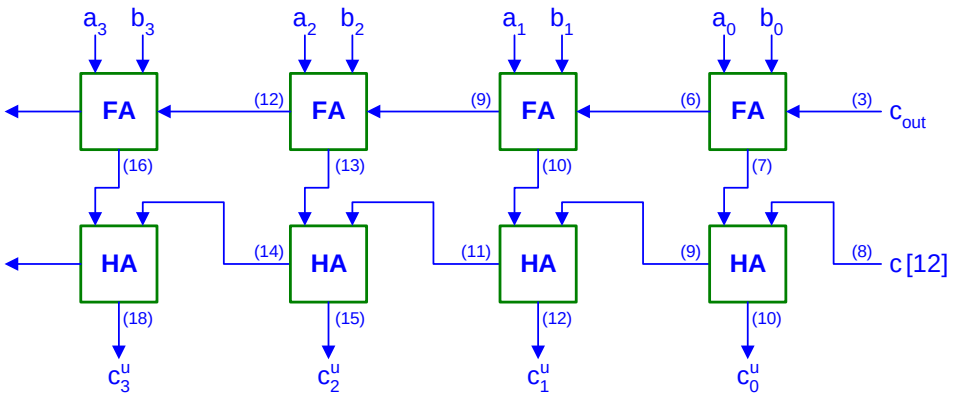
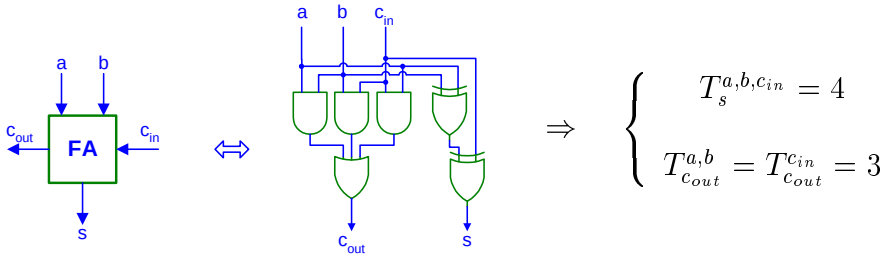
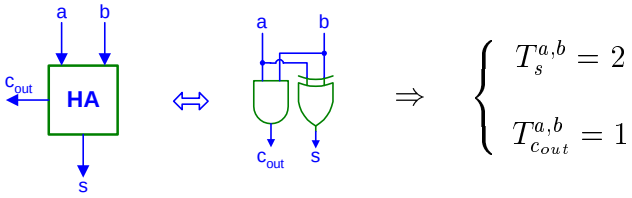


Figura 7.14: Estrutura utilizada para somar os $\log_2 \ell$ bits mais significativos ($\ell = 16$).

o tempo de propagação correspondente a este circuito é o indicado pelos números junto de cada ligação na figura 7.14, correspondendo a um total de $T = 18$ para o caso de uma implementação com $\ell = 16$.



(a) Bloco tipo *full-adder*



(b) Bloco tipo *half-adder*

Figura 7.15: Estrutura interna dos blocos constituintes do somador Adder2C, utilizado para somar os $\log_2 \ell$ bits mais significativos.

A estrutura completa do bloco de soma de cada elemento de processamento do somador em árvore é a ilustrada na figura 7.16, sendo constituída por um somador do tipo *compressor (4,2)* para os $\log_2 h + 8$ bits menos significativos e por um somador do tipo Adder2C, com 2 linhas de propagação de *carry* para os $\log_2 \ell$ bits mais significativos.

Nesta figura encontram-se também representados os vários registos de saída presentes em cada elemento de processamento, alcançando-se uma estrutura do somador em árvore do tipo sistólica.

A saída do bloco somador em árvore será, assim, definida pelo conjunto de vectores final S_acc, C^d_acc e C^u_acc, que representa o valor da medida de similaridade do bloco candidato em consideração.

Antes de terminar a descrição deste bloco convém ainda referir que, à semelhança do que se fez com a matriz de processamento, implementou-se também, em cada somador em árvore, um registo de deslocamento de alguns sinais de controlo. Estes sinais são necessários ao somador propriamente dito e a outros blocos subsequentes, garantindo, assim, o correcto sincronismo no funcionamento dos vários blocos do circuito.

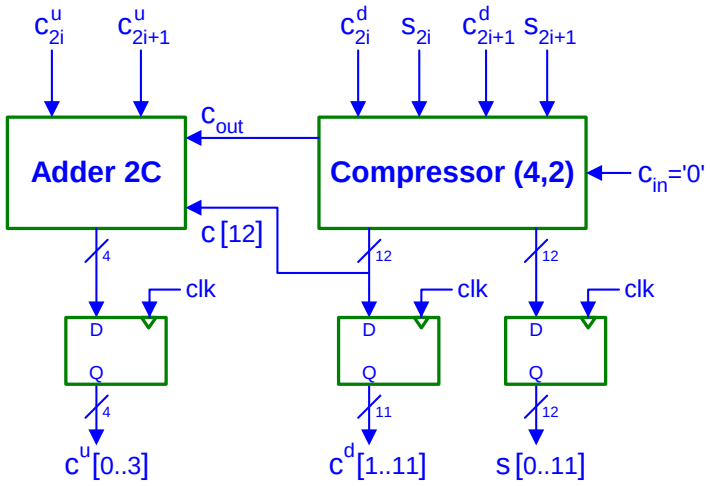


Figura 7.16: Estrutura interna de cada elemento de processamento do somador em árvore.

7.5 Bloco de comparação

O bloco de comparação é o bloco responsável pela selecção do macrobloco candidato com maior medida de similaridade. Esta selecção é feita utilizando os valores de medida de similaridade calculados em cada bloco activo do processador e acumulados no somador em árvore respectivo. Este bloco pode ser decomposto em dois níveis distintos:

- Nível de pré-selecção, onde se efectua a selecção do macrobloco candidato óptimo de entre o conjunto de macroblocos processados num dado bloco activo;
- Nível de selecção hierárquico, onde se efectua a selecção do macrobloco candidato com maior medida de similaridade de entre o conjunto de macroblocos seleccionados no nível de pré-selecção.

7.5.1 Nível de pré-selecção

Este primeiro bloco de selecção é o responsável pela escolha do macrobloco candidato que apresenta maior medida de similaridade de entre o conjunto de macroblocos processados por um determinado bloco activo do circuito. Para isso, neste primeiro nível de selecção é utilizado um elemento de memória, para armazenar os valores associados ao macrobloco candidato que apresenta, até um dado momento, maior medida de similaridade. A selecção é efectuada comparando o valor da medida de similaridade calculado no somador em árvore correspondente e o valor armazenado neste elemento de memória.

O diagrama de blocos de cada elemento de processamento deste primeiro nível de selecção encontra-se representado na figura 7.17. Este é, essencialmente, constituído por

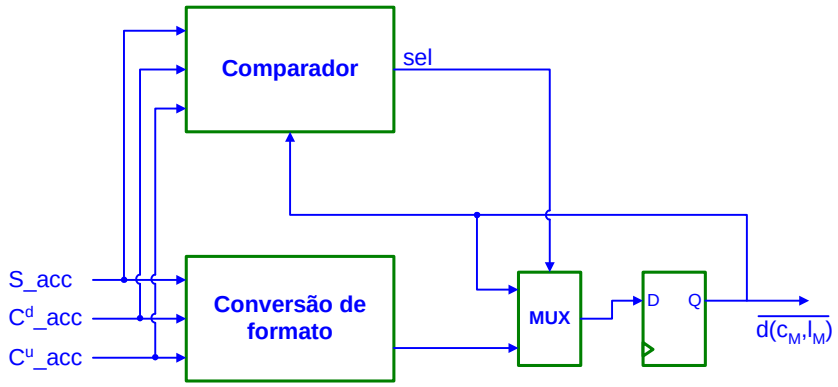


Figura 7.17: Diagrama de blocos de cada elemento de processamento do nível de pré-selecção.

dois módulos fundamentais:

- um módulo comparador, que determina se o valor da medida de similaridade apresentado pelo macrobloco candidato em consideração é maior do que o valor óptimo até então encontrado;
- um módulo de conversão de formato, que converte o formato do valor da medida de similaridade obtido à saída do somador em árvore, constituído por um vector de *carry* e um vector de soma (ver figura 7.9), num único vector.

O multiplexador permite, então, seleccionar o valor que deverá ser armazenado no elemento de memória, e que será utilizado na comparação com os macroblocos candidatos seguintes. De seguida, descreve-se, mais detalhadamente, cada um destes módulos.

Módulo comparador

A comparação efectuada neste módulo é realizada através de uma operação de subtracção que, utilizando a representação em complemento para dois, vem dada por:

$$dif = A - B = A + \overline{B} + 1 \tag{7.6}$$

Consequentemente, e considerando o valor da medida de similaridade obtido à saída do somador em árvore dada por:

$$\begin{aligned} d(c,l) = {} & 2^{(\log_2 h+8)} \times (\mathsf{C^u\_acc}\,[(\log_2 \ell - 1)\,..0]) \\ & + 2 \times \left(\mathsf{C^d\_acc}\,[(\log_2 \mathsf{h} + 7)\,..1]\right) \\ & + \mathsf{S\_acc}\,[(\log_2 \mathsf{h} + 7)\,..0], \end{aligned} \tag{7.7}$$

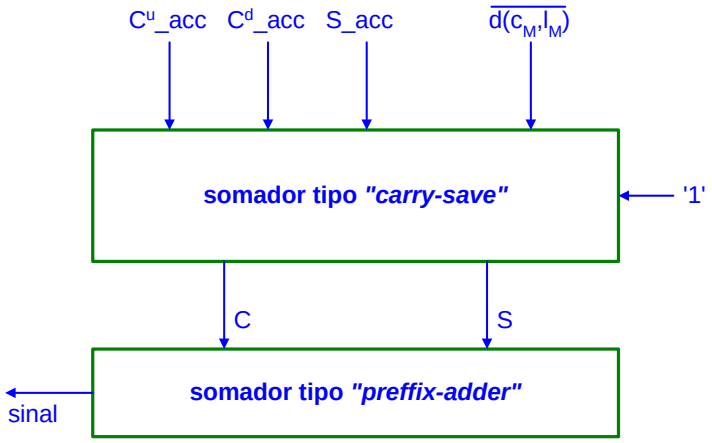


Figura 7.18: Diagrama de blocos do circuito rápido para comparação.

o valor da subtracção é calculado da seguinte forma:

$$dif = d(c,l) + \overline{d(c_M,l_M)} + 1, \tag{7.8}$$

onde $\overline{d(c_M,l_M)}$ representa o valor óptimo até então obtido da medida de similaridade, utilizando a notação de complemento para um.

Para que o resultado da comparação seja calculado da forma mais rápida possível, optou-se por efectuar a operação de subtracção em duas fases distintas: utiliza-se, numa primeira fase, um somador do tipo *carry-save*, para efectuar o cálculo da soma de $d(c,l)$ com $\overline{d(c_M,l_M)}$ e '1'. Numa segunda fase, aplica-se uma estrutura do tipo *prefix-adder*, para calcular o valor final de *dif* a partir dos vectores de bits de *carry* e de soma calculados na fase anterior (figura 7.18).

Na figura 7.19 apresenta-se a configuração dos vários operandos envolvidos nas duas fases desta operação. Para simplificar a notação, designou-se nesta figura por w o número total de bits dos vectores envolvidos: $w = \log_2 \ell + \log_2 h + 8$. Conforme se pode verificar, optou-se por concatenar o vector $\mathsf{C^d\_acc}\,[(\log_2 \mathsf{h} + 7)\,..\,1]$, obtido no somador em árvore, com o valor $'1'$, necessário para efectuar a operação de subtracção (ver equação 7.8), obtendo-se, assim, um vector com $\log_2 h + 8$ bits. Evitou-se, assim, a utilização desnecessária de recursos de *hardware*, que seriam desperdiçados caso se considerasse esta parcela como um operando independente. O resultado obtido à saída deste somador do tipo *carry-save* será, assim, constituído por dois vectores com $w = \log_2 \ell + \log_2 h + 8$ bits de comprimento:

- $\mathsf{S\_dif}\,[(\log_2 \ell + \log_2 \mathsf{h} + 7)\,..\,0]$ - vector de soma;
- $\mathsf{C\_dif}\,[(\log_2 \ell + \log_2 \mathsf{h} + 8)\,..\,1]$ - vector de *carry*.

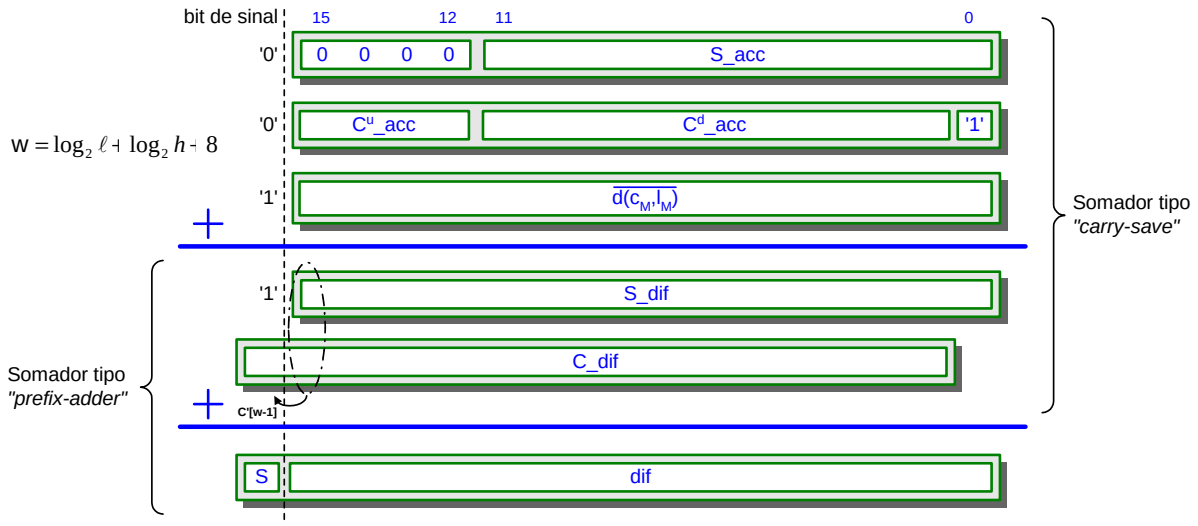


Figura 7.19: Operandos envolvidos na operação de comparação.

Como já foi visto num somador semelhante, utilizado no elemento processador activo (ver figura 7.7), o tempo de propagação de uma estrutura do tipo *carry-save* é de $T = 4$.

O cálculo do valor do bit de sinal do resultado, já que o valor propriamente dito da operação de subtracção (vector *dif*) não tem qualquer interesse prático, faz-se com base na seguinte equação (ver figura 7.19):

$$sinal = \mathsf{S\_dif}\,[\mathsf{w}] + \mathsf{C\_dif}\,[\mathsf{w}] + \mathsf{C}'\,[\mathsf{w}-1], \tag{7.9}$$

sendo $w = \log_2 \ell + \log_2 h + 8$ a dimensão dos vectores de soma e de *carry* obtidos no final da primeira fase e $\mathsf{C}'\,[\mathsf{w}-1]$ o bit de transporte obtido a partir da soma dos bits $\mathsf{S\_dif}\,[\mathsf{w}-1]$ e $\mathsf{C\_dif}\,[\mathsf{w}-1]$.

Atendendo a que $\mathsf{S\_dif}\,[\mathsf{w}] =' 1'$, resultante da representação em complemento para um do valor $\overline{d(c_M, l_M)}$ (ver figura 7.19), o resultado da equação 7.9 resume-se à soma de dois bits:

$$sinal = \overline{\mathsf{C\_dif}\,[\mathsf{w}] \oplus \mathsf{C}'\,[\mathsf{w}-1]} \tag{7.10}$$

Assim sendo, verifica-se que basta efectuar o cálculo do bit de *carry* $\mathsf{C}'\,[\mathsf{w}-1]$, resultante da soma dos vectores $\mathsf{S\_dif}$ e $\mathsf{C\_dif}$, e aplicar a equação 7.10. Para calcular o valor de $\mathsf{C}'\,[\mathsf{w}-1]$ optou-se, uma vez mais, pela utilização de um somador do tipo *prefix-adder*, conforme se ilustra na figura 7.20. Tal como se descreveu anteriormente, esta estrutura permite obter o sinal de *carry* pretendido com um tempo de propagação dado por (ver

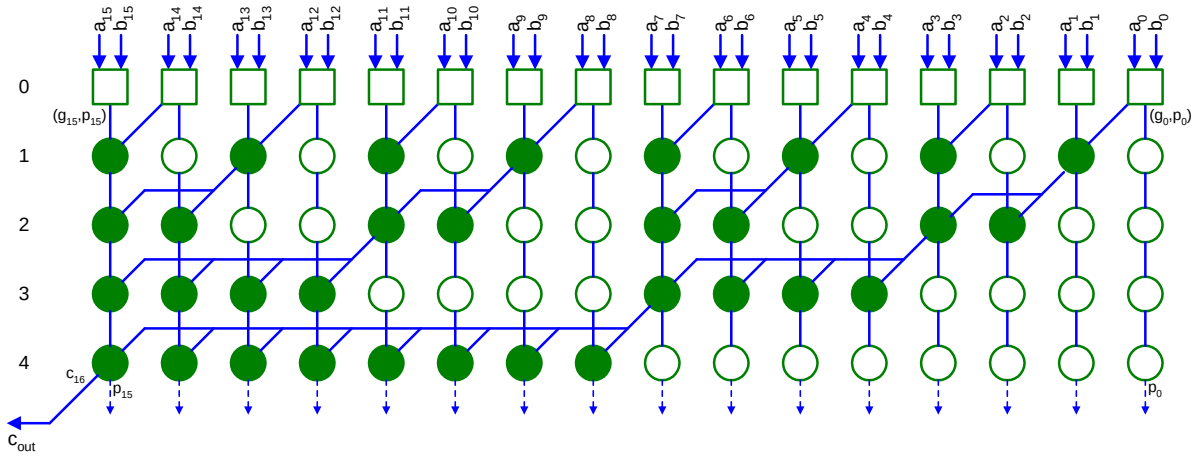


Figura 7.20: Somador do tipo *prefix-adder* utilizado para o cálculo do bit de *carry* $\mathsf{C}'[\mathsf{w}-1]$, para $\ell = h = 16$.

figura 6.12):

$$T_{\mathsf{C}'[\mathsf{w}-1]} = T_{\square} + \lceil \log_2 w \rceil \times T_{\bullet} \tag{7.11a}$$
$$= 2 + 2 \times \lceil \log_2 w \rceil \tag{7.11b}$$
$$= 2\,(1 + \lceil \log_2 w \rceil) \tag{7.11c}$$

O bit de sinal obtido através da equação 7.10 permitirá, assim, seleccionar o valor a armazenar no elemento de memória, de acordo com o seguinte esquema de decisão:

$$sinal =' 1' \quad \Leftrightarrow \quad dif = d^t(c,l) - d^t(c_M, l_M) > 0 \quad \Rightarrow \quad d^{t+1}(c_M, l_M) = d^t(c,l)$$
$$sinal =' 0' \quad \Leftrightarrow \quad dif = d^t(c,l) - d^t(c_M, l_M) < 0 \quad \Rightarrow \quad d^{t+1}(c_M, l_M) = d^t(c_M, l_M)$$

O bit de *sinal* é obtido à saída deste sub-bloco com um tempo de propagação dado por:

$$T_{sinal} = T_{carry-save} + T_{prefix-adder} + T_{XOR} \tag{7.12a}$$
$$= 4 + 2\,(1 + \lceil \log_2 w \rceil) + 2 \tag{7.12b}$$
$$= 2\,\lceil \log_2 w \rceil + 8. \tag{7.12c}$$

Considerando uma implementação com $\ell = h = 16$, obtém-se o valor $T_{sinal} = 16$.

Conforme se descreveu anteriormente, o valor correspondente à medida de similaridade do macrobloco candidato considerado até ao momento como óptimo é utilizado para realizar a comparação, usando uma representação em complemento para um. Para além

disso, é conveniente notar que, para efeitos de estimação de movimento, este valor não tem uma importância relevante, em contraste com o que acontece com o par de coordenadas correspondente, que será utilizado na fase de compensação de movimento do sistema de codificação de vídeo. Atendendo a estes factos, optou-se por armazenar, no elemento de memória, o valor de similaridade óptimo utilizando a representação em complemento para um, visto terem sido reconhecidas algumas vantagens ao nível da rapidez do circuito e da utilização de recursos de *hardware*.

De seguida, passar-se-á a descrever os procedimentos de cálculo deste valor, efectuado no módulo de conversão de formato.

Módulo de conversão de formato

Este módulo efectua, em paralelo com o módulo de comparação, a conversão do formato redundante da medida de similaridade obtida no bloco somador em árvore, composto por um vector de *carry* e um vector de soma, para o formato binário não redundante na notação em complemento para um.

A conversão efectuada resume-se, na prática, à soma dos dois vectores de entrada, conforme se ilustra na figura 7.21. Com o dimensionamento efectuado, o bit de *carry* de saída (c_{out}) assume sempre o valor $'0'$, pelo que este bit não é, na prática, considerado na explicação subsequente.

Os vectores de *carry* e de soma são somados com base numa estrutura do tipo Sklansky-lento, já apresentada na figura 6.14. Na prática, verifica-se que qualquer outra estrutura poderia ser utilizada para efectuar esta operação, desde que o seu tempo de propagação não excedesse o tempo de propagação do módulo comparador que, tal como se referiu, calcula em paralelo o valor do bit de sinal de selecção do multiplexador. Tal como se referiu na secção 6.4, esta topologia caracteriza-se por apresentar um tempo de

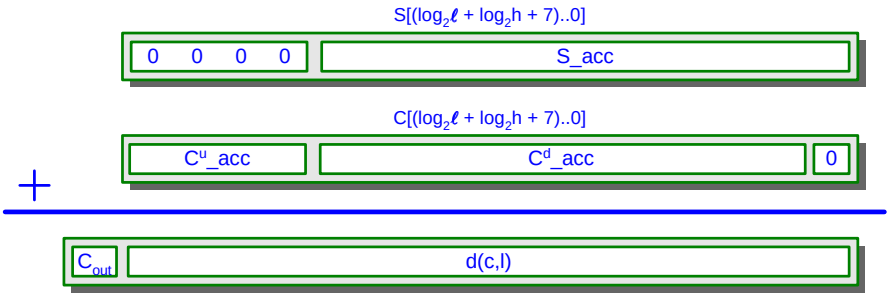


Figura 7.21: Conversão do formato utilizado para representar o valor da medida de similaridade de cada macrobloco candidato.

propagação dado por (ver equação 6.43c):

$$T_s = 2\lceil \log_2 w \rceil + 5, \tag{7.13}$$

em que $w = \log_2 \ell + \log_2 h + 8$. Considerando uma implementação com $\ell = h = 16$, obtém-se o valor $T_s = 13$.

O valor final à saída deste bloco, $\overline{d(c,l)}$, é obtido através da inversão de todos os w bits que constituem o resultado da soma.

Assim, verifica-se que o tempo máximo de propagação deste primeiro nível de comparação é dado por:

$$T_{pré\_sel} = \max\{T_{sinal}, T_s\} + T_{mux} \tag{7.14}$$

Como se verificou, $T_{sinal} > T_s$, pelo que:

$$T_{pré\_sel} = T_{sinal} + T_{mux} = 2\lceil \log_2 w \rceil + 8 + 2 \tag{7.15a}$$

$$= 2\lceil \log_2 (\log_2 \ell + \log_2 h + 8) \rceil + 10 \tag{7.15b}$$

Considerando, uma vez mais, uma implementação com $\ell = h = 16$, obtém-se o valor $T_{pré\_sel} = 18$. Este valor é inferior ao valor obtido para o caminho crítico do elemento processador activo ($T = 23$, ver tabela 7.1), pelo que se conclui que este bloco não condiciona a máxima frequência de funcionamento do circuito.

Antes de terminar a descrição deste bloco convém, ainda, tecer algumas considerações a propósito das fases de inicialização destes circuitos de comparação. De facto, e considerando a utilização da medida de dissemelhança SAD, apenas se garante o correcto funcionamento deste circuito se o elemento de memória for inicializado com um valor muito elevado. Este procedimento foi realizado inicializando o elemento de memória com um vector composto por $w = \log_2 \ell + \log_2 h + 8$ bits inicializados a zero. Desta forma, como se pressupõe que o valor armazenado neste registo se encontra representado em complemento para um, o seu valor será, na realidade, o correspondente a:

$$\overline{MAX\_INT} = 1\underbrace{0000\ldots0}_{\log_2 \ell + \log_2 h + 8}, \tag{7.16}$$

que corresponde à representação em complemento para um do maior número positivo, representado em complemento para dois com os w bits a $'1'$.

7.5.2 Nível de selecção hierárquico

Este nível de selecção é o responsável pela escolha do macrobloco candidato com maior medida de similaridade, de entre o conjunto de macroblocos seleccionados no primeiro nível correspondentes a cada um dos blocos activos que compõem o processador.

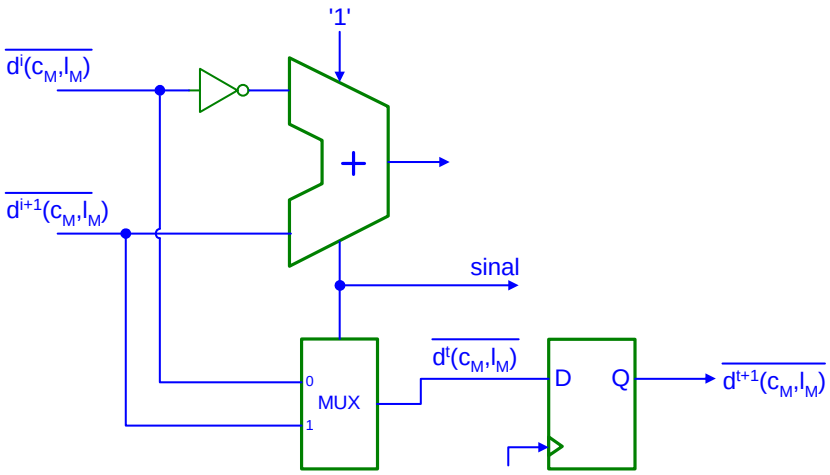


Figura 7.22: Estrutura interna de cada nó da árvore do nível de selecção hierárquico.

A arquitectura concebida para este bloco consiste numa estrutura em árvore binária com NC entradas, apresentando, à sua saída, o valor final correspondente ao macrobloco candidato com maior medida de similaridade. A estrutura adoptada neste bloco é em tudo idêntica à utilizada para o bloco somador em árvore, tendo-se distinguido, também, duas situações: *i)* o número de blocos activos (NC) corresponde a uma potência inteira de 2; *ii)* o valor de NC não é uma potência de 2. Assim, tal como se descreveu na secção 7.4, o projecto deste bloco é realizado de uma forma automática.

Cada um dos níveis desta estrutura é responsável pela comparação dos valores obtidos no nível anterior, seleccionando os valores correspondentes aos macroblocos candidatos com maior medida de similaridade, que são entregues ao nível seguinte. A estrutura interna de cada nó de comparação é a que se encontra representada na figura 7.22.

A propósito desta figura, convém lembrar que os valores da medida de similaridade obtidos à saída do primeiro nível de selecção encontram-se representados usando a notação em complemento para um, pelo que é necessário efectuar a inversão de todos os bits de um dos operandos por forma a realizar a operação de subtracção. O somador presente neste circuito é do tipo *Sklansky-lento*, por forma a garantir a frequência máxima de funcionamento do processador.

Antes de terminar a descrição deste bloco, convém ainda referir que, embora não tenha sido representado nas figuras anteriores, o sinal de controlo dos multiplexadores de selecção da medida de similaridade correspondente ao macrobloco candidato escolhido é também usado para seleccionar os valores correspondentes às coordenadas horizontal e vertical dos vectores de movimento correspondentes.

Refere-se ainda que, tal como se fez no bloco correspondente à matriz de proces-

samento e no bloco somador em árvore, utilizou-se também neste bloco um registo de deslocamento para garantir o sincronismo entre o processamento e os respectivos sinais de controlo.

7.6 Circuito de controlo

Para o funcionamento integrado de todos os blocos que constituem o processador de estimação de movimento agora proposto, foi necessário desenvolver um circuito de controlo que gerasse todos os sinais de comando necessários para garantir o fluxo de informação entre os vários blocos.

Para facilitar a concepção e o desenvolvimento deste controlador, dividiu-se o circuito de controlo em vários circuitos com funções localizadas e mais simples, cujo funcionamento é sincronizado através de um protocolo baseado na troca de alguns sinais de comando.

O circuito de controlo desenvolvido encontra-se dividido nos seguintes blocos controladores:

- Controlador principal, com funções de controlo sobre todas as operações do sistema;
- Controladores de entrada de dados, com funções de controlo sobre a entrada de dados da área de pesquisa e do macrobloco de referência; a interface destes controladores com o controlador principal baseia-se num sinal de disparo[2], que desencadeia o início da entrada de dados, e num sinal de paragem[3], que informa o controlador principal do seu estado de repouso.
- Gerador de sinais de relógio, com a função de gerar os sinais de relógio presentes no sistema.

De seguida, passar-se-á a descrever o funcionamento de cada um destes controladores.

7.6.1 Controlador principal

O controlador principal é o responsável pelo controlo de todos os blocos do sistema, gerando todo um conjunto de sinais que garantem a coordenação das tarefas realizadas nos diversos blocos. Conforme se ilustra na figura 7.23, este bloco é constituído pelos seguintes sub-blocos:

- Máquina de estados;
- Controlador de sentido de processamento;

[2]Do Inglês *Trigger*.
[3]Do Inglês *Idle*.

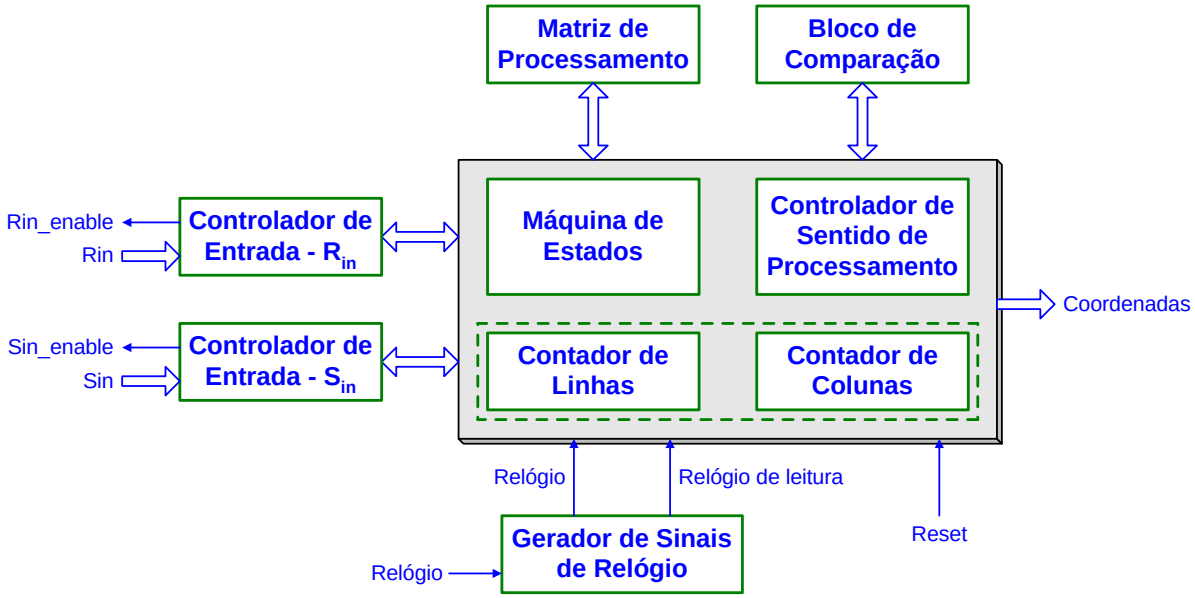


Figura 7.23: Constituição do controlador principal do sistema de controlo do processador.

- Contador de colunas;
- Contador de linhas.

Máquina de estados

A máquina de estados do sistema de controlo principal encontra-se representada na figura 7.24 e é constituída por 10 estados distintos, codificados através de 4 bits:

- E_R - Ciclo de inicialização;
- E_0, E_1 - Ciclo de enchimento dos registos de entrada de dados;
- E_2 - Transferência de dados para a matriz de processamento;
- E_3, E_4, E_5, E_6 - Ciclo de funcionamento em regime permanente;
- E_8, E_9 - Ciclo de espera pelos controladores de entrada de dados.

Trata-se de uma máquina de estados de Moore, implementada com *flip-flops* do tipo D, com o seguinte funcionamento:

– o estado E_R corresponde ao ciclo de inicialização, encontrando-se o controlador neste estado sempre que se activa o sinal de *reset*;

– após a inibição do sinal de *reset*, procede-se ao preenchimento dos registos de entrada, sendo feita a activação dos respectivos controladores no estado E_0, transitando, de seguida, para o estado E_1; a máquina permanecerá neste último estado

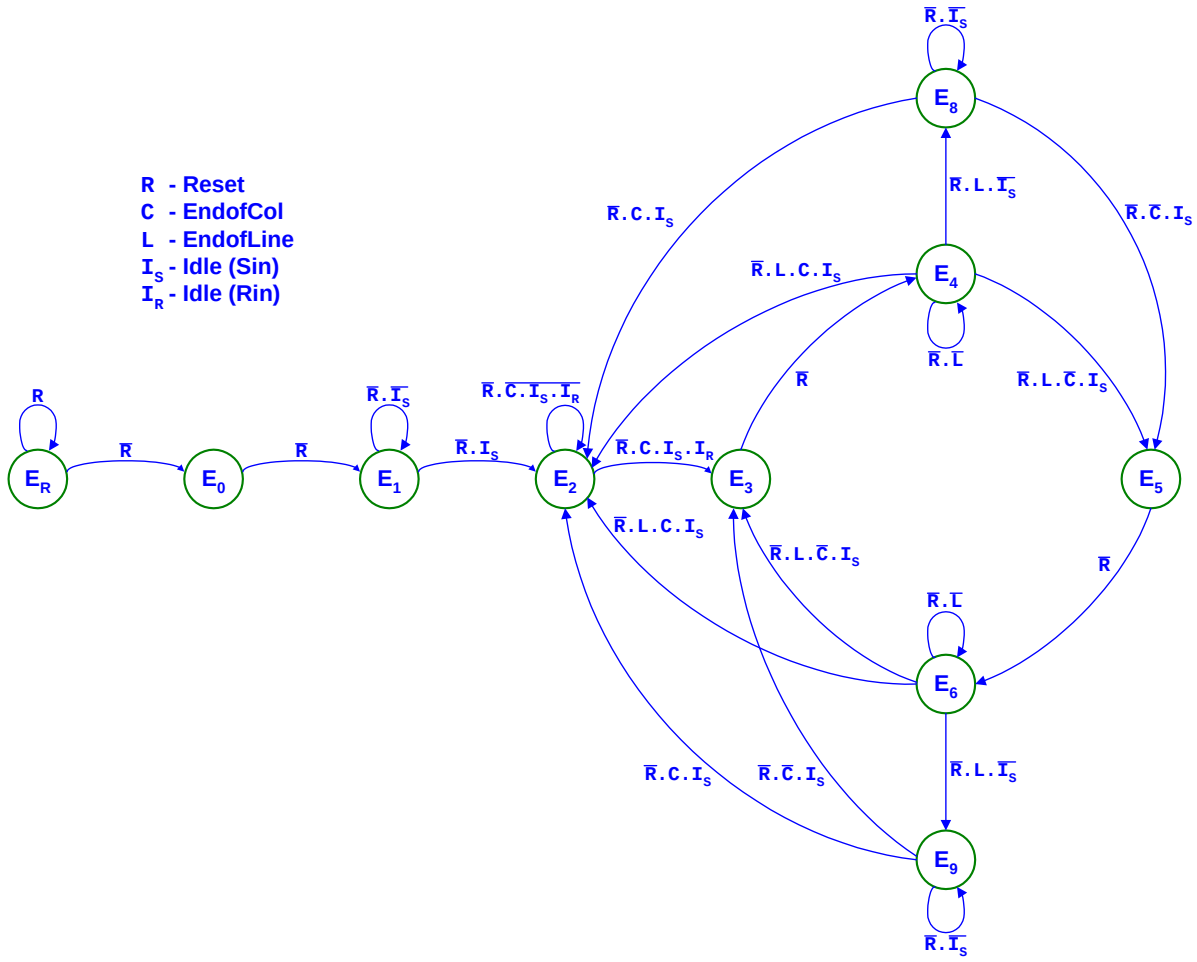


Figura 7.24: Máquina de estados do controlador principal.

até que o controlador do registo de entrada da área de pesquisa active o respectivo sinal de *idle*, sinalizando o término da operação de leitura de dados;

- os dados lidos pelo controlador de registos de entrada da área de pesquisa são, então, enviados no estado E_2 para a matriz de processamento, repetindo-se este processo h vezes até preencher completamente a matriz de processamento; o sistema aguarda ainda nesse estado que o controlador dos registos de entrada de dados do macrobloco de referência sinalize o término da sua operação, passando, depois, para o ciclo de funcionamento em regime permanente.
- o ciclo de funcionamento em regime permanente é composto por dois grupos de estados:
  - Nos estados E_3 e E_4 efectua-se o deslocamento dos pixels da área de pesquisa na matriz de processamento no sentido de rotação horário (ver figura 5.12); após a activação do sinal *EndofLine* pelo contador de colunas, sinalizando o término do processo de transferência de dados, o controlador activa, então, os

sinais de controlo da matriz de processamento, por forma a que os pixels da área de pesquisa sejam deslocados de uma linha;

- nos estados E_5 e E_6 o funcionamento do controlador é em tudo semelhante, correspondendo, no entanto, ao deslocamento horizontal dos pixels no sentido de rotação inverso; tal como no estado E_4, no estado E_6 é activado o sinal de controlo que desencadeia a transferência dos pixels para a linha superior logo que se verifique a activação do sinal *EndofLine*; nesta situação, a máquina de estados transita para o estado E_3 e o ciclo de funcionamento repete-se.

A excepção a este funcionamento cíclico verifica-se sempre que se dá o término da área de pesquisa atribuída ao macrobloco de referência, sinalizada pela activação do sinal *EndofCol* em simultâneo com o sinal *EndofLine* nos estados E_4 ou E_6. Nessa situação, o sistema transitará para o estado E_2, activando os sinais de disparo dos controladores dos registos de entrada.

- Os estados E_8 e E_9 correspondem a estados de espera da conclusão da leitura por parte do controlador de registos de entrada da área de pesquisa. O sistema transitará para um destes estados a partir dos estados E_4 ou E_6, respectivamente, sempre que os blocos activos terminam o processamento do macrobloco candidato localizado na extremidade da matriz de processamento (sinalizado pela activação do sinal *EndofLine*) e o controlador de registos de entrada da área de pesquisa não tenha, ainda, indicado o término da leitura de mais uma linha de dados, sinalizado através da activação do sinal *idle*.

Desta forma, verifica-se que o processador apresenta um funcionamento cíclico ao longo do processamento de cada macrobloco de referência, através dos estados E_3, E_4, E_5, E_6, E_8 e E_9. A excepção a este funcionamento dá-se, apenas, aquando da mudança de macrobloco de referência, onde se verifica a inclusão do estado E_2 neste ciclo de funcionamento.

Uma outra situação extraordinária verifica-se, também, sempre que se dá a activação do sinal de *reset*, situação em que o controlador transitará para o estado E_R independentemente do estado em que ele se encontre.

A máquina apresenta um total de 10 saídas principais, correspondentes a cada um dos estados. Consequentemente, optou-se por atribuir, a cada uma delas, a designação do estado respectivo E_x. Estas saídas serão, então, utilizadas para definir os restantes sinais de controlo do circuito.

Controlador de sentido de processamento

O bloco controlador de sentido de processamento é o responsável pelo controlo do sentido de deslocamento dos pixels da área de pesquisa na matriz de processamento, através do sinal *LeftnRight*. Sempre que este sinal se encontra no nível $'1'$ o deslocamento dos pixels é feito no sentido de rotação horário, tomando o sentido inverso quando este sinal apresenta o nível $'0'$.

Conforme se verifica na figura 7.25, o controlador do sentido de rotação é implementado através de um circuito bastante simples, constituído por uma máquina de estados com apenas dois estados e controlado pelo sinal *LeftnRightEn* (entrada de *enable*), obtido através da seguinte expressão:

$$LeftnRightEn = R + [(E_4 + E_6)\,.L.I_s] + (E_8 + E_9) \tag{7.17}$$

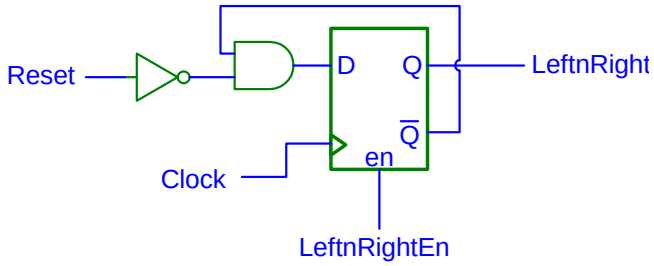


Figura 7.25: Controlador do sentido de rotação.

Verifica-se, assim, que a mudança de estado dar-se-á apenas nos estados E_8 e E_9, e nos estados E_4 e E_6 sempre que se verifique o fim do curso de deslocamento horizontal (*EndofLine*) e o término da leitura de uma nova linha da área de pesquisa, por parte do circuito controlador dos registos de entrada.

Contador de colunas

O bloco contador de colunas é o bloco responsável pela indicação das coordenadas horizontais dos vectores de movimento correspondentes aos macroblocos candidatos considerados na pesquisa (sinal *ColNr*). Para além disso, efectua também o controlo do fim do curso de deslocamento na matriz de processamento dos pixels da área de pesquisa, através da geração do sinal *EndofLine*.

Conforme se pode verificar na figura 7.26, este contador suporta contagens crescentes ou decrescentes, dependendo do sinal de controlo $D/\overline{U}$, sinal este que é, por sua vez, obtido a partir do sistema de controlo de sentido de processamento descrito anteriormente (sinal *LeftnRight*).

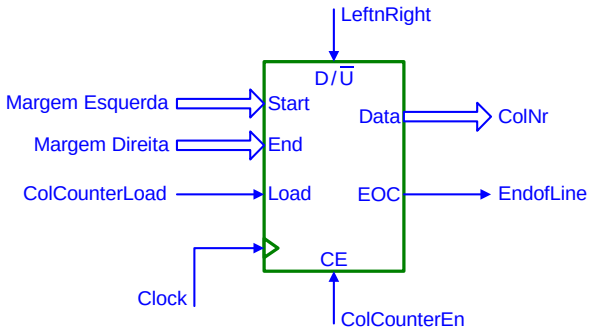


Figura 7.26: Contador de colunas do controlador principal de processamento.

Os limites do curso de contagem são controlados pelos vectores introduzidos nas entradas *Start* e *End*, correspondentes às coordenadas do macrobloco candidato situado nas margens esquerda e direita da área de pesquisa, respectivamente. O valor destas coordenadas é dado pelas seguintes expressões (ver figuras 5.14 e 5.16):

$$Margem\ Esquerda = -(p-1) \tag{7.18}$$

$$Margem\ Direita = -(p-1) + T - 1 \tag{7.19}$$

onde p é o parâmetro que define a dimensão da área de pesquisa considerada e T é o número de macroblocos candidatos processados por cada bloco activo, dado pela equação 5.6.

Para além destes sinais, o funcionamento deste controlador é definido pelos sinais de carregamento[4] e de activação[5] do contador, dados pelas seguintes expressões:

$$ColCounterLoad = R + E_R + E_1 \tag{7.20}$$

$$ColCounterEn = \overline{R}.\left[(E_3 + E_5) + \left[(E_4 + E_6)\,.\,\left(\overline{L} + L.I_S\right)\right] + (E_8 + E_9)\right] \tag{7.21}$$

Assim, o carregamento do contador é realizado apenas numa situação de activação do sinal de *reset* e durante os estados iniciais da máquina de estados.

Em relação ao sinal de activação, verifica-se que o contador deve encontrar-se activo durante todo o ciclo de funcionamento em regime permanente, composto pelos estados E_3, E_4, E_5, E_6, E_8 e E_9. A única excepção verifica-se na situação particular em que o circuito se encontra nos estados E_4 ou E_6 e, embora o processador tenha chegado ao fim do curso no deslocamento horizontal efectuado, o controlador de entrada de dados não terminou, ainda, a sua operação. Esta situação corresponde ao ciclo de relógio em que o sistema transita dos estados E_4 ou E_6, para os estados E_8 ou E_9, respectivamente (ver figura 7.24).

[4] Do Inglês *Load*.
[5] Do Inglês *Enable*.

Antes de terminar a descrição deste bloco, convém ainda referir a necessidade de utilizar um somador para efectuar o cálculo da coordenada horizontal efectiva dos macroblocos candidatos processados pelos vários blocos activos. De facto, verifica-se que o valor do sinal *ColNr* corresponde, apenas, às coordenadas dos vectores de movimento do bloco activo situado na extremidade esquerda da matriz de processamento. Por forma a obter o valor efectivo das coordenadas correspondentes aos restantes vectores de movimento torna-se necessário somar, a esta variável, o valor de *offset* correspondente, sendo dado por:

$$ColNr(c) = ColNr + offset(c)\ , \quad c = 0 \ldots (NC-1) \tag{7.22}$$

$$offset(c) = c \cdot T, \tag{7.23}$$

em que T é, uma vez mais, dado pela equação 5.6.

Contador de linhas

O bloco contador de linhas é o bloco responsável por três funções distintas:

- controlo do carregamento inicial, durante o estado $\mathsf{E_2}$, da matriz de processamento com os pixels correspondentes às h primeiras linhas da área de pesquisa;
- controlo do ciclo de funcionamento em regime permanente da máquina de estados, correspondente aos estados $\mathsf{E_3}$, $\mathsf{E_4}$, $\mathsf{E_5}$, $\mathsf{E_6}$, $\mathsf{E_8}$ e $\mathsf{E_9}$, através do sinal de sinalização *EndofCol*;
- geração das coordenadas verticais dos vectores de movimento durante o ciclo de funcionamento em regime permanente.

Conforme se pode verificar na figura 7.27, este sistema de controlo é constituído por um contador com entradas de início e fim do curso de contagem. Estas entradas dependem do tipo de funções executadas pelo circuito, conforme se indica nas expressões seguintes:

$$\begin{cases} Limite\ Inf_0 = 0 \\ Limite\ Sup_0 = h-2 \end{cases} \qquad \begin{cases} Limite\ Inf_1 = -(p-1) \\ Limite\ Sup_1 = p \end{cases} \tag{7.24}$$

onde p é, uma vez mais, o parâmetro que define a dimensão da área de pesquisa e h define o número de linhas de elementos processadores da matriz de processamento.

A escolha destes limites é feita por intermédio de dois multiplexadores, actuados por um sinal de selecção definido pela seguinte expressão:

$$Sel = \overline{R}.\left[(E_2.C) + E_3 + E_5 + \left[(E_4 + E_6 + E_8 + E_9).\overline{C}.\overline{L}\right]\right] \tag{7.25}$$

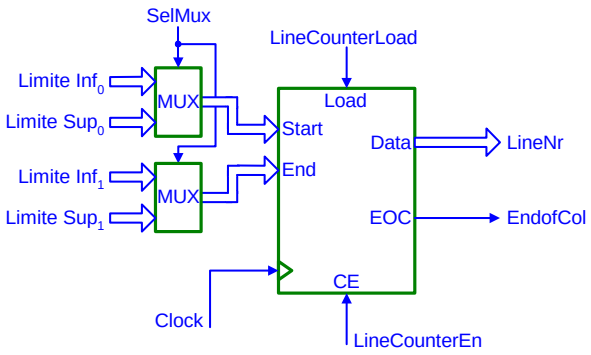


Figura 7.27: Contador de linhas do controlador principal de processamento.

Assim, durante a fase de transferência de dados para a matriz de processamento, correspondente ao estado E_2, o circuito efectua a contagem de $h-1$ linhas que, juntamente com a linha carregada previamente nos estados E_0 e E_1, perfaz as h linhas de elementos processadores da matriz de processamento.

Durante o ciclo de funcionamento em regime permanente, o circuito executa, então, a contagem de $2p$ linhas, correspondentes ao número de macroblocos candidatos considerados ao longo da direcção vertical da área de pesquisa, conforme se descreveu na secção 5.4 e se ilustrou na figura 5.14.

Para além destes sinais, o funcionamento deste controlador é definido pelos sinais de carregamento (*Load*) e de activação (*Enable*), dados pelas seguintes expressões:

$$LineCounterLoad = R + E_R \tag{7.26}$$

$$LineCounterEn = \overline{R}.\left[(E_2.I) + \left[(E_4 + E_6)\,.L.I\right] + (E_8 + E_9)\right] \tag{7.27}$$

O sinal de carregamento encontra-se activo apenas durante a fase de inicialização do processador. O contador encontra-se no estado activo durante a fase de pré-carregamento da matriz de processamento no estado E_2 (sempre que o controlador do registo de entrada sinaliza o fim da leitura de uma linha de pixels) e durante a fase correspondente ao ciclo de funcionamento em regime permanente da máquina de estados.

O valor obtido à saída deste contador indica, à semelhança do contador de colunas, o valor da coordenada vertical do vector de movimento correspondente aos macroblocos candidatos sob processamento nos blocos activos do processador.

7.6.2 Controlador de entrada de dados

Conforme se ilustrou na figura 7.23, o circuito de entrada de dados é constituído por dois blocos de controlo distintos:

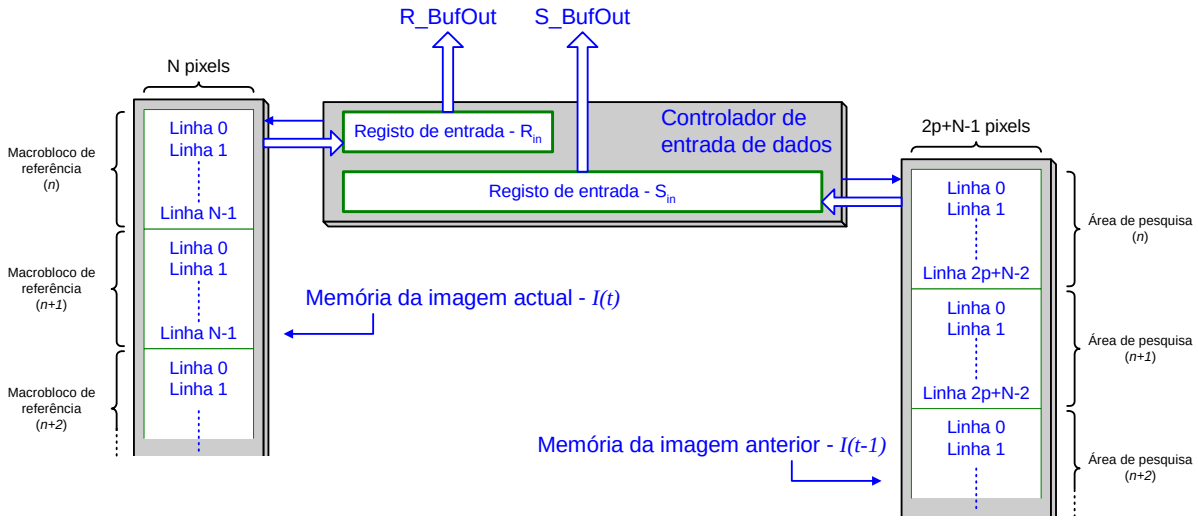


Figura 7.28: Interface do controlador de entrada de dados com o banco de memórias de vídeo.

- controlador de entrada do macrobloco de referência - R_{in}, para a leitura e transferência dos pixels do macrobloco de referência para os blocos activos da matriz de processamento;
- controlador de entrada da área de pesquisa - S_{in}, que controla a leitura dos pixels correspondentes à área de pesquisa, e a sua transferência para os elementos processadores da linha inferior da matriz de processamento.

A interface destes dois blocos de controlo com o banco de memórias de vídeo encontra-se ilustrada na figura 7.28. De seguida, passar-se-á a descrever o funcionamento de cada um destes blocos controladores.

Controlador de entrada do macrobloco de referência

O controlador de entrada do macrobloco de referência tem, como se referiu, a função de ler os pixels do macrobloco de referência e de transferi-los para os blocos activos da matriz de processamento. Esta tarefa é desencadeada pelo controlador principal, através da actuação do sinal *trig* (obtido a partir das saídas da sua máquina de estados). De seguida, este controlador de entrada activa o sinal Rin_Enable, conduzindo à activação da memória correspondente à imagem actual (ver figura 7.28), permitindo a leitura dos N^2 pixels correspondentes a um macrobloco de referência.

Na figura 7.29 apresenta-se o diagrama de blocos deste controlador, fazendo-se, de seguida, a descrição de cada um dos seus blocos.

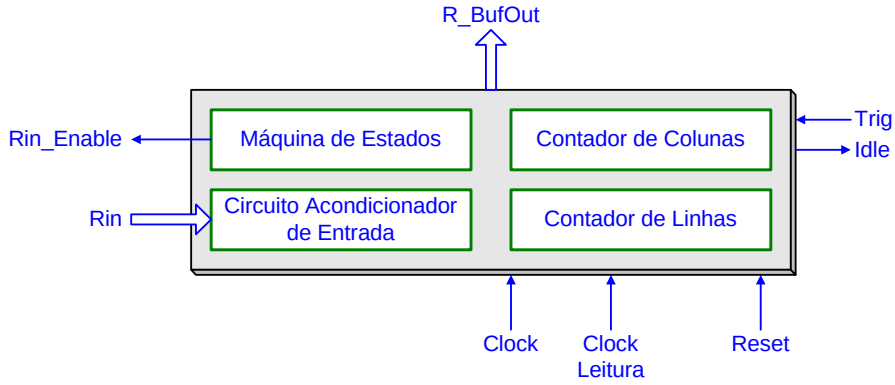


Figura 7.29: Constituição interna do controlador de entrada do macrobloco de referência.

Máquina de estados

A máquina de estados deste controlador encontra-se representada na figura 7.30 e é constituída pelos seguintes cinco estados:

- E_R - estado de inicialização;
- E_1 - ciclo de enchimento do circuito acondicionador de entrada;
- E_2 - estado de transferência de dados para a matriz de processamento;
- E_4, E_5 - ciclos de sinalização.

O funcionamento desta máquina de estados é o seguinte:

– o estado E_R corresponde ao ciclo de inicialização, encontrando-se o controlador neste

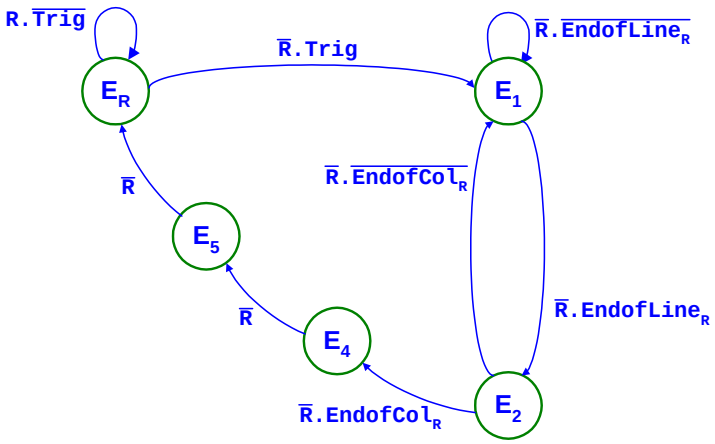


Figura 7.30: Máquina de estados do controlador de entrada do macrobloco de referência.

estado sempre que o sinal de *reset* é activado, e até que seja actuada a sua entrada de início de funcionamento (*trig*) pelo controlador principal;

- após a activação da entrada *trig*, procede-se, no estado E_1, à leitura de uma linha composta por N pixels do macrobloco de referência;
- terminada a leitura da linha de pixels, sinalizada pelo sinal $EndofLine_R$, o circuito transita para o estado E_2, onde efectua a transferência em paralelo de todos os N pixels para os blocos activos da matriz de processamento;
- de seguida, dependendo da activação ou não do sinal $EndofCol_R$, sinalizando o término da leitura do macrobloco de referência, após ter efectuado a leitura de N linhas de pixels, a máquina transitará para o estado E_4 ou regressará ao estado E_2, repetindo a operação de leitura dos pixels;
- caso o sistema transite para o estado E_4, ele será então conduzido ao estado de inicialização E_R, passando, entretanto, pelo estado E_5, onde serão activados alguns sinais de controlo do circuito.

Assim, o funcionamento do circuito é, essencialmente, desencadeado pelo sinal de activação *trig*. Passa, repetitivamente pelo conjunto de estados E_1 e E_2 e permanece, depois, num estado de repouso à espera de nova ordem de execução.

Circuito acondicionador de entrada

O circuito acondicionador de entrada tem a função de ler os pixels de uma forma sequencial que são, depois, entregues aos blocos activos da matriz de processamento sob a forma de um vector composto por N pixels. Tem, assim, a função de converter o formato dos dados de série para paralelo.

Para o desenvolvimento deste circuito, adoptou-se uma estrutura com N registos ligados em série, com as respectivas saídas acessíveis do exterior, conforme se apresenta na figura 7.31. Um dado ciclo de enchimento deste tipo de estrutura termina, apenas, após a entrega, à matriz de processamento, de todos os N pixels previamente lidos.

Contador de colunas

O circuito contador de colunas é o responsável pelo controlo do número de pixels lido pelo circuito acondicionador de entrada, activando a sua saída de $EndofLine_R$ sempre que termina um ciclo de enchimento.

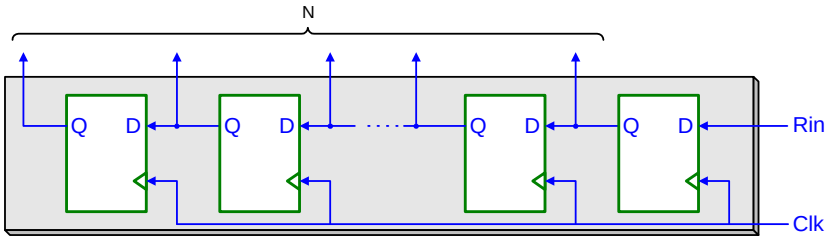


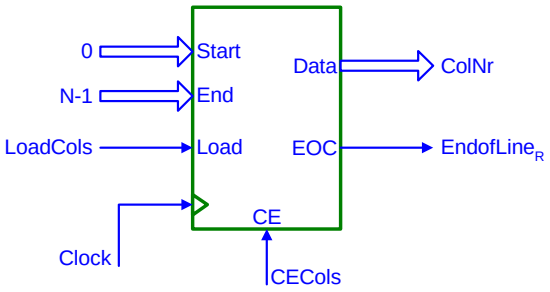
Figura 7.31: Circuito acondicionador de entrada dos pixels do macrobloco de referência.

Conforme se apresenta na figura 7.32(a), utiliza-se um circuito semelhante ao da figura 7.26. Este circuito é constituído por um contador com entradas de início e fim do curso de contagem, que tomam, neste caso, os valores 0 e $N-1$, respectivamente. Para além destas entradas, o circuito apresenta, ainda, entradas de controlo de carregamento (*Load*) e de activação (*Enable*), dadas pelas seguintes expressões:

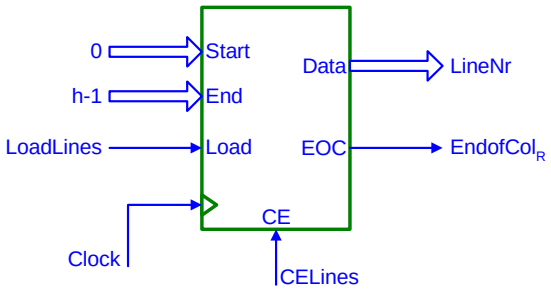
$$LoadCols = R + \overline{E_1} \tag{7.28}$$

$$CECols = E_1 \cdot \overline{EndofLine_R} \tag{7.29}$$

Verifica-se assim que, à excepção do estado E_1, o circuito apresenta sempre a sua entrada de carregamento activa, encontrando-se, assim, pronto para iniciar uma nova contagem. Esta contagem é activada durante o estado E_1 apenas quando a saída $EndofLine_R$ não se encontre activa.



(a) Contador de colunas.



(b) Contador de linhas

Figura 7.32: Contadores de colunas e de linhas do controlador de entrada do macrobloco de referência.

Contador de linhas

O circuito contador de linhas é o responsável pelo controlo do número de linhas lido pelo circuito acondicionador de entrada, por forma a preencher as h linhas de elementos processadores dos blocos activos.

Conforme se ilustra na figura 7.32(b), este circuito é, tal como o circuito controlador de colunas, constituído por um contador com entradas de início e fim do curso de contagem. Neste caso, o circuito deverá começar a contar no valor 0 e deverá terminar no valor $h-1$.

O circuito apresenta, ainda, entradas de controlo de carregamento (*Load*) e de activação (*Enable*), dadas pelas seguintes expressões:

$$LoadLines = R + E_R \tag{7.30}$$

$$CELines = E_2 \cdot \overline{EndofCol_R} \cdot \overline{EndofLine_R} \tag{7.31}$$

Verifica-se, assim, que o circuito efectua a inicialização da contagem sempre que se encontra no seu estado de repouso $\mathsf{E_R}$. Para além disso, o valor da saída deverá ser incrementado sempre que o controlador passe pelo estado $\mathsf{E_2}$ e não se verifique o enchimento do circuito acondicionador de entrada nem o fim do curso de contagem do próprio contador.

Controlador de entrada da área de pesquisa

Este controlador de entrada é responsável pela leitura de uma linha de pixels da área de pesquisa e pela sua transferência para a matriz de processamento. Esta tarefa é desencadeada pelo controlador principal através da activação da entrada *trig*, dando-se assim início à operação de leitura. Nesta operação, o controlador de entrada começa por activar o sinal *Sin_Enable*, activando assim o porto de saída da memória onde se encontra a imagem anterior (ver figura 7.28), lendo, de seguida, os L' pixels correspondentes a uma linha da área de pesquisa.

Na figura 7.33 apresenta-se o diagrama de blocos deste controlador, fazendo-se, de seguida, a descrição de cada um dos seus blocos.

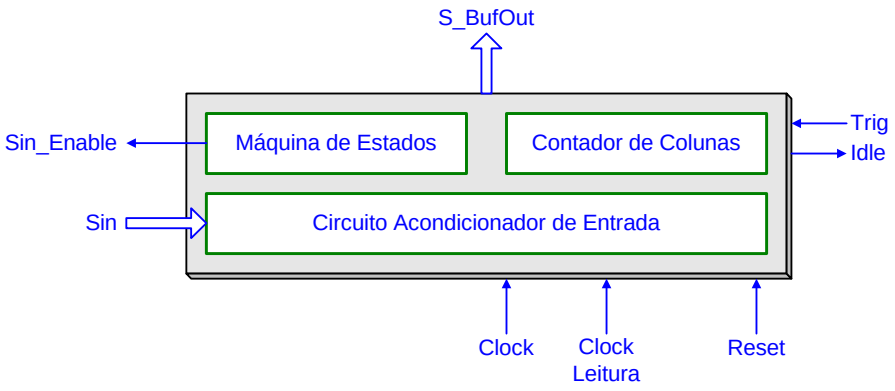


Figura 7.33: Constituição interna do controlador de entrada da área de pesquisa.

Máquina de estados

A máquina de estados deste controlador encontra-se representada na figura 7.34(a) e é constituída pelos seguintes dois estados:

- E_R - estado de inicialização;
- E_1 - ciclo de leitura.

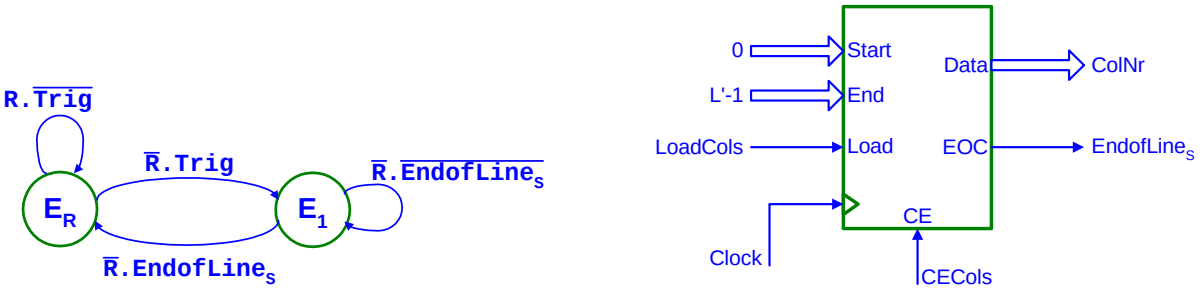
O funcionamento desta máquina de estados é o seguinte:

– o estado E_R corresponde ao ciclo de inicialização, encontrando-se o controlador neste estado sempre que o sinal de *reset* é activado e até que seja actuada a entrada de activação (*trig*);

– após a activação da entrada *trig* procede-se, no estado E_1, à leitura de uma linha composta por L' pixels da área de pesquisa;

– terminada a leitura da linha de pixels, sinalizada pelo sinal $EndofLine_S$, o circuito transita novamente para o estado de inicialização E_R, ficando a aguardar uma nova activação do sinal *trig*.

Assim, e tal como nos circuitos de controlo anteriormente descritos, verifica-se que o funcionamento do circuito é, essencialmente, desencadeado pelo sinal de activação *trig*, permanecendo depois num estado de repouso a aguardar por nova ordem de execução.



(a) Máquina de estados. **(b)** Contador de colunas.

Figura 7.34: Máquina de estados e contador de colunas do controlador de entrada dos pixels da área de pesquisa.

Contador de colunas

O circuito contador de colunas é o responsável pela contabilização do número de pixels lido pelo circuito acondicionador de entrada, activando o sinal $EndofLine_S$ sempre que o circuito acondicionador de entrada se encontre totalmente preenchido.

Conforme se ilustra na figura 7.34(b), este circuito é constituído por um contador com entradas de início e fim do curso de contagem, tomando, neste caso, os valores 0 e $L'-1$, respectivamente.

Para além destas entradas, o circuito apresenta, ainda, entradas de controlo de carregamento (*Load*) e de activação (*Enable*), dadas pelas seguintes expressões:

$$LoadCols = R + E_R \tag{7.32}$$

$$CECols = E_1 \cdot \overline{EndofLine_S} \tag{7.33}$$

Verifica-se, assim, que o circuito executa a inicialização do valor da contagem no estado $\mathsf{E_R}$ e sempre que se active o sinal de *reset*. A contagem encontra-se activa no estado $\mathsf{E_1}$ até ao instante em que a saída $EndofLine_S$ é activada, o que indica que o circuito acondicionador de entrada se encontra preenchido.

Circuito acondicionador de entrada

O circuito acondicionador de entrada lê, de uma forma sequencial, os pixels da área de pesquisa (ver figura 7.28), e transfere-os, em paralelo, para a matriz de processamento aquando da inversão do sentido de deslocamento dos pixels da área de pesquisa. Esta tarefa é executada em fracções, correspondentes a cada uma das linhas da área de pesquisa, constituída por L' pixels. Tal como no controlador de entrada do macrobloco de referência, este bloco tem, assim, uma função de conversão de formato do tipo série em paralelo.

À semelhança do controlador anteriormente descrito, para o desenvolvimento deste circuito adoptou-se uma estrutura com L' registos ligados em série com as respectivas saídas acessíveis do exterior, conforme se apresenta na figura 7.35.

O esquema de processamento que se adopta, baseado numa mobilidade dos pixels da área de pesquisa sobre uma estrutura do tipo cilíndrica, complica, ligeiramente, a configuração do circuito de acondicionamento de entrada. Para analisar este facto considere-se, a título de exemplo, a matriz de processamento ilustrada na figura 7.36(a), correspondente a uma implementação do processador com dois blocos activos ($NC = 2$) e com $N = 4$ e $p = 7$. Para facilitar a exposição, representou-se, na figura 7.36(b), esta matriz

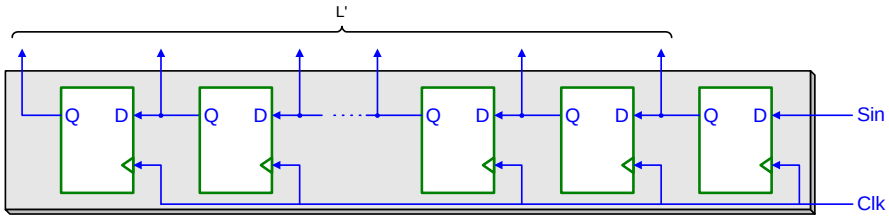


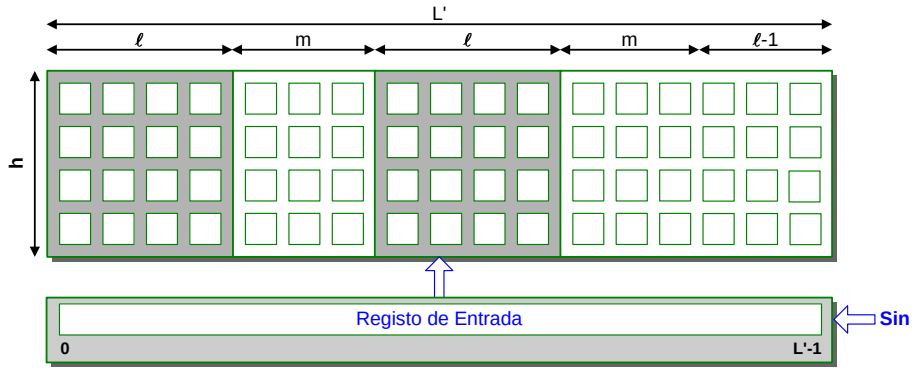
Figura 7.35: Circuito acondicionador de entrada dos pixels da área de pesquisa.

de processamento de uma forma esquemática, numerando cada uma das colunas processadas por cada um dos blocos activos A e B. Conforme seria de esperar, algumas destas colunas são processadas por mais do que um bloco activo, pelo que se apresentam ambas as numerações.

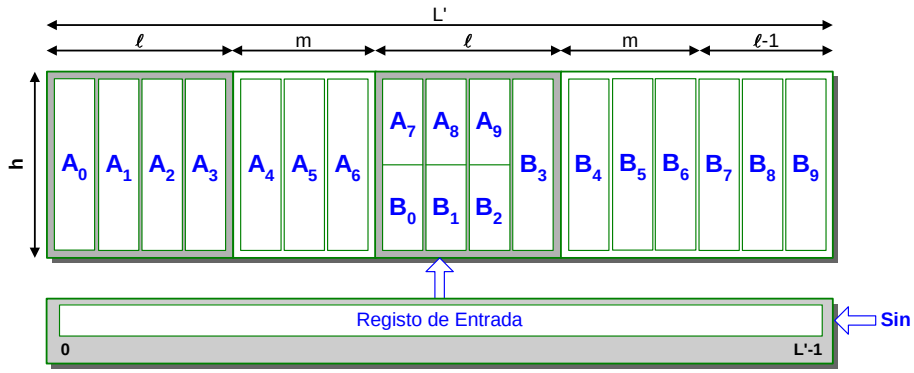
A situação ilustrada na figura 7.36(b) corresponde a uma das situações extremas de entrada de dados, em que, neste caso, a disposição dos pixels da área de pesquisa na matriz corresponde à disposição natural dos mesmos na imagem. Desta forma, há apenas a necessidade de transferir, em paralelo, todos os L' pixels armazenados no registo de entrada para a matriz de processamento.

Contudo, na outra situação extrema de entrada de dados, correspondente a um deslocamento completo dos pixels da área de pesquisa na matriz de processamento, a disposição espacial dos pixels no registo de entrada encontra-se desalinhada relativamente à disposição dos pixels na matriz de processamento. Na realidade, o seu alinhamento é garantido apenas se a entrada dos pixels no agregado de registos em série do circuito acondicionador de entrada for feita no ponto assinalado na figura 7.36(c), interligando-se, assim, os dois registos localizados nas extremidades esquerda e direita. Esta configuração baseia-se na partição do agregado de L' registos ligados em série em dois agregados de menor dimensão: um com $NC.(\ell + m) - m$ registos e outro com $m + \ell - 1$ registos.

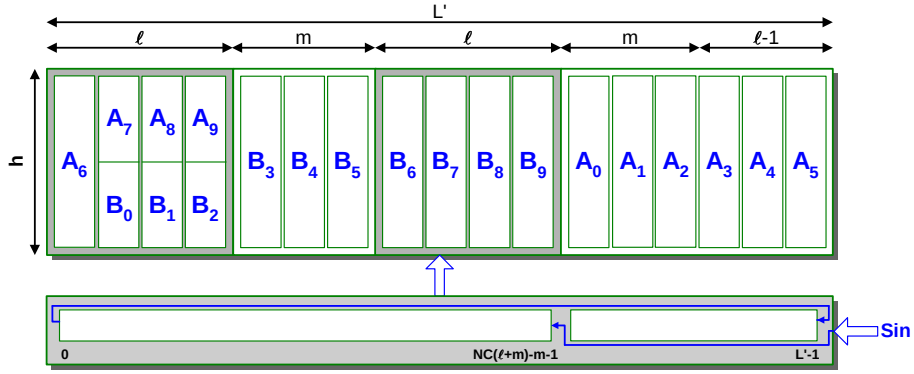
Na figura 7.37 apresenta-se o circuito de alinhamento projectado. Neste caso, a configuração das ligações existentes no agregado de registos é variável ao longo do tempo, dependente do sentido de deslocamento dos pixels da área de pesquisa. Este mecanismo foi conseguido através da utilização de multiplexadores, conforme se ilustra nesta figura. Torna-se assim possível efectuar a transferência de todos os L' pixels em paralelo para a área desejada da matriz de processamento.



(a) Implementação considerada ($NC = 2$, $N = h = 4$ e $p = 7$).



(b) Situação correspondente a um alinhamento espacial entre a disposição dos pixels no registo de entrada e na matriz de processamento.



(c) Situação correspondente a um desalinhamento espacial entre a disposição dos pixels no registo de entrada e na matriz de processamento não é garantido.

Figura 7.36: Circuito acondicionador de entrada dos pixels da área de pesquisa.

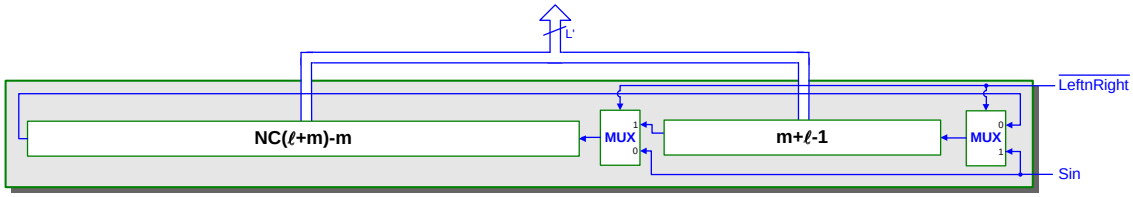


Figura 7.37: Estrutura do circuito condicionador de entrada dos pixels da área de pesquisa.

7.6.3 Gerador de sinais de relógio

Um dos objectivos fundamentais no projecto e desenvolvimento do processador para estimação de movimento proposto nesta dissertação foi o de minimizar o tempo de processamento. Tal objectivo foi atingido desenvolvendo circuitos que aumentam o nível de paralelismo explorado e cancelam os ciclos de relógio desnecessários para efectuar o preenchimento da matriz de processamento.

Contudo, o cumprimento deste objectivo levanta um outro problema relacionado com a leitura dos pixels. De facto, conforme se descreveu anteriormente, é necessário um total de $V = NC.\left\lfloor \frac{2p}{NC} \right\rfloor$ ciclos de relógio para processar cada uma das linhas da matriz de processamento, enquanto para efectuar a leitura dos pixels da área de pesquisa necessários para preencher uma nova linha da matriz de processamento são necessários $L' + 3$ ciclos de relógio[6].

Uma forma possível de resolver este problema consiste na utilização de várias entradas para leitura dos pixels da área de pesquisa, levando à utilização de vários dispositivos de memória para armazenar a imagem anterior, que é, na realidade, a solução adoptada por vários autores [9, 23, 20]. Contudo, esta solução é onerosa, pois aumenta o número de dispositivos de memória requeridos, bem como o número de portos do processador.

A solução proposta neste trabalho baseia-se no facto de, em geral, os dispositivos de memória apresentarem ciclos de leitura bastante mais rápidos do que o tempo correspondente ao caminho crítico deste processador. Assim, optou-se por utilizar dois sinais de relógio distintos neste processador:

- Relógio de leitura, utilizado para ler os pixels do exterior de uma forma expedita, com base nos circuitos controladores de entrada de dados;
- Relógio de processamento, utilizado para realizar o restante processamento.

O relógio de processamento é obtido a partir do relógio de leitura, por intermédio de um circuito de divisão da frequência por um determinado factor. Consequentemente, o

[6]Num dos ciclos de relógio actua-se o sinal *trig* e noutro actua-se o sinal *idle*.

processador proposto apresenta uma única entrada de relógio, correspondente ao relógio de leitura com a frequência mais elevada.

Para garantir o correcto funcionamento do circuito é necessário que:

$$t_{proc} \geq t_{leitura} \tag{7.34a}$$

$$\Leftrightarrow \quad \#ciclos_{proc} \times T_{proc} \geq \#ciclos_{leitura} \times T_{leitura} \tag{7.34b}$$

em que t_{proc} é o tempo de processamento de uma linha da área de pesquisa e $t_{leitura}$ é o tempo de leitura de uma nova linha da área de pesquisa.

Considerando que $f_{proc} = \frac{1}{T_{proc}} = \frac{f_{leitura}}{\alpha} \Leftrightarrow T_{leitura} = \frac{T_{proc}}{\alpha}$, obtém-se de 7.34a:

$$\#ciclos_{proc} \geq \frac{\#ciclos_{leitura}}{\alpha} \tag{7.35}$$

Considerando $\#ciclos_{proc} = NC.\left\lfloor \frac{2p}{NC} \right\rfloor$ e $\#ciclos_{leitura} = L' + 3$ vem que:

$$NC.\left\lfloor \frac{2p}{NC} \right\rfloor \geq \frac{L'+3}{\alpha} \tag{7.36}$$

Atendendo à equação 5.9, obtém-se:

$$NC.\left\lfloor \frac{2p}{NC} \right\rfloor \geq \frac{NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + (\ell - 1) + 3}{\alpha} \tag{7.37a}$$

$$\Leftrightarrow \quad \alpha \geq \frac{NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor + \ell + 2}{NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor} \tag{7.37b}$$

Como $a \geq k \left\lfloor \frac{a}{k} \right\rfloor$ (para $k, a \in \mathcal{N}$) vem que:

$$\alpha \geq 1 + \frac{\ell + 2}{NC \cdot \left\lfloor \frac{2p}{NC} \right\rfloor} \geq 1 + \frac{\ell + 2}{2p} \tag{7.38}$$

Assim, considerando apenas factores de divisão de frequência correspondentes a valores inteiros, conclui-se que a relação entre as frequências dos relógios utilizados neste processador deve ser a seguinte:

$$f_{proc} = \frac{f_{leitura}}{\alpha} \ , \qquad \alpha = \left\lceil 1 + \frac{\ell + 2}{2p} \right\rceil \tag{7.39}$$

Em virtude do circuito de geração de sinais de relógio funcionar a uma frequência mais elevada do que os restantes elementos do processador, optou-se por utilizar um circuito de divisão de frequência que usasse a menor quantidade de circuitos de lógica combinatória possível. Por isso, optou-se por implementar este circuito recorrendo a um contador do tipo *ring-counter* [45], composto por α *flip-flops*. Na figura 7.38 apresenta-se um circuito deste tipo para o caso particular de uma configuração com $\alpha = 4$.

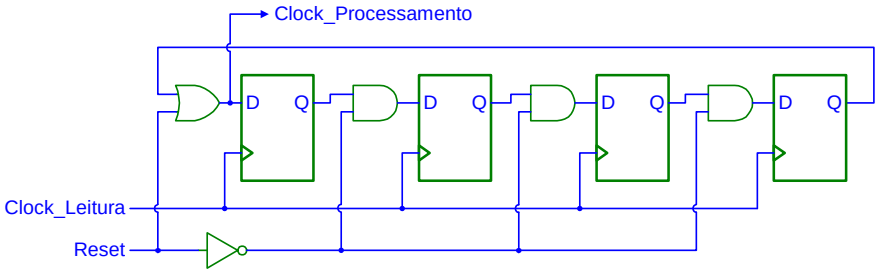


Figura 7.38: Estrutura do circuito de divisão de frequência para gerar o sinal de relógio de processamento.

7.7 Integração do processador num sistema de codificação de vídeo

Conforme se referiu na introdução, pretendeu-se com este trabalho de investigação desenvolver um processador de estimação de movimento de alta eficiência, com vista à sua integração em sistemas de codificação de vídeo de uso geral, por forma a acelerar os procedimentos de estimação e compensação de movimento (ver figura 1.4).

Para além disso, outro dos objectivos deste trabalho foi simplificar, o mais possível, a interface de controlo e de dados entre o processador desenvolvido e o sistema de codificação, facilitando, assim, a sua integração em sistemas de codificação de vídeo.

O processador desenvolvido apresenta, apenas, quatro portos de entrada e quatro portos de saída, conforme se ilustra na figura 7.39.

Os portos de entrada são os seguintes:

- Rin [7..0] - porto de entrada de dados correspondentes aos pixels do macrobloco de referência;
- Sin [7..0] - porto de entrada de dados correspondentes aos pixels da área de pesquisa;
- Clock - sinal de relógio (de leitura);
- Reset - sinal de inicialização.

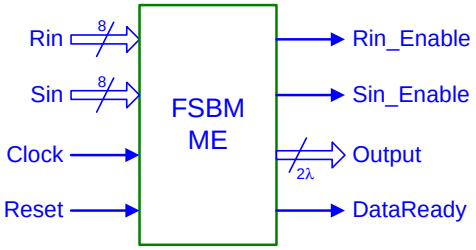


Figura 7.39: Portos de entrada e de saída do processador desenvolvido ($\lambda = \log_2(p+1)+1$).

Os portos de saída são os seguintes:

- Rin_Enable - sinal de activação da memória de imagem actual $I(t)$;
- Sin_Enable - sinal de activação da memória de imagem anterior $I(t-1)$;
- Output$\{[(\lambda-1)..0]\,;[(\lambda-1)..0]\}$ - sinal correspondente ao par de coordenadas do vector de movimento calculado $(\lambda = \log_2(p+1)+1)$;
- DataReady - sinal que indica, ao sistema de codificação de vídeo, o término do processamento do macrobloco candidato sob consideração.

Na figura 7.40 apresenta-se o diagrama de blocos de um sistema de codificação de vídeo que integra o processador desenvolvido. Nesta figura, pode observar-se a interacção do processador de estimação com o sistema de compensação de movimento e com o banco de memórias correspondentes às imagens actual e anterior. Para facilitar a descrição e a representação, optou-se por não se representar o bloco de controlo global do sistema, pelo que se ligaram todos os sinais de controlo do processador directamente aos blocos do sistema de codificação de vídeo.

Por fim, representa-se na figura 7.41 um diagrama temporal dos vários sinais de interface do processador referentes ao processo de estimação de movimento para um dado macrobloco candidato. Conforme se pode observar, a memória correspondente à imagem actual é apenas lida durante a fase inicial do processamento, correspondente ao período de escrita dos respectivos pixels nos blocos activos da matriz de processamento. Este

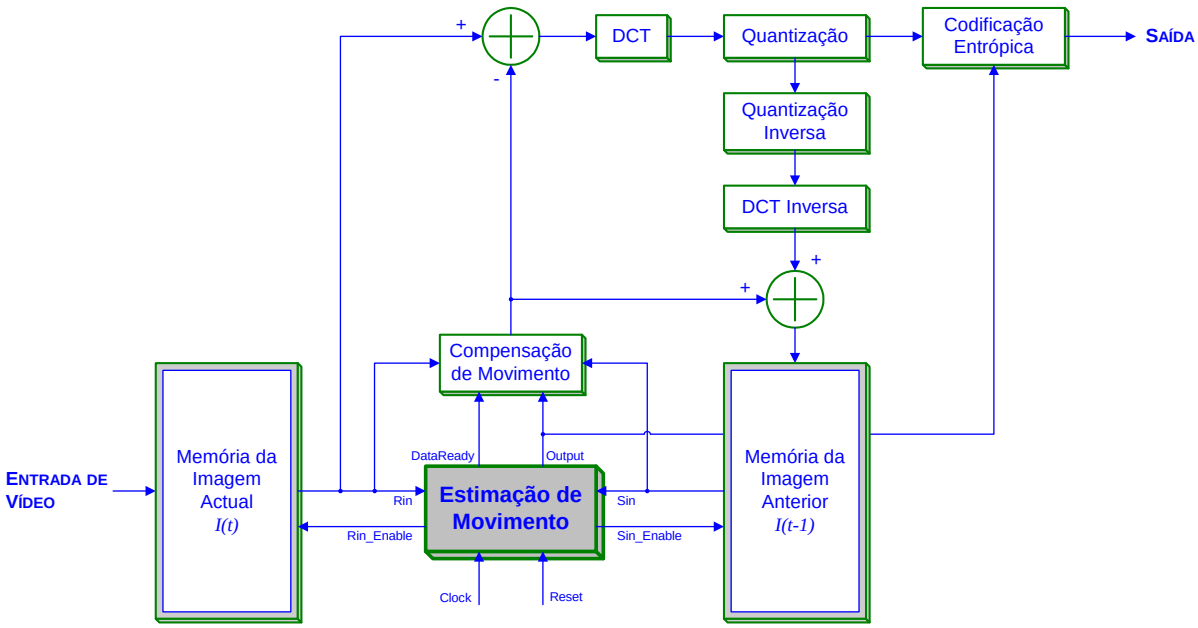


Figura 7.40: Integração do processador desenvolvido num sistema de codificação de vídeo.

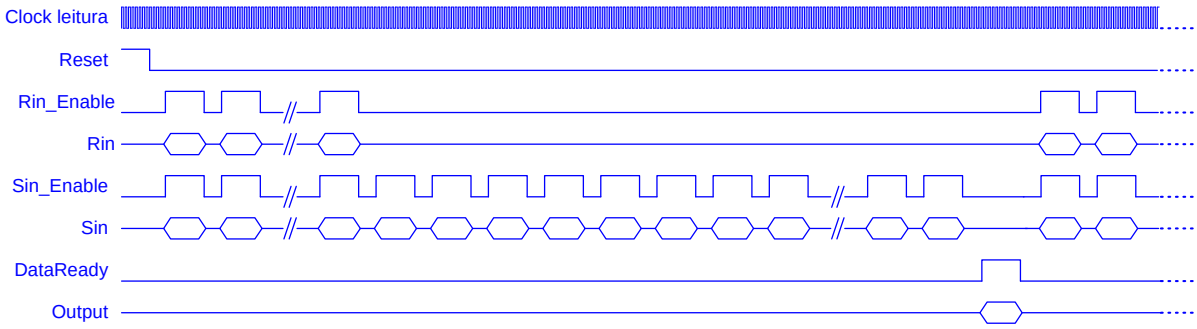


Figura 7.41: Diagrama temporal dos sinais de interface do processador concebido aquando do processo de estimação de movimento.

facto contrasta com a memória correspondente à imagem anterior, que é lida praticamente ao longo de todo o processamento. Refira-se, ainda, o comportamento do sinal de controlo DataReady, que permite informar o sistema de codificação exterior do término do processamento e, como consequência, da presença da informação correspondente às coordenadas do vector de movimento no porto de saída Output.

Capítulo 8

Características do processador e conclusões

Conteúdo

8.1 Tecnologia

O desenvolvimento do processador proposto foi realizado utilizando o processo tecnológico ES2 ECPD07, da empresa Atmel [46, 47, 48]. Este processo faz parte de um conjunto de processos tecnológicos disponibilizado por esta empresa, vocacionados para a implementação de circuitos integrados dedicados baseados em células lógicas[1]. Em particular, o desenvolvimento realizado foi efectuado com base na biblioteca ES2 de células normalizadas[2], contendo um conjunto completo de circuitos combinatórios e de células de memória.

Em termos tecnológicos, o processo ECPD07 baseia-se numa tecnologia CMOS[3] *standard*, com duas camadas de metal e utilizando uma largura de canal igual a $0,7$ µm. As características gerais desta tecnologia encontram-se descritas na tabela 8.1 [46, 49]. Apesar de não ser a tecnologia mais actual, foi utilizada por se encontrar disponível para a realização do processador.

Característica	**Valor**
Tensão de alimentação	5 V
Processo tecnológico	CMOS - $0,7$ µm, 2 camadas de metal
Tempo de propagação da célula lógica básica	230 ps (célula NAND2)
Frequência máxima de funcionamento	250 MHz (célula DFF - *flip-flop* tipo D)
Tipo de entrada/saída	TTL[4], CMOS

Tabela 8.1: Características gerais da tecnologia ECPD07 da Atmel [46] (valores típicos).

8.2 Descrição do circuito e ferramentas CAD utilizadas

A descrição dos diversos blocos que compõem o processador proposto foi realizada utilizando a linguagem de descrição de circuitos VHDL [38, 39]. Por forma a permitir a implementação de diversas configurações do processador, correspondentes a diferentes arquitecturas (ver secção 5.4), a programação foi efectuada para que o código seja facilmente parametrizável, através da utilização de entradas de configuração do tipo '`generic`'. Para além disso, para garantir os níveis de optimização desejados para as estruturas de processamento, optou-se por realizar uma descrição inteiramente estrutural [**?**].

[1] Do Inglês *Cell-Based ASICs (CBICs).*
[2] Do Inglês *Standard Cell Library (StdLib)* .
[3] Do Inglês *Complementary Metal-Oxide Semiconductor.*

O estudo de diferentes arquitecturas para os vários blocos que constituem o processador foi possível através de uma descrição modular dos respectivos circuitos. Desta forma, para substituir a implementação de um dado dispositivo por outro basta, apenas, alterar a respectiva instanciação nos vários blocos que o utilizam.

As ferramentas de projecto utilizadas foram as ferramentas que integram o pacote comercializado pela empresa *Synopsys*, nomeadamente:

- *VHDL compiler* - compilador da linguagem VHDL [44];
- *Design Analyzer* - conjunto de ferramentas de síntese e optimização de circuitos [43, 42];
- *VHDL Debugger* - utilizado para a simulação dos diversos circuitos que integram o processador [50, 51].

Todas estas ferramentas foram utilizadas numa estação de trabalho do tipo *Ultra Sparc-10*, equipada com 512 MBytes de memória física e com o sistema operativo *Solaris*, versão 7.

8.3 Resultados experimentais

Efectuada a descrição estrutural do processador em VHDL, verificou-se, através de uma simulação ao nível funcional, o correcto funcionamento de todas as suas unidades de processamento, de acordo com o que foi descrito nos capítulos anteriores. Nesta fase, aplicaram-se à entrada do processador vectores de teste reais, verificando-se, nos portos de saída, os valores esperados correspondentes aos vectores de movimento calculados.

Na segunda fase, procedeu-se à síntese de todos os blocos constituintes de processador usando, para tal, a biblioteca de células lógicas ES ECPD07 da Atmel, descrita na secção 8.1. Na síntese efectuada do processador utilizaram-se, como parâmetros de configuração, alguns valores correntemente utilizados nas normas de codificação de vídeo actuais [3, 4, 5], como sejam: macroblocos de referência de 16×16 pixels e deslocamentos em cada uma das direcções ortogonais da área de pesquisa definidos no intervalo $[-15, 16]$. Para além disso, optou-se por utilizar um único bloco activo, constituído por 16×16 elementos processadores. Os valores destes parâmetros encontram-se resumidos na tabela 8.2.

A síntese do circuito foi efectuada de uma forma incremental, começando-se por sintetizar todos os elementos de menor granularidade, correspondentes aos blocos de mais baixo nível, passando pelos elementos de nível intermédio e terminando no topo da estrutura, correspondente ao processador completo. Em cada um dos níveis intermédios

Parâmetro	Valor
N	16
p	16
h	16
ℓ	16
NC	1

Tabela 8.2: Parâmetros do processador utilizados na síntese realizada.

efectuaram-se, sempre que foi conveniente, procedimentos de optimização da respectiva estrutura, por forma a obter os níveis de desempenho desejados, nomeadamente, em termos de tempo de processamento. Estas optimizações foram realizadas através da imposição de restrições temporais entre nós críticos do circuito sob consideração. Em alguns casos particulares, onde o tempo de processamento não influencia o desempenho global do processador, optou-se por se impor restrições ao nível da área ocupada pelo circuito, minimizando, assim, os custos de implementação do processador.

Terminada a síntese do circuito, extraíram-se a partir do resultado obtido, os diversos valores que caracterizam cada um dos blocos em termos de tempo de processamento e em termos de área de material semicondutor ocupada, que se apresentam nas secções seguintes. Realizou-se, também, uma simulação do circuito sintetizado utilizando a tecnologia ECPD07, confirmando-se o correcto funcionamento do processador.

8.3.1 Frequência máxima de funcionamento

Na tabela 8.3 apresentam-se os valores correspondentes ao tempo de processamento de alguns dos blocos principais que compõem as unidades de processamento e controlo do processador. A tabela foi organizada de uma forma hierárquica, colocando os valores totais do tempo de processamento dos blocos de mais alto nível nas colunas colocadas mais à direita da tabela.

Como a separação entre os diversos níveis é, em geral, realizada por intermédio de registos de *pipeline*, a frequência máxima de funcionamento é determinada pelo bloco correspondente ao maior valor do tempo de processamento. Nota-se, no entanto, que para além deste tempo há ainda que considerar os tempos de estabelecimento[5] e de retenção[6] dos registos de *pipeline*, não considerados nesta tabela.

[5]Do Inglês *setup*.
[6]Do Inglês *hold*.

Matriz de Processamento	Elemento Processador Activo	AbsDiff	2,60 ns	6,76 ns	6,76 ns
		Accumulator	1,02 ns		
		Incrementer	0,95 ns		
	Elemento Processador Passivo			2,19 ns	
Somador em Árvore	Elemento de Processamento	Compressor	2,31 ns	4,41 ns	5,63 ns
		Adder2C	2,57 ns		
Comparador	Nível de Selecção	Comparador	3,65 ns	8,64 ns	8.72 ns
		Conversão de Formato	2,86 ns		
	Nível de Selecção Hierárquico	Elemento Processador	5,74 ns	0,94 ns	
Unidade de Controlo	Controlador Principal			6,62 ns	6,67 ns
	Controlador de Entrada - R_{in}			4,08 ns	
	Controlador de Entrada - S_{in}			4,46 ns	
	Gerador de Sinais de Relógio			0,98 ns	
PROCESSADOR COMPLETO					**8,72 ns**

Tabela 8.3: Tempo de processamento dos principais blocos que constituem as unidade de processamento e de controlo do processador.

De acordo com o que se pode concluir a partir dos valores apresentados, verifica-se que, tal como se tinha previsto no capítulo anterior, os blocos correspondentes ao *elemento processador activo da matriz de processamento* e ao *nível de selecção da unidade de comparação* são os blocos que determinam a frequência máxima de funcionamento, com tempos de propagação de $6,76$ ns e $8,64$ ns, respectivamente.

No que se refere ao primeiro destes blocos, o estudo do seu caminho crítico foi anteriormente realizado na secção 7.2.3, apresentando-se na figura 8.1 a distribuição do tempo de propagação pelos diversos componentes que constituem o seu caminho crítico.

O valor obtido para o primeiro nível de selecção da unidade de comparação pode parecer, à primeira vista, inesperado. De facto, considerando os tempos de propagação relativamente pequenos dos seus blocos comparador e conversor de formato, ilustrados na figura 7.17 ($3,65$ ns e $2,86$ ns, respectivamente), não era de esperar o valor relativamente elevado obtido para o bloco de nível de granularidade superior: cerca de $8,64$ ns. Após se ter investigado a razão para esta diferença de cerca de 5 ns, verificou-se que tal valor deve-se aos seguintes factores: *i)* à presença dos circuitos combinatórios responsáveis pelo controlo do valor inicial de comparação, descrito pela equação 7.16; *ii)* ao circuito que controla a entrada de activação do registo responsável pelo armazenamento do valor máximo da medida de similaridade até então encontrada, que é dependente do resultado da comparação e do estado da unidade central de controlo do processador; *iii)* aos tempos

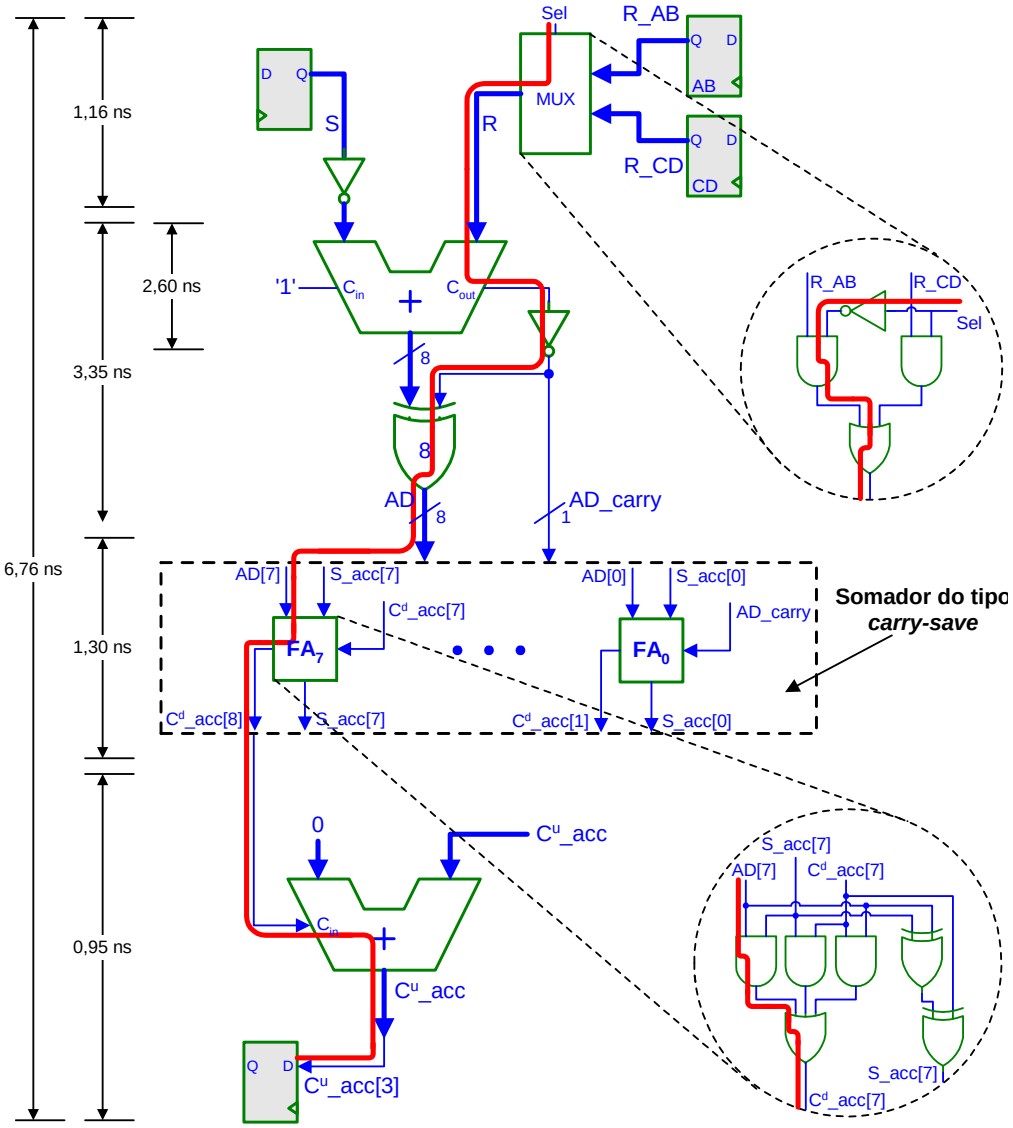


Figura 8.1: Distribuição do tempo de propagação total do elemento processador activo pelos diversos componentes que constituem o caminho crítico.

de estabelecimento e de retenção da entrada e saída do próprio registo, não incluídos nos tempos correspondentes às estruturas combinatórias do módulo comparador e de conversão de formato.

Note-se, porém, que tal como se referiu aquando da descrição da unidade de controlo efectuada na secção 7.6, as entradas de controlo dos registos de entrada R_{in} e S_{in} funcionam, ao contrário de todos os outros blocos do processador, a um ritmo mais elevado, dado pela frequência do relógio de leitura. De acordo com a equação 7.39, verifica-se que o factor de escalamento (α) entre a frequência de funcionamento dos restantes blocos do

processador e a frequência de leitura dos pixels é, para esta implementação, dada por:

$$\alpha = \left\lceil 1 + \frac{16+2}{2 \times 16} \right\rceil = 2, \tag{8.1}$$

pelo que $f_{leitura} = 2 \times f_{proc}$ que, em termos do período dos respectivos relógios se tem: $T_{leitura} = \frac{1}{2} \times T_{proc}$.

Assim sendo, a frequência máxima do sinal a aplicar no porto de entrada de relógio do processador (ver figura 7.39), será dada por:

$$f = \frac{1}{\max\left\{T_{max}^{leitura}, \frac{1}{2} \times T_{max}^{proc}\right\}}, \tag{8.2}$$

em que $T_{max}^{leitura}$ é o valor máximo do tempo de processamento correspondente ao caminho crítico das unidades de controlo dos registos de entrada (que funcionam com o relógio de leitura), e T_{max}^{proc} é o valor máximo do tempo de processamento correspondente ao caminho crítico dos restantes blocos do processador.

Considerando, a partir da tabela 8.3, os valores de $T_{max}^{leitura} = 4,46$ ns e $T_{max}^{proc} = 8,72$ ns, a frequência máxima de funcionamento do circuito será de:

$$f = \frac{1}{\max\left\{4,46 \text{ ns}, \frac{1}{2} \times 8,72 \text{ ns}\right\}} = \frac{1}{4,46 \text{ ns}} = 224,215 \text{ MHz}. \tag{8.3}$$

De acordo com a tabela 8.2, correspondente às características da tecnologia de implementação utilizada, verifica-se que este valor se encontra bastante próximo do valor máximo para a frequência de funcionamento admissível por esta tecnologia (250 MHz) pelo que se conclui que, para além do processador concebido se encontrar dentro da gama de funcionamento admissível, a implementação tira partido das potencialidades desta tecnologia.

Assim, as frequências máximas de funcionamento do processador são as seguintes:

$$f_{entrada} = 224,215 \text{ MHz} \quad \Leftrightarrow \quad \begin{cases} f_{leitura} = 224,215 \text{ MHz} \\ f_{proc} = \frac{1}{2} \times f_{leitura} = 112,108 \text{ MHz} \end{cases} \tag{8.4}$$

8.3.2 Área ocupada

Na tabela 8.4 apresentam-se os valores correspondentes à área ocupada pelos principais blocos que compõem as unidade de processamento e controlo do processador. À semelhança do que se fez na secção anterior, dispuseram-se estes valores de uma forma hierárquica, decompondo os valores das unidades de processamento de mais alto nível de granularidade (apresentados na coluna mais à direita), nas áreas ocupadas pelos principais blocos que as constituem.

Matriz de Processamento	Elemento Processador Activo	AbsDiff	118 472,6 μm^2	492 871,3 μm^2 x (16x16 unid.)	201,28 mm^2
		Accumulator	54 811,2 μm^2		
		Incrementer	5 950,8 μm^2		
	Elemento Processador Passivo			151 430,0 μm^2 x (31x16 unid.)	
Somador em Árvore	Elemento de Processamento	Compressor	108 303,8 μm^2	217 694,4 μm^2 x (15 unid.)	3,279 mm^2
		Adder2C	34 051,8 μm^2		
Comparador	Nível de Selecção	Comparador	203 832,0 μm^2	657 308,8 μm^2	0,763 mm^2
		Conversão de Formato	204 044,8 μm^2		
	Nível de Selecção Hierárquico	Elemento Processador	291 156,0 μm^2	29 358,8 μm^2	
Unidade de Controlo	Controlador Principal			151 635,2 μm^2	1,619 mm^2
	Controlador de Entrada - R_{in}			396 594,6 μm^2	
	Controlador de Entrada - S_{in}			1 066 314,3 μm^2	
	Gerador de Sinais de Relógio			4 529,6 μm^2	
PROCESSADOR COMPLETO					**206,946 mm^2**

Tabela 8.4: Área de semicondutor ocupada pelos principais blocos que constituem as unidades de processamento e controlo do processador.

Note-se, porém, que devido aos valores dos parâmetros utilizados na configuração adoptada, alguns destes blocos de nível intermédio podem, na realidade, não ser utilizados na síntese global do sistema. Um exemplo desta situação verifica-se com o elemento processador do nível de selecção hierárquico da unidade de comparação que, devido à adopção de um único bloco activo ($NC = 1$) não é, neste caso, utilizado na síntese global do circuito.

Conforme se pode verificar, a unidade responsável pela maior parte da área ocupada pelo processador é a matriz de processamento, correspondendo a cerca de $97,26\%$ da área total ocupada. Tal era já esperado, visto que um dos compromissos adoptados para atingir os níveis de desempenho desejados consistiu no aumento do nível de paralelismo utilizado no cálculo da medida de similaridade.

Pode-se concluir a partir da análise da tabela 8.4 que o circuito integrado requer uma área de semicondutor total de cerca de $206,946$ mm^2 que, apesar de ser um valor elevado, se encontra dentro dos limites aceitáveis para esta tecnologia.

8.3.3 Desempenho

Os valores apresentados nas secções anteriores permitem-nos estimar o desempenho do estimador de movimento desenvolvido, nomeadamente, no que se refere ao ritmo de processamento.

Conforme a descrição efectuada na secção 5.4, o tempo necessário para efectuar a estimação do vector de movimento de um determinado macrobloco é dado por:

$$t_{macrobloco} = t_{proc} + t_{leitura}, \tag{8.5}$$

em que t_{proc} é o tempo necessário para a estimação propriamente dita e $t_{leitura}$ é o tempo necessário para o preenchimento dos elementos processadores da matriz de processamento com os pixels da área de pesquisa e do macrobloco de referência.

De acordo com a expressão 5.23a, o valor de $t_{leitura}$ é dado por:

$$t_{leitura} = (h \times L') \times T_{leitura}, \tag{8.6}$$

em que $T_{leitura}$ é o período do relógio de leitura, correspondente à frequência máxima de funcionamento.

De acordo com a tabela 5.2, o valor de t_{proc} para a topologia A é dado por:

$$t_{proc} = (2p)^2 \times T_{proc}, \tag{8.7}$$

em que T_{proc} é o período do relógio de processamento, dado por $T_{proc} = 2 \times T_{leitura}$.

Considerando uma implementação em que $h = \ell = N$, a expressão 8.5 vem dada por:

$$\begin{aligned} t_{macrobloco} &= \left[2.(2p)^2 + N.(2p + N - 1)\right] \times T_{leitura} && \text{(8.8a)} \\ &= N^2 \left[2.(2k)^2 + \left(2k + 1 - \frac{1}{N}\right)\right] \times T_{leitura}, && \text{(8.8b)} \end{aligned}$$

em que o valor de k é, tal como nos capítulos anteriores, dado pela razão entre o deslocamento máximo considerado em cada direcção na área de pesquisa (p) e a dimensão do macrobloco de referência (N): $k = \frac{p}{N}$.

Considerando uma imagem constituída por $N_h \times N_v$ pixels, o tempo de processamento total será dado por:

$$\begin{aligned} t_{imagem} &= \frac{(N_h \times N_v)}{N^2} \times N^2 \left[2.(2k)^2 + \left(2k + 1 - \frac{1}{N}\right)\right] \times T_{leitura} && \text{(8.9a)} \\ &= \frac{(N_h \times N_v) \times \left[2.(2k)^2 + \left(2k + 1 - \frac{1}{N}\right)\right]}{f_{leitura}} && \text{(8.9b)} \end{aligned}$$

Desta forma, o ritmo de processamento será dado por:

$$R = \frac{1}{t_{imagem}} = \frac{f_{leitura}}{(N_h \times N_v) \times \left[2.(2k)^2 + \left(2k + 1 - \frac{1}{N}\right)\right]} \tag{8.10}$$

O gráfico apresentado na figura 8.2 ilustra a variação do ritmo de processamento com o factor k para diferentes formatos da imagem, considerando macroblocos constituídos por

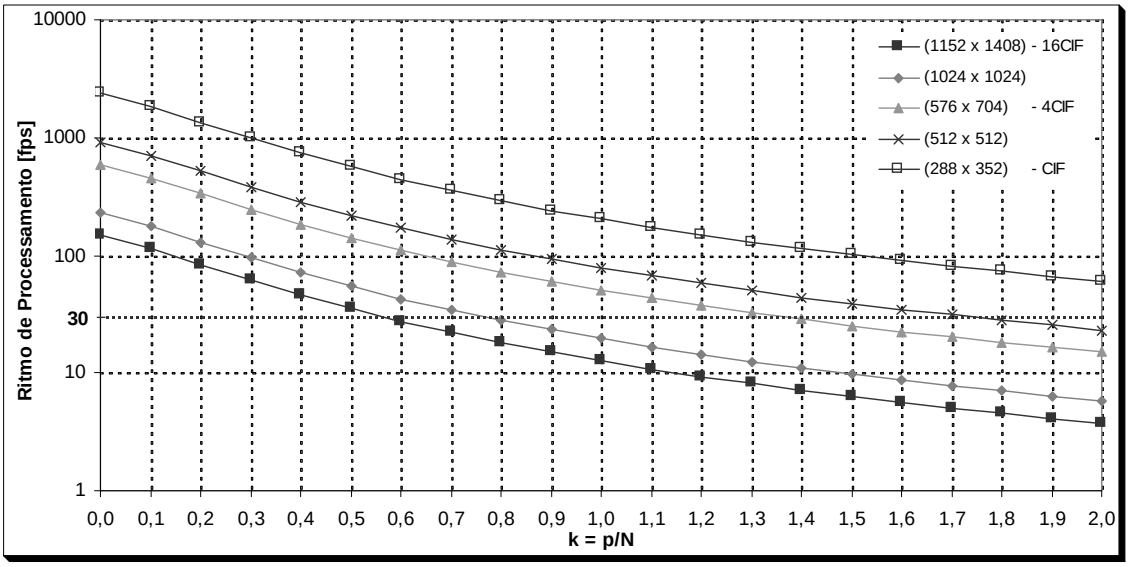


Figura 8.2: Número de tramas processadas por segundo para diferentes formatos de imagem.

16×16 pixels. Conforme se pode verificar, para valores de k inferiores a $0,5$ o processador apresenta capacidade de processar todos os formatos de imagem considerados em tempo real, correspondente a um ritmo de processamento mínimo de cerca de 30 fps. Para $k = 1$, correspondente à situação de se ter $p = N$, os valores dos ritmos de processamento são os apresentados na tabela 8.5, onde se pode verificar que apenas para os formatos de imagem correspondentes a vídeo de muito alta resolução não se atinge processamento em tempo real. No entanto, os valores obtidos permitem prever que as arquitecturas propostas podem, com a utilização de uma tecnologia mais recente ($\lambda = 0,35\mu$m), obter processamento em tempo real para imagens de elevada resolução.

Formato da imagem	**Ritmo de Processamento (*fps*)**
1152×1408 - 16CIF	$12,6$
1024×1024	$19,6$
576×704 - 4CIF	$50,6$
512×512	$78,2$
288×352 - CIF	$202,2$

Tabela 8.5: Ritmo de processamento para diferentes formatos de imagem e para $k = 1$ ($p = N$).

8.4 Conclusões

Na presente dissertação, apresentou-se um trabalho de investigação na área dos processadores dedicados para estimação de movimento em sequências de vídeo, baseado no método de emparelhamento de blocos com pesquisa exaustiva.

Este trabalho começou com um estudo das técnicas de codificação de vídeo preditivas, que permitem explorar a redundância temporal em sequências de imagens, em particular, as técnicas de estimação de movimento por emparelhamento de blocos. Após se terem considerado várias medidas de similaridade para a estimação, concluiu-se que a medida de dissemelhança SAD[7] é a mais apropriada para o desenvolvimento de circuitos estimadores de movimento, dado produzir bons resultados e apresentar uma reduzida complexidade computacional. O estudo realizado abordou, ainda, a análise de algoritmos de pesquisa óptimos e sub-óptimos, tendo-se procedido a uma análise comparativa das vantagens de uns e de outros.

O desenvolvimento da arquitectura apresentada nesta tese foi realizado com base numa análise comparativa de várias arquitecturas propostas nos últimos anos para estimação de movimento, nomeadamente, arquitecturas bidimensionais (AB2, *Vos*, *Hsieh*, AS2, Tipo 2 e *Chang*) e unidimensionais (AB1 e AS1). Esta análise foi realizada com base nas características das arquitecturas, em termos de área ocupada (A), tempo de processamento (T) e produto área-tempo (AT) dos circuitos correspondentes. Tomando como medida o produto área-tempo, concluiu-se, desta análise, que a arquitectura bidimensional proposta por *Vos* é a mais eficiente.

Nesta dissertação é proposta uma nova classe de arquitecturas baseadas na arquitectura bidimensional proposta por *Vos*, que apresentam uma eficiência superior a todas as arquitecturas até hoje conhecidas. Esta nova classe introduz melhorias ao nível do esquema de processamento, da transferência de dados para a matriz de processadores e do paralelismo utilizado no cálculo simultâneo de múltiplas medidas de similaridade. Para além disso, estas novas arquitecturas são escaláveis, podendo-se diminuir o tempo de processamento aumentando o nível de paralelismo explorado, através da introdução de blocos de processamento adicionais. Por outro lado, conseguiu-se, com estas novas arquitecturas, uma eficiência de utilização dos recursos de *hardware* próxima dos 100%. Em relação à arquitectura proposta por *Vos*, regista-se uma diminuição do número de elementos de memória requeridos de aproximadamente 50%. Tal foi conseguido através de uma disposição dos vários elementos de processamento segundo uma superfície cilíndrica, permitindo, assim, um maior aproveitamento dos recursos.

[7] Do Inglês *Sum of Absolute Differences*.

Para realizar estimadores de movimento eficientes é fundamental dispor de unidades aritméticas rápidas, nomeadamente, para o cálculo de diferenças, de somas e de valor absoluto. Para o desenvolvimento deste tipo de unidades aritméticas consideraram-se estruturas de soma do tipo *carry-lookahead*, baseadas em aritmética redundante e do tipo *prefix-adder*. Concluiu-se que as unidades aritméticas do tipo *prefix-adder* são as mais eficientes para esta aplicação, tendo-se adoptado as unidades propostas por *Sklansky* para o bloco do circuito de soma e de cálculo do valor absoluto da diferença.

O desenvolvimento do processador incluiu, também, uma fase de optimização dos diversos circuitos, nomeadamente, através do ajuste do comprimento de palavra necessário para representar os resultados produzidos nos diferentes nós de processamento; e do desenvolvimento de um método de projecto automático de estruturas de soma e de comparação em árvore, para um número de operandos que não seja uma potência inteira de 2.

Por fim, procedeu-se à descrição do processador para estimação de movimento proposto em VHDL. O processador foi sintetizado utilizando a biblioteca de células lógicas ES2 ECPD07 da Atmel, baseado numa tecnologia CMOS - $0,7$ µm. Dos resultados obtidos conclui-se que o processador pode funcionar à frequência de 100 MHz, ocupando uma área de semicondutor de cerca de 200 mm^2. Estes resultados permitem concluir que, com base nas normas actuais de codificação de vídeo, o circuito desenvolvido tem capacidade de estimar movimento em tempo real em sequências de imagens de média e elevada definição.

O elevado desempenho do processador dedicado para estimação de movimento foi obtido com base num trabalho de investigação realizado aos vários níveis do projecto de circuitos de processamento dedicados. Desse trabalho resultou a proposta de novas arquitecturas e esquemas de processamento, o desenvolvimento de unidades aritméticas rápidas e o projecto de um circuito integrado, que permite estimar movimento em tempo real.

Referências

[1] Kevin Martins Ferreira, *Compressão de Video para Canais de Banda Estreita*, Tese de Mestrado, Instituto Superio Técnico, Lisbon, Março 1996.

[2] Vasudev Bhaskaran e Konstantinos Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, Kluwer Academic Publishers, primeira edição, 1995.

[3] ITU-T, *ITU-T Recommendation H.261 - "Video Codec for Audiovisual Services at* $p \times 64$ *kbits"*, Março 1993.

[4] ITU-T, *ITU-T Recommendation H.262 - "Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Video"*, Setembro 1995.

[5] ITU-T, *ITU-T Recommendation H.263 - "Video Coding for Low Bitrate Communication"*, Fevereiro 1998.

[6] ISO, *ISO/IEC JTC1 CD 11172 - "Coding of moving pictures and associated audio for digital storage media up to 1.5 Mbit/s"*, 1992.

[7] ISO, *ISO/IEC JTC1 CD 13818 - "Generic coding of moving pictures and associated audio"*, 1994.

[8] ITU-T, *ITU-T Recommendation T.81 - "Digital compression and coding of continuous-tone still images"*, 1993.

[9] L. Vos e M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Transactions on Circuits and Systems*, volume 36, nº 10, págs. 1309–1316, Outubro 1989.

[10] M. Ibrahim Sezan e Reginald L. Lagendijk (eds.), *Motion Analysis and Image Sequence Processing*, Kluwer Academic Publishers, 1993.

[11] Keshab K. Parhi e Takao Nishitani (eds.), *Digital Signal Processing for Multimedia Systems*, Marcel Dekker, Inc., 1999.

[12] T. Koga, K. Iinuma, A. Hirano, Y. Iijima e T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. Nat. Telecomm. Conference*, New Orleans, LA, Novembro 1981, págs. G5.3.1–G5.3.5.

[13] J. R. Jain e A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, volume COM-29, nº 12, págs. 1799–1808, Dezembro 1981.

[14] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Transactions on Image Processing*, volume 38, nº 7, págs. 950–953, Julho 1990.

[15] Liang-Wei Lee, Jhing-Fa Wang, Jau-Yien Lee e Jung-Dar Shie, "Dynamic search-window adjustment and interlaced search for block-matching algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, volume 3, nº 1, págs. 85–87, Fevereiro 1993.

[16] Renxiang Li, Bing Zeng e Ming L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, volume 4, nº 4, págs. 438–442, Agosto 1994.

[17] E. Ogura, Y. Ikenaga, Y. Iida, Y. Hosoya, M. Takashima e K. Yamash, "A cost effective motion estimation processor LSI using a simple and efficient algorithm," in *Proceedings of International Conference on Consumer Electronics - ICCE*, 1995, págs. 248–249.

[18] S. Kirkpatrick, Jr. C. D. Gelatt e M. P. Vecchi, "Optimization by simulated annealing," *Science*, volume 220, págs. 671–680, Maio 1983.

[19] T. Komarek e P. Pirsch, "Array architectures for block matching algorithms," *IEEE Transactions on Circuits and Systems*, volume 36, nº 10, págs. 1301–1308, Outubro 1989.

[20] S. Chang, J. H. Hwang e C. W. Jen, "Scalable array architecture design for full search block matching," *IEEE Transactions on Circuits and Systems for Video Technology*, volume 5, nº 4, págs. 332–343, Agosto 1995.

[21] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988.

[22] C. H. Hsieh e T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, volume 2, nº 2, págs. 169–175, Junho 1992.

[23] K. M. Yang, M. T. Sun e L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Transactions on Circuits and Systems*, volume 36, nº 10, págs. 1317–1325, Outubro 1989.

[24] L. Vos e M. Schobinger, "VLSI architectures for a flexible block matching processor," *IEEE Transactions on Circuits and Systems for Video Technology*, volume 5, nº 5, págs. 417–428, Outubro 1995.

[25] L. Vos, M. Stegherr e T. G. Noll, "VLSI architectures for the full-search blockmatching algorithm," in *Proceedings of the ICASSP'89*, 1989, págs. 1687–1690.

[26] L. Vos e M. Schobinger, "Efficient architecture of a programmable block matching processor," in *Proceedings of the International Conference on Applications - Specific Array Processors*, Venice - Italy, 1993, pág. 560.

[27] R. P. Brent e H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, volume C-31, nº 3, págs. 260–264, Março 1982, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[28] D. Goldberg, "Computer arithmetic," in *Computer Architecture: A Quantitative Approach* (editado por J. L. Hennessy e D. A. Patterson), Morgan Kaufmann, appendix A, págs. A1–A72, segunda edição, 1996.

[29] T. F. Ngai, M. J. Irwin e S. Rawat, "Regular, area-time efficient carry-lookahead adders," *Journal of Parallel and Distributed Computing*, volume 3, nº 3, págs. 92–105, 1986, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[30] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on electronic computers*, volume 10, págs. 389–400, 1961, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[31] Keshab K. Parhi, "Fast VLSI binary addition," in *Proc. of 1997 IEEE Workshop on Signal Processing Systems: Design and Implementation*, Leicester, U.K., Novembro 1997, págs. 232–241.

[32] Keshab K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, Wiley-Interscienc, 1999.

[33] J. Sklansky, "Conditional sum addition logic," *IRE Transactions on Electronic Computers*, volume EC-9, nº 6, págs. 226–231, Junho 1960.

[34] Akhilesh Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Transactions on Computers*, volume 42, nº 10, págs. 1163–1170, Outubro 1993.

[35] Keshab K. Parhi, "Fast low-energy VLSI binary addition," in *Proc. of IEEE Conference on Computer Design*, Austin, Outubro 1997, págs. 676–684.

[36] H. R. Srinivas e Keshab K. Parhi, "A fast VLSI adder architecture," *IEEE Journal of Solid-State Circuits*, volume 27, nº 5, págs. 761–767, Maio 1992.

[37] K. Hwang, *Computer Arithmetic Principles, Architecture and Design*, Wiley, New York, 1979.

[38] IEEE, *IEEE Std 1076-1987 - IEEE standard VHDL language reference manual*, Março 1988.

[39] IEEE, *ANSI/IEEE Std 1076-1993 - IEEE standard VHDL language reference manual*, Junho 1994.

[40] Zainalabedin Navabi, *VHDL : Analysis and Modeling of Digital Systems (McGraw-Hill Series in Electrical and Computer Engineering)*, McGraw-Hill Companies, Inc., primeira edição, Setembro 1993.

[41] Zoran Salcic, *VHDL and FPLDs in digital systems design, prototyping and customization*, Kluwer Academic Publishers, 1998.

[42] Synopsys, Inc., *Design Analyzer Reference Manual (v1999.10)*, 1999.

[43] Synopsys, Inc., *Design Compiler User Guide (v1999.10)*, 1999.

[44] Synopsys, Inc., *VHDL Compiler Reference Manual (v1999.10)*, 1999.

[45] Herbert Taub, *Digital circuits and microprocessors*, McGraw-Hill, Inc., 1982.

[46] Atmel ES2, *ES ECPD07 Library Databook*, Setembro 1997.

[47] Atmel ES2, *ATMEL CBIC Design Guidelines*, Junho 1996.

[48] Atmel ES2, *ES2 Synopsys Synthesis Kit User Guide*, Outubro 1995.

[49] Synopsys, Inc., *Semiconductor Vendor Partners - Atmel Cell-Based ASIC Products*, Jan. 2000, URL `http://www.synopsys.com/partners/svp/atmel_cb_svp.html`.

[50] Synopsys, Inc., *VHDL Simulation Reference Manual (v1999.10)*, 1999.

[51] Synopsys, Inc., *VHDL Simulation User Guide (v1999.10)*, 1999.