Universidade de Aveiro Departamento de Electrónica, Informática e Telecomunicações
2007

Marco Paulo Carvalho Ribeiro dos Santos

Codificação de Vídeo MPEG-4 em FPGA

Universidade de Aveiro Departamento de Electrónica, Informática e Telecomunicações
2007

Marco Paulo Carvalho Ribeiro dos Santos

Codificação de Vídeo MPEG-4 em FPGA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Prof. Doutor António Navarro, Professor do Departamento de Engenharia Electrónica e Telecomunicações da Universidade de Aveiro

Dedico esta dissertação à Inês Mendes, aos meus pais e ao “Lucas”.

O Júri

Presidente

Doutor António Manuel de Brito Ferrari Almeida
Professor Catedrático da Universidade de Aveiro

Doutor Leonel Augusto Pires Seabra de Sousa
Professor Associado do Instituto Superior Técnico da Universidade Técnica de Lisboa

Doutor António José Nunes Navarro Rodrigues
Professor Auxiliar da Universidade de Aveiro

Agradecimentos

Agradeço a todas as pessoas que de algum modo me ajudaram, incentivaram e apoiaram na elaboração desta dissertação. No entanto, quero especialmente agradecer ao meu orientador Prof. Doutor António Navarro pela sua compreensão, ajuda e oportunidades que me deu porque sem elas este trabalho era impossível. Além disso, fica aqui um enorme agradecimento ao meu amigo António Silva pela sua ajuda inestimável. Por fim, agradeço à Universidade de Aveiro e à empresa Xilinx, Inc. por ter facultado o uso da plataforma de desenvolvimento Wildcard™II para a elaboração desta dissertação.

Palavras-chave

Codificação de video, YUV, MPEG-4, MPEG4 AVC/H.264, FPGA, VHDL, IDCT, VLD, VLC, códigos de comprimento variável, deblocking MPEG-4 AVC/H.264, remoção de artefactos MPEG-4 AVC/H.264, estimação de movimento.

Resumo

O trabalho apresentado nesta dissertação foi realizado no âmbito do grupo ISG (*Implementation Study Group*) do MPEG (*Motion Picture Expert Group*). Tem como objectivo principal analisar o desempenho da implementação de diversões módulos da norma MPEG-4 e da sua versão avançada MPEG-4 AVC (*Advanced Video Coding*) em FPGA (*Field Programmable Gate Array*). A dissertação é essencialmente constituída por duas partes. Na primeira parte, estudam-se os conceitos básicos da codificação de vídeo, nomeadamente os conceitos integrantes da norma MPEG-4. De seguida, estuda-se a norma MPEG-4 AVC. Na segunda parte, esta dissertação apresenta o desenvolvimento de seis módulos implementados em VHDL e compilados para o circuito FPGA da XILINX, Virtex-II. Dos seis módulos, quatro constituem um descodificador MPEG-4 (excepto a compensação de movimento) enquanto os outros dois fazem parte do codificador MPEG-4 AVC.

keywords

Video coding, YUV, MPEG-4, MPEG4 AVC/H.264, FPGA, VHDL, IDCT, VLD, VLC, variable length codes, deblocking MPEG-4 AVC/H.264, motion estimation.

Abstract

This dissertation shows the work developed in the framework of the ISG (Implementation Study Group) group. The main objective of this work is to analyse the performance of several MPEG-4 modules and MPEG-4 AVC (Advanced Video Coding) modules in FPGA (field Programmable Gate Array). This dissertation is organized into two parts. In the first part, the basic video coding concepts are studied, mainly the MPEG-4 concepts, followed by the MPEG-4 AVC standard. In the second part, it is discussed six modules implemented in VHDL and compiled for the XILINX Virtex-II FPGA. Four of those six developed modules are used to implement an MPEG-4 decoder (except the motion compensation stage) and the other two are modules of the MPEG-4 AVC coder.

ÍNDICE

ACRÓNIMOS E GLOSSÁRIO 5

CAPÍTULO 1. INTRODUÇÃO 9

CAPÍTULO 2. CONCEITOS DE VÍDEO 11

2.1- AMOSTRAGEM 11
2.2- ESPAÇO DE COR (*COLOUR SPACE*) 13
2.2.1 RGB 13
2.2.2 YCbCr 14
2.3- FORMATOS DE AMOSTRAGEM 16
2.4- FORMATOS DE VÍDEO 19
2.5- QUALIDADE DE VÍDEO 19

CAPÍTULO 3. CODIFICAÇÃO DE VÍDEO 22

3.1- MODELO TEMPORAL 23
3.2- MODELO ESPACIAL 27
3.2.1 Transformada 28
3.2.2 Quantificador 32
3.2.3 Reordenação e codificador de zeros. 34
3.3- CODIFICADOR DE ENTROPIA 35
3.3.1 Codificação preditiva – Predictive coding 35
3.3.2 Códigos de comprimento variável – VLC 36
3.4- CODIFICADOR DPCM/DCT 38

CAPÍTULO 4. NORMA MPEG-4 AVC/H.264 43

4.1- CAMADA DO CODIFICADOR DE VÍDEO – VCL 47
4.1.1 Imagens e campos 49
4.1.2 Espaço de cor YC_bC_r e amostragem 4:2:0 49
4.1.3 Divisão da imagem em macroblocos 49
4.1.4 Partições e grupos de partições – Slices e Group Slices 49
4.1.5 Codificação adaptativa de Imagem/Campo (Frame/Field) 51
4.1.6 Predição Intra – Intra frame prediction 53
4.1.7 Predição Inter – Inter predicton 55
4.1.8 Transformada 58
4.1.9 Entropia 63
4.1.10 Filtro de remoção de artefactos – Deblocking filter 65

CAPÍTULO 5. PLATAFORMA DE DESENVOLVIMENTO 68

5.1- WILDCARD™II 68
5.1.1 API de software 70
5.1.2 Modelos VHDL 73
5.1.3 Interfaces PE 74
5.2- FPGA – VIRTEX-II™ 82

CAPÍTULO 6. MÓDULOS DESENVOLVIDOS 86

6.1- MÓDULO VLD 87
6.1.1 Algoritmo 88
6.1.2 Implementação 90
6.1.3 Resultados 92
6.2- MÓDULO IQ 93
6.2.1 Algoritmo 93
6.2.2 Implementação 97
6.2.3 Resultados 101
6.3- MÓDULO IDCT 101
6.3.1 Algoritmo 102
6.3.2 Implementação 107
6.3.3 Resultados 111
6.4- MÓDULO VLD+IQ+IDCT 112
6.4.1 Implementação 112
6.4.2 Resultados 114
6.5- MÓDULO DEBLOCKING 114
6.5.1 Algoritmo 115
6.5.2 Implementação 116
6.5.3 Resultados 118
6.6- MÓDULO ESTIMAÇÃO DE MOVIMENTO 118
6.6.1 Algoritmo 121
6.6.2 Implementação 123
6.6.3 Resultados 128

CAPÍTULO 7. CONCLUSÕES 129

REFERÊNCIAS 132

ANEXO 135

ÍNDICE DE FIGURAS

Figura 1 – Amostragem espacial e temporal. 11
Figura 2 – Sequência de vídeo entrelaçado. 13
Figura 3 – RGB. 14
Figura 4 – Formato de amostragem 4:4:4. 17
Figura 5 – Formato de amostragem 4:2:2. 17
Figura 6 – Formato de amostragem 4:2:0. 18
Figura 7 – CoDec de vídeo. 22
Figura 8 – Diagrama de blocos dum codificador de vídeo. 23
Figura 9 – Coeficientes da DCT de um bloco 4×4. 31
Figura 10 – Reconstrução a partir de a) um, b) dois, c) três e d) cinco coeficientes. 31
Figura 11 – Quantificador a) linear e b) não linear. 33
Figura 12 – Varrimento em ziguezague para a) quadro e b) campo. 34
Figura 13 – Codificador DPCM/DCT. 38
Figura 14 – Descodificador DPCM/DCT 38
Figura 15 – Imagem actual F_n. 39
Figura 16 – Imagem de referência F'_{n-1}. 39
Figura 17 – Imagem residual F_n-F'_{n-1}. 40
Figura 18 – Vectores de movimento. 40
Figura 19 – Foco de interesse das normas de codificação de vídeo. 44
Figura 20 – Estrutura do codificador de vídeo MPEG-4 AVC/H.264. 45
Figura 21 – Diagrama de blocos do codificador MPEG-4 AVC/H.264. 47
Figura 22 – Perfis do MPEG-4 AVC/H.264. 48
Figura 23 – Exemplos de group slices: a) interleaved, b) dispersed e c) foreground and background. 51
Figura 24 – Conversão campo/imagem. 53
Figura 25 – a) Intra_4×4 e b) as oitos direcções. 53
Figura 26 – Modos de predição Intra_4x4. 54
Figura 27 – Modos de predição Intra_16×16. 54
Figura 28 – partições de a) macroblocos e b) sub-macroblocos. 56
Figura 29 – Interpolação para luminância *half-sample*. 57
Figura 30 – Interpolação para luminância *quarter-sample*. 57
Figura 31 – Princípio de funcionamento do filtro de remoção de artefactos. 66
Figura 32 – a) com filtro remoção de artefactos e b) sem filtro. 66
Figura 33 – Placa PCMCIA Wildcard™II. 68
Figura 34 – Diagrama de blocos da WCII. 69
Figura 35 – Níveis de simulação da WCII. 74
Figura 36 – Ciclo de escrita LAD Bus na WCII. 78
Figura 37 – Ciclo de leitura LAD Bus na WCII. 78
Figura 38 – Ciclo de escrita ZBT-SRAM na WCII. 80
Figura 39 – Ciclo de leitura ZBT-SRAM na WCII. 80
Figura 40 – Ciclo de escrita DDR-SDRAM da WCII. 81
Figura 41 – Ciclo de leitura DDR-SDRAM da WCII. 82
Figura 42 – Arquitectura de uma FPGA Virtex-II™ da Xilinx®. 83
Figura 43 – Virtex-II™ IOB. 83
Figura 44 – a) Virtex-II™ CLB e b) Virtex-II™ *slice*. 84
Figura 45 – Algoritmo do módulo VLD. 88
Figura 46 – Algoritmo para a descodificação de coeficientes não DC. 89
Figura 47 – Diagrama de blocos do módulo VLD. 90
Figura 48 – Diagrama temporal do módulo VLD. 92
Figura 49 – Matriz de pesos para a quantificação inversa a) Intra e b) Inter. 96
Figura 50 – Diagrama de blocos do módulo IQ. 97
Figura 51 – a) Fluxograma de quantificação inversa para MPEG-4 e b) controlo m*ismatch*. 98
Figura 52 – Fluxograma para quantificação inversa H.263. 99
Figura 53 – Diagrama temporal do módulo IQ. 100

Figura 54 – Diagrama de blocos do módulo IDCT. 107
Figura 55 – Diagrama de blocos dos sub-módulos do módulo IDCT. 108
Figura 56 – Esquema da multiplicação M do sub-módulo OP3 do módulo IDCT. 109
Figura 57 – Esquema da multiplicação γ_4M do sub-módulo OP3 do módulo IDCT. 109
Figura 58 – Diagrama temporal do módulo IDCT. 111
Figura 59 – Diagrama de blocos do módulo VLD+IQ+IDCT. 113
Figura 60 – Diagrama temporal do módulo VLD+IQ+IDCT. 113
Figura 61 – Algoritmo do módulo Deblocking. 115
Figura 62 – Diagrama de blocos do módulo Deblocking. 116
Figura 63 – Diagrama temporal do módulo Deblocking. 117
Figura 64 – Processo de procura *block-matching*. 119
Figura 65 – Modo a) macrobloco e b) 8×8. 120
Figura 66 – Janela de procura (*search window*). 121
Figura 67 – Algoritmo do módulo Estimação de Movimento. 122
Figura 68 – SADs 4×4, 8×4, 8×8, 16×8, 8×16 e 16×16. 123
Figura 69 – Diagrama de blocos do módulo Estimação de Movimento. 124
Figura 70 – Organização das memórias SRAM no módulo Estimação de Movimento. 126
Figura 71 – Diagrama temporal do módulo Estimação de Movimento. 127

ACRÓNIMOS E GLOSSÁRIO

ADC *Analog to Digital Converter* – Conversor de Analógico para Digital.

AND Função lógica E (AND em inglês).

API *Application Programming Interface* – Interface de Programação de Aplicativos. É um conjunto de rotinas e normas estabelecidos por um objecto para oferecer as suas funcionalidades a outros objectos, sem que estes necessitem de saber como funciona internamente o objecto que oferece a API.

ASIC *Application Specific Integrated Device* – Circuito Integrado de Aplicação Específica. É um circuito integrado definido para uma aplicação específica.

AVC ver H.264.

BRAM *Block SelectRam* **–** Bloco de memória RAM com dupla porta (Dual port RAM) específico da família de FPGA Virtex-II da XILINX.

Buffer Espaço de memória temporário.

CardBus ver PCMCIA.

Chroma O mesmo que crominância (Cb ou U para azul e Cr ou V para vermelho).

CLB *Configurable logic block* – Bloco de lógica configurável.

CoDec Acrónimo derivado da conjugação das palavras inglesas para codificador, *enCoder*, e para descodificador, *Decoder*.

CPLD *Complex Programmable Logic Device* – Dispositivo de Lógica Programavél Complexo. É dispositivo semicondutor basicamente constituído por várias PAL que estão interligadas por uma matriz.

CPU *Central Processing Unit* – Unidade central de processamento.

DAC *Digital to Analog Converter* – Conversor de Digital para Analógico.

DCM *Digital Clock Manager* – Unidade digital controladora de relógios específica da família de FPGA Virtex-II da XILINX**.**

DCT *Discrete Cosine Transform* – Transformada de Co-senos Discreta.

DDR *Double Data Rate* – Taxa de Transferência Dupla. Técnica de transferência de dados síncrona nos dois flancos do relógio duplicando assim a taxa de transferência de dados.

DDR-SDRAM Memória SDRAM com DDR (taxa de transferência dupla). Memória com o dobro da largura de banda de pico de uma SDRAM.

DMA *Direct Memory Access* – Acesso Directo à Memória. Técnica de acesso à memória sem intervenção do CPU.

DRAM Memória RAM dinâmica. Tipo de memória RAM onde é necessário proceder ao refrescamento periódico de seu conteúdo para o preservar.

FFT *Fast Fourier Transform* – Transformada Rápida de Fourier.

Fixed-point Cálculo matemático com vírgula fixa (números inteiros).

Floating-point Cálculo matemático com vírgula flutuante (números reais).

FPGA *Field Programmable Gate Array*. É um dispositivo semicondutor que contêm componentes de lógica programável e interligações programáveis com uma estrutura parecida com a do ASIC.

FSM *Finite State Machine* – Máquina de Estados Finita.

H.263 Norma de codificação de vídeo.

H.264 Norma de codificação de vídeo.

MPEG-4 AVC ver H.264.

HLD *Hardware Description Language* – Linguagem de Descrição de Hardware.

I/O Interface de Entrada (I - *input*) e Saída (O – *output*).

IDCT *Inverse Discrete Cosine Transform* – Transformada Inversa de Co-senos Discreta.

Inter Codificação de imagens de vídeo com previsão temporal ou compensação de movimento.

Intra Codificação de imagens de vídeo sem predição temporal.

LAD bus Barramento de dados, endereços e sinais de controlo entre o controlador CardBus e o bloco de elementos programáveis da WCII.

Luma O mesmo que luminância (Y).

LUT *Look-Up-Table* – Tabela de Consulta.

Macrobloco Unidade elementar de uma região de uma imagem codificada. Normalmente com dimensões 16 × 16 pixels.

Makefile Ficheiro que contém informação sobre como automatizar um processo.

MBAFF *Picture-Adaptive Frame/Field coding* – Codificação adaptável de imagem/quadro para macroblocos.

MPEG Motion Pictures Expert Group.

MPEG-2 Norma de codificação multimédia.

MPEG-4 Norma de codificação multimédia.

NOT Função lógica de negação (NOT em inglês).

OR Função lógica OU (OR em inglês).

PAFF *Picture-Adaptive Frame/Field coding* – Codificação adaptável de imagem/quadro para imagens.

PAL *Programmable Array Logic* – Conjunto de Lógica Programável. Semelhante à PLA, no entanto o plano OR é fixo. Este tipo de dispositivos contém ou podem conter multiplexadores, vários elementos XOR e o mais importante, elementos síncronos, isto é, *flip-flops*.

PC *Personal Computer* – Computador pessoal.

PCI *Peripheral Component Interconnect* – Interligação de Componentes Periféricos. É um barramento paralelo de 32 ou 64 bits síncrono (33MHz a 66MHz) criado pela Intel® e usado para interligar os diversos periféricos numa placa mãe de um PC.

PCMCIA *Personal Computer Memory Card International Association* – Associação Internacional de Cartas de Memória para Computadores Pessoais. Norma para dispositivos de informática do tamanho de um cartão de crédito desenvolvido para oferecer capacidades extra a computadores portáteis. Existem placas PCMCIA do tipo I, II e III.

Pixel *Picture element* – Elemento básico da imagem. Uma imagem é constituída por um ou mais pixel.

PLA *Programmable Logic Array* – Conjunto de Lógica Programável. É um circuito integrado constituído por um conjunto de entradas ligadas a um plano de portas AND. Essas entradas são ligadas ao plano AND de acordo com o que se pretende. As saídas do plano AND estão ligadas a

	um plano OR também ele configurável de onde se obtém a função lógica pretendida. Para cada entrada existe a opção de negação.
PSNR	*Peak to Signal Noise Ratio* – Relação Sinal Ruído de Pico que serve para medir objectivamente a qualidade de uma imagem.
RAM	*Random Access Memory* – Memória de Acesso Aleatório.
SDR	*Single Data Rate* – Taxa de Transferência Simples. A informação é transferida num só flanco do relógio em cada ciclo – ver DDR.
SDRAM	Memória DRAM síncrona.
SRAM	Memória RAM estática. Tipo de memória RAM que mantêm o seu conteúdo desde que se mantenha a alimentação.
Stream	Fluxo de dados/informação.
Verilog	ver VHDL.
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*. É uma das várias linguagens de programação HDL. Sendo actualmente em conjunto com o Verilog uma das linguagens HDL mais populares.
VLC	*Variable Length Code* – Código de Comprimento Variável.
VLD	*Variable Length Decoder* – Descodificador de Códigos de Comprimento Variável.
VLE	*Variable Length Encoder* – Codificador de Códigos de Comprimento Variável.
VO	*Video Object* – Objecto de Vídeo.
VOP	*Video Object Plane* – Plano do Objecto de Vídeo.
WCII	Placa PCMCIA do tipo II para desenvolvimento em FPGA designada por Wildcard™II desenvolvida pela Annapolis Micro Systems, Inc.
XOR	Função lógica OU EXCLUSIVO (EXCLUSIVE OR em inglês).
YCbCr	Espaço de cor composto por luminância (Y), crominância azul (Cb) e crominância vermelha (Cr).
YUV	Espaço de cor (ver YCbCr).
ZBT-SRAM	*Zero Bus Turnaround SRAM*. É uma memória SRAM síncrona que, devido a não necessitar de ciclos de espera (turnaround cycles), entre ciclos de escrita e ciclos de leitura, utiliza na totalidade a sua largura de banda.

Capítulo 1. INTRODUÇÃO

A codificação de vídeo tornou possível uma variedade de serviços que são hoje uma realidade, tais como: a televisão digital, os filmes em suportes digitais como o DVD, o *streaming* (fluxo de bits/informação) de filmes pela Internet, entre outros. Um dos aspectos para o sucesso destes serviços é a capacidade de compressão que o codificador de vídeo oferece. Quanto maior for a taxa de compressão, mais canais de TV podem ser transmitidos, rentabilizando mais a largura de banda do operador, e consequentemente melhor é a qualidade de um filme gravado em DVD.

No entanto, a grande maioria dos serviços exemplificados, usa como norma de codificação de vídeo o MPEG-2. Este formato conta com pelo menos 10 anos e, como tal, já não se adequa muito às necessidades de hoje em dia. Foi, então, imperativo desenvolver uma norma de codificação de vídeo adequada às necessidades actuais. As normas MPEG-4 [1] e MPEG-4 AVC/H.264 [2] são apontadas como sucessores do MPEG-2. Por exemplo, os novos formatos de DVD de alta definição, HD-DVD e Blu-Ray (BR), já são codificados em MPEG-4 AVC/H.264. Também, para a distribuição de vídeo na Internet já se usa em grande medida o formato MPEG-4 Visual.

Contudo, estes novos formatos de codificação, MPEG-4 AVC/H.264 e MPEG-4 Visual, necessitam de uma capacidade computacional relativamente elevada. Torna-se, portanto, necessário desenvolver módulos de elevado desempenho para se obter vídeo comprimido em tempo real. Isso pode ser desenvolvido através de software optimizado ou então com a ajuda de hardware dedicado.

Deste modo, o trabalho aqui apresentado tem como objectivo o desenvolvimento em hardware de módulos, nomeadamente em FPGA (*Field Programmable Gate Array*), de alguns dos estágios do CoDec (*enCoder/Decoder* – Codificador/Descodificador) de vídeo MPEG-4 Visual e MPEG-4 AVC/H.264. Os módulos desenvolvidos representam

tipicamente a parte de maior cálculo computacional, o que significa que estes módulos podem ser usados como processadores auxiliares na codificação de vídeo.

Assim, para melhor se compreender os módulos desenvolvidos, esta dissertação começa por descrever os conceitos de vídeo como a amostragem de vídeo, espaços de cor, formatos de amostragem (progressiva ou entrelaçada), formatos de vídeo e qualidade de imagem. Seguidamente, é mostrado o codificador de vídeo: modelo temporal, modelo espacial e codificador de entropia. É descrito o modelo híbrido de codificação usado na grande maioria dos codificadores de vídeo baseados em macroblocos o DPCM/DCT, analisando-se de forma pormenorizada a norma MPEG-4 AVC/H.264 no último capítulo alusivo à revisão dos algoritmos de codificação.

Finalmente, são descritos os módulos desenvolvidos, mostrando as suas interfaces, algoritmos, diagrama temporais e a percentagem de ocupação do módulo na FPGA. Os módulos desenvolvidos da norma MPEG-4 Visual foram:

- Módulo VLD
- Módulo IQ
- Módulo IDCT
- Módulo VLD+IQ+IDCT

Enquanto que para norma MPEG-4 AVC/H.264 foram desenvolvidos os seguintes módulos:

- Módulo *Deblocking*
- Módulo Estimação de Movimento

A dissertação termina com as conclusões deste trabalho.

Capítulo 2. CONCEITOS DE VÍDEO

Neste capítulo, apresentam-se os conceitos de vídeo fundamentais necessários para a compreensão da codificação de vídeo, assunto a ser discutido no capítulo 3. Primeiramente, descreve-se a amostragem de um sinal de vídeo, e em seguida, definem-se os espaços de cor utilizados para representar o vídeo. Mostram-se os vários formatos de amostragem e os formatos de vídeo comuns. Por fim, é analisado como é avaliada a qualidade do vídeo reconstruído, objectivamente ou subjectivamente.

2.1- Amostragem

O sinal de vídeo digital representa uma sequência de imagens reais obtidas por amostragem espacial e temporal. A amostragem espacial, ou resolução, produz uma imagem ou uma fotografia (*slide*), enquanto que a amostragem temporal, em unidades de imagens por segundo, produz uma sequência de imagens que criam uma ilusão de movimento, conforme se mostra na Figura 1.

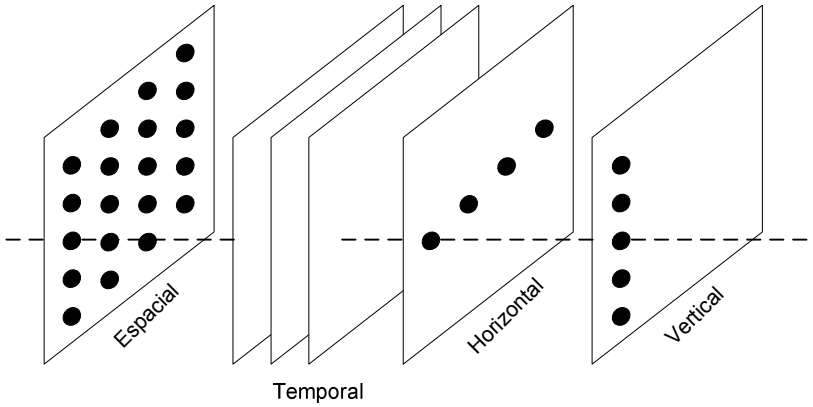

Figura 1 – Amostragem espacial e temporal.

Cada amostra espacial (os pontos negros na Figura 1) é denominada de pixel, do inglês *picture element*, e as amostras temporais são um conjunto de pixels no mesmo plano espacial. Estes pixels são representados por valores numéricos, em *bits* ou *bytes* (1byte = 8bits), que definem o brilho e a cor da amostra [3].

O número de amostras espaciais influencia a qualidade da imagem, enquanto que o número de amostras temporais influencia a qualidade da fluidez do movimento. Por exemplo, a mesma imagem, amostrada espacialmente com um número inferior de amostras apresenta um aspecto mais rugoso (efeito tipo mosaico) que uma amostragem mais densa, ou seja, com mais amostras. Na mesma sequência de imagens amostrada temporalmente com uma menor taxa de mostragem por segundo observa-se que o movimento não é tão fluído, pois verificam-se irregularidades no movimento (*choppiness* ou *flicker*), como seria com uma maior taxa de amostragem. Por exemplo, uma taxa de 10 a 20 imagens por segundo é o típico de comunicações de vídeo de baixo débito, 25 a 30 é a frequência temporal para televisão (com entrelaçamento para melhorar a qualidade do movimento) enquanto uma frequência mais alta é necessária em televisão de alta definição. Normalmente, 50 a 60 imagens por segundo dão a sensação de um movimento fluído (suave, sem "saltos").

No entanto, o aumento do número de imagens por segundo, apesar de melhorar significativamente a qualidade do vídeo, também aumenta o débito de transmissão. Uma das formas para minimizar este problema, é fazer uso de uma técnica onde a amostragem espacial não é feita por toda a imagem, amostragem progressiva, mas por campos (*fields*), i.e., amostragem entrelaçada. Durante cada intervalo da amostragem temporal só um campo da imagem é amostrado, sendo o próximo campo amostrado no intervalo seguinte de amostragem temporal. Um campo representa as linhas pares, enquanto que o outro campo representa as linhas impares, Figura 2. Como cada campo contém só metade da informação, o débito total de transmissão é inferior, isto é, igual a metade do débito para uma amostragem progressiva com o mesmo número de imagens por segundo. No entanto, o vídeo torna-se mais fluído em comparação com a mesma sequência com amostragem progressiva [3]. Por exemplo, o sistema PAL de televisão é composto por 50 campos por segundo, amostragem entrelaçada, contudo o movimento é mais suave do que uma amostragem progressiva a 25 imagens por segundo. Para situações de cenas onde não há grande (ou mesmo nenhum) movimento nas sequências, por exemplo a filmagem de um

objecto imóvel onde nada se move, a amostragem progressiva apresenta uma imagem mais "suave".

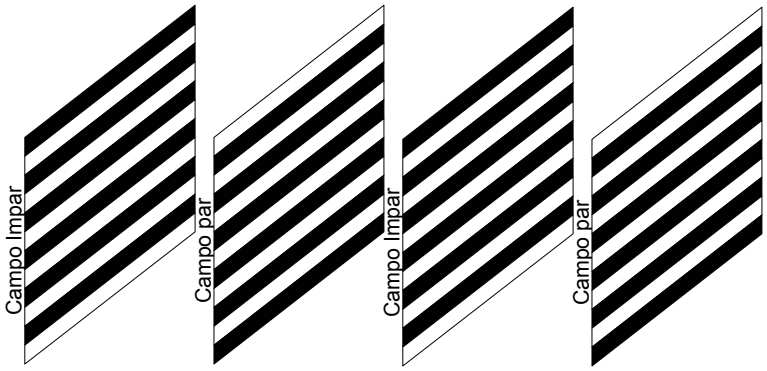

Figura 2 – Sequência de vídeo entrelaçado.

2.2- Espaço de cor (*Colour Space*)

Em aplicações onde só se pretende vídeo monocromático, cada pixel contém apenas a informação do brilho, i.e., valores mais elevados representam mais brilho e vice-versa. Mas quando se pretende vídeo a cores, é necessário de alguma forma representar essa cor. À forma como se representa o brilho e a cor chama-se de espaço de cor, ou em inglês, *Colour Space*. O espaço de cor não é mais do que uma representação matemática de um conjunto de cores. Existindo vários espaços de cor sendo cada um deles mais apropriado para uma determinada aplicação.

Referem-se aqui somente os espaços de cor relevantes para este trabalho.

2.2.1 RGB

Este espaço de cor é representado por um sistema de coordenadas cartesiano tridimensional onde cada eixo representa as três cores primárias aditivas: Vermelho (R - *Red*), Verde (G – *Green*) e Azul (B – *Blue*) [4]. Quantidades diferentes de cada uma destas componentes definem uma cor. Um vector neste espaço de cor que tenha quantidades iguais de cada componente representa várias intensidades de brilho, ou seja, escalas ou tonalidades de cinzento, Figura 3.

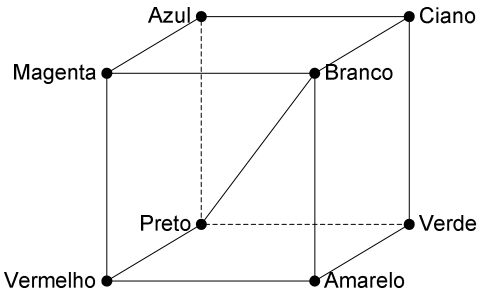

Figura 3 – RGB.

Cada pixel da imagem, usando o RGB, é representado por três valores, cada um corresponde ao valor de R, G e B respectivamente, da amostra espacial da imagem. O RGB é o mais apropriado para captura e apresentação de imagens coloridas uma vez que os ecrãs usam as componentes de cor R, G e B para representar imagens bem como as câmaras que usam *Charge Couple Devices* (CCDs), com os sensores R, G e B para capturarem imagens. No entanto, o RGB não é muito eficiente para imagens reais (naturais). Cada componente de cor usa a mesma largura de banda, mesmo que a imagem ou vídeo capturado apresente muito pouca informação numa das três componentes de cor. E caso seja necessário efectuar algum ajuste no brilho do vídeo, é necessário fazer ajustes em todas as componentes RGB, atrasando assim o processamento da imagem.

Desta forma, existe outro espaço de cor que resolve alguns dos defeitos do RGB denominado o espaço de cor YCbCr.

2.2.2 YCbCr

O espaço YCbCr, ou designado algumas vezes YUV, explora o facto do sistema visual humano ser mais sensível ao brilho do que à cor.

Neste espaço o brilho ou luminância, Y, ou vulgarmente conhecido por *luma* do inglês *lumimance*, é representado pela seguinte equação [4]:

$$Y = k_r R + k_g R + k_b R, \text{onde } \mathrm{k_r}, \mathrm{k_g} \text{ e } \mathrm{k_b} \text{ são factores de peso} \qquad \textbf{(2.2.1)}$$

A informação da cor é representada pela crominância, ou *chroma* do inglês *chrominance*. Cada crominância é a diferença de cor entre as componentes R, G e B e a luminância Y:

$$\begin{aligned} C_b &= B - Y \\ C_r &= R - Y \\ C_g &= G - Y \end{aligned} \qquad \textbf{(2.2.2)}$$

Verifica-se que para especificar o espaço de cor YCbCr são necessárias quatro componentes. Uma componente define o brilho, Y, e as restantes componentes definem as cores: crominância azul, C_b (*chroma blue*), crominância vermelha, C_r (*chroma red*), e crominância verde, C_g (*chroma green*). Apesar de existir mais uma componente do que o espaço RGB, o espaço de cor YUV é mais eficiente. Vejamos, o somatório $C_b + C_r + C_g$ é uma constante, logo só duas componentes de cor necessitam de ser guardadas e/ou transmitidas, C_b e C_r, além da luminância, Y. Mas como o olho humano é mais sensível ao brilho (*luma*, Y), as restantes componentes podem ser transmitidas com menor resolução/largura de banda. Verifica-se, então, que é possível reduzir a largura de banda necessária para a mesma imagem usando o espaço de cor YCbCr do que seria necessária usando o espaço de cor RGB. Tal é possível, porque no espaço de cor YCbCr é possível as componentes de cor terem uma resolução diferente da componente de brilho ao contrário do verificado no espaço RGB onde as três componentes devem ter todas a mesma resolução. Desta forma, obtém-se uma forma simples de compressão usando o espaço de cor YCbCr. Note-se que, as diferenças entre uma imagem usando o espaço de cor RGB e uma imagem usando o espaço de cor YCbCr para um observador casual, ou não experiente (sem ter o "olho treinado"), não existem ou são muito ténues.

As seguintes equações mostram como se converte de RGB para YCbCr e vice-versa.

$$\begin{aligned} & k_b + k_r + k_g = 1 \\ & Y = k_r R + (1 - k_b - k_r) G + k_b B \\ & C_b = \frac{0.5}{1 - k_b}(B - Y) \\ & C_r = \frac{0.5}{1 - k_r}(R - Y) \end{aligned} \qquad \textbf{(2.2.3)}$$

$$
\begin{aligned}
R &= Y + \frac{1-k_r}{0.5} C_r \\
G &= Y - \frac{2k_b(1-k_b)}{1-k_b-k_r} C_b - \frac{2k_r(1-k_r)}{1-k_b-k_r} C_r \\
B &= Y + \frac{1-k_b}{0.5} C_b
\end{aligned}
\tag{2.2.4}
$$

Segundo as recomendações ITU-R BT.601 [5], dado valor de $k_b = 0.114$ e $k_r = 0.299$ obtêm-se as seguintes equações:

$$
\begin{aligned}
Y &= 0.299R + 0.587G + 0.114B \\
C_b &= 0.564(B-Y) \\
C_r &= 0.713(R-Y)
\end{aligned}
\tag{2.2.5}
$$

$$
\begin{aligned}
R &= Y + 1.402C_r \\
G &= Y - 0.344C_b - 0.714C_r \\
B &= Y + 1.772C_b
\end{aligned}
\tag{2.2.6}
$$

2.3- Formatos de amostragem

Numa imagem RGB, as três componentes têm a mesma resolução, ou seja, existe uma amostra por cada componente da amostragem da imagem. No entanto, no espaço YCbCr é possível ter-se resoluções diferentes por cada componente Y, Cb e Cr, tal como foi referido anteriormente [3].

Existe uma nomenclatura, Y:Cb:Cr, que identifica cada um destes formatos. O formato de amostragem 4:4:4 indica que existe uma amostra de cada componente para cada pixel da imagem, Figura 4.

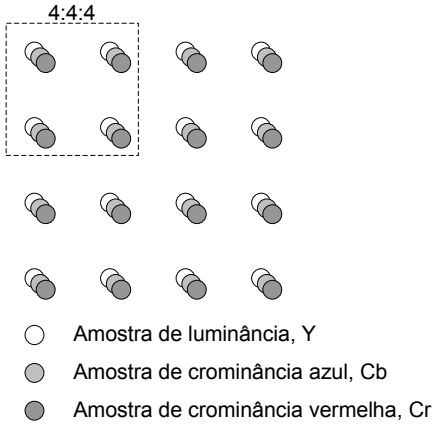

Figura 4 – Formato de amostragem 4:4:4.

O formato 4:2:2 (conhecido por YUY2) indica que por cada 2 pixels horizontais de Y existe um pixel de crominância mas a resolução vertical de Y e Cb, Cr são iguais, como se pode observar na Figura 5.

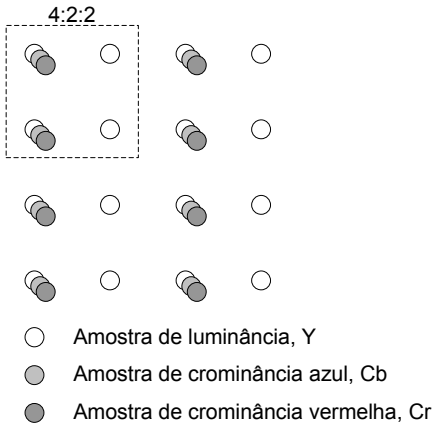

Figura 5 – Formato de amostragem 4:2:2.

O formato 4:1:1 (YUV12), usado em algumas aplicações de compressão de vídeo de DV (*Digital Video*) e vídeo de consumo, indica que por cada 4 pixels horizontais de Y existe 1 pixel de Cb e Cr mantendo a mesma resolução vertical.

Outro formato muito usado é o formato 4:2:0 (ou YV12) que é usado em DVDs e em televisão digital. Neste formato, a resolução das componentes de crominância é metade da resolução da componente Y tanto na horizontal como na vertical, tal como mostra a Figura 6. A nomenclatura deste formato é um pouco confusa, uma vez que o 0, zero, poderia indicar que a componente Cr está ausente, tal é erróneo.

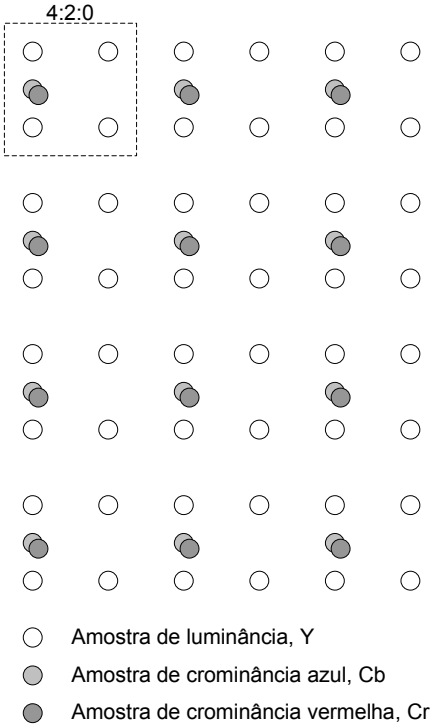

Figura 6 – Formato de amostragem 4:2:0.

A tabela seguinte mostra o número de bits necessários para codificar uma imagem com resolução 720 × 576 pixels e onde cada componente Y, Cb e Cr é representada por 8 bits.

Tabela 1 – Número de bits para os formatos 4:4:4 e 4:2:0 de uma imagem 720×576 pixels.

Formato	Número de bits
4:4:4	720 × 576 × 8 × 3 = 9953280 bits
4:2:0	(720 × 576 × 8) + (360 × 288 × 8 × 2) = 4976640 bits Metade dos bits do formato 4:4:4

Como se pode observar na Tabela 1, o número de bits necessários para a mesma imagem no formato 4:2:0 é metade do valor no formato 4:4:4. Também se verifica que o formato 4:4:4 necessita exactamente do mesmo número de bits que a mesma imagem no formato RGB, i.e., o formato 4:4:4 também é referido como R:G:B vídeo. O formato 4:2:0 é também normalmente descrito como 12 bits por pixel uma vez que em 4:4:4 são necessárias 12 amostras o que perfaz 12 × 8 bits = 96 bits e em média 96/4 = 24 bits/pixel. Portanto, em 4:2:0 só 6 amostras são necessárias, ou seja, 6 × 8 bits = 48 bits e em média 48/4 bits = 12 bits.

Quando se procede à apresentação da imagem, conversão YCbCr para RGB, nos formatos onde a resolução da crominância é inferior à luminância, procede-se a uma interpolação para se obterem os pixels que faltam.

Para que se possa trabalhar com o sinal de vídeo, é necessário definir a resolução do mesmo, ou seja, além de se definirem os espaços de cor e os formatos de amostragem do vídeo, é necessário definir os formatos de resolução da imagem.

2.4- Formatos de vídeo

Existe um conjunto de formatos que definem a resolução das imagens de um vídeo. Ao nome desse conjunto de formatos chama-se *Common Intermediate Format* (CIF) e definem as seguintes resoluções [3]:

Tabela 2 – Resoluções dos formatos CIF.

Formato	Resolução da luminância (Horiz. × Vert.)	Bits por imagem no formato 4:2:0 e 8 bits por amostra
Sub-QCIF	128 × 96	147456
Quarter CIF (QCIF)	176 × 144	304128
CIF	352 × 288	1216512
4CIF	704 × 576	4866048

Cada um destes formatos está adaptado a uma determinada aplicação. Por exemplo, o formato 4CIF é usado em DVD enquanto que os formatos QCIF ou SQCIF são mais apropriados para aplicações multimédia móveis.

2.5- Qualidade de vídeo

É necessário medir e quantificar a qualidade de uma imagem de vídeo para comparar, especificar e analisar um determinado sistema de vídeo. No entanto, essa não é uma tarefa fácil uma vez que a qualidade de uma imagem de vídeo normalmente é subjectiva e não objectiva. Para um dado observador, a qualidade de imagem de um determinado vídeo pode ser boa, enquanto que para um outro pode ser só razoável.

A recomendação ITU-R Recomendation BT.500 [6] indica um método para se fazer uma análise subjectiva. Esse método, *Double Stimulus Continuous Quality Scale* (DSCQS), sugere que um observador seja sujeito às sequências de imagens de vídeo A e B. As imagens do vídeo A e B são exibidas aleatoriamente sem que o observador saiba de qual fonte provém a imagem. Além disso, é pedido ao observador que atribua um valor numa escala para cada sequência. No fim, os valores são normalizados e é atribuída uma pontuação a cada sequência [4].

Apesar deste método ser aceitável para testes subjectivos, este teste apresenta alguns problemas práticos. Por exemplo, se o observador for uma pessoa habituada a encontrar os problemas em codificadores de vídeo (um "*expert*"), este tentar encontrá-los e vai atribuir uma pontuação não isenta. Por outro lado, é necessário usar uma vasta gama de observadores para se obterem resultados fiáveis, o que se traduz em custos acrescidos e muito tempo dispendido.

Outra forma de avaliar a qualidade das imagens de vídeo, é usar testes objectivos. Estes permitem avaliar a qualidade das imagens, usando algoritmos.

Um dos métodos objectivos é a relação sinal/ruído de pico, ou em inglês *Peak Signal to Noise Ratio* (PSNR):

$$PSNR_{dB} = 10\log_{10}\frac{(2^n-1)^2}{MSE} \qquad \textbf{(2.5.1)}$$

$$MSE = \frac{\sum[f(i,j)-F(i,j)]^2}{N^2} \qquad \textbf{(2.5.2)}$$

A PSNR depende do erro quadrático médio, MSE, da imagem original, F(i,j), e a imagem de análise, f(i,j), do vídeo relativamente $(2^n - 1)^2$ que é o valor máximo possível do sinal ao quadrado da imagem onde *n* é o número de bits/amostra.

Para se utilizar este método é necessário ter-se a imagem original (antes da codificação) o que pode ser um problema, uma vez que nem sempre é possível ter essa

imagem. Também o PSNR nem sempre dá os mesmos resultados que as análises de qualidade subjectivas. Em alguns casos, onde a qualidade da imagem atribuída pelo PSNR é boa, uma análise subjectiva indica fraca qualidade, porque o tipo de imagem influencia o observador. Casos típicos são os exemplos de imagens onde o rosto de uma pessoa, na imagem, apresenta alguma distorção devido ao processo de codificação, apesar de ser baixa a distorção e como tal o PSNR ser elevado, um observador julgará como má, a qualidade de imagem, porque o observador humano dá mais importância à qualidade do rosto humano na imagem, mais do que ao resto da imagem.

Existem outros métodos objectivos para a avaliação da qualidade das imagens. Neste trabalho, não será abordado mais nenhum outro método.

Após o estudo dos formatos de amostragem, formatos de vídeo e os espaços de cores, a análise dos métodos de codificação por ser melhor entendida. Estes conceitos são fundamentais para entender o próximo capítulo onde se descrevem os métodos de codificação usados actualmente.

Capítulo 3. CODIFICAÇÃO DE VÍDEO

O processo de codificação de vídeo tem como objectivo comprimir um sinal de vídeo para posterior armazenamento/transmissão. O processo inverso, a descodificação, descomprime o vídeo comprimido para o seu formato original. Um sistema que faça a codificação e descodificação de vídeo designa-se por CoDec, que em inglês significa enCOder/DECoder (codificador/descodificador) [7], Figura 7. Com a compressão, consegue-se reduzir o número de bits necessários para representar a mesma informação sem compressão [4].

Figura 7 – CoDec de vídeo.

A codificação pode comprimir o sinal de vídeo, sem qualquer perda de informação, o que normalmente se chama compressão sem perdas (*lossless compression*), em oposição a compressão com perdas (*lossy compression*).

Quando se utiliza *lossless compression*, a informação reconstruída é exactamente igual à informação original. Apesar, de o vídeo descodificado apresentar a mesma qualidade que o vídeo original, utilizando este tipo de compressão (codificação), a taxa de compressão é relativamente baixa. Por exemplo, a máxima taxa de compressão *lossless* é utilizando a norma JPEG-LS [8] que obtém uma taxa de 3 a 4:1.

A codificação *lossy* obtém muito boas taxas de compressão, removendo informação redundante, sem que a degradação da qualidade do vídeo seja visualmente perceptível. Contudo, utilizando este tipo de codificação, o vídeo após a descodificação não é

exactamente igual ao vídeo original. Essa informação redundante pode ser eliminada tanto espacialmente, na imagem, como temporalmente, numa sequência de imagens. Por exemplo, no domínio temporal existe uma grande correlação entre imagens (*frames*) adjacentes do vídeo, especialmente se a taxa de amostragem temporal for elevada. Por exemplo, uma cena onde não existe movimento apreciável. As várias imagens adjacentes do vídeo são muito similares entre si. No domínio espacial encontra-se redundância em imagens onde o fundo é, por exemplo, uma parede branca. Numa vizinhança de alguns pixels da parede o brilho e cor são muito idênticos, logo se pode suprimir alguma informação sem que haja grande diferença visual entre a imagem comprimida e a imagem não comprimida (original).

Um codificador de vídeo consiste em três principais unidades funcionais: modelo temporal (*temporal model*), modelo espacial (*spatial model*) e codificador de entropia (*entropy encoder*), Figura 8, [4].

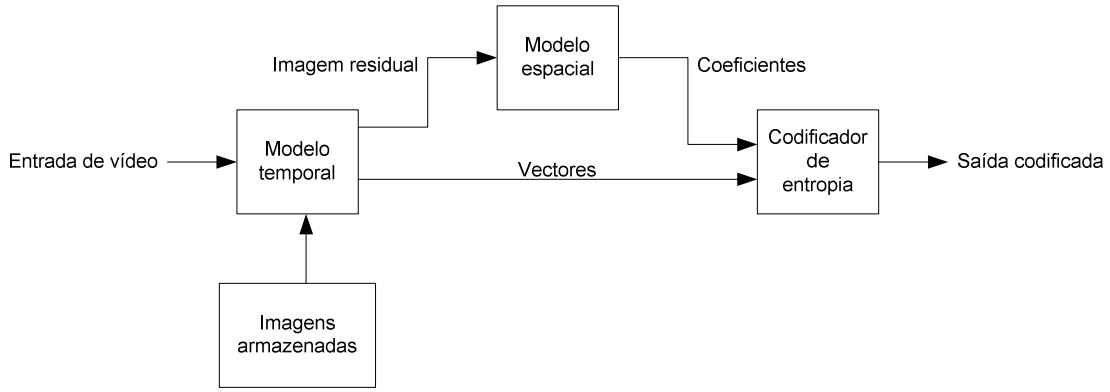

Figura 8 – Diagrama de blocos dum codificador de vídeo.

3.1- Modelo temporal

O modelo temporal tem como função reduzir a redundância temporal. Esta redução obtém-se encontrando similaridades entre imagens consecutivas, fazendo muitas vezes uma predição, prognóstico, da imagem actual do vídeo. Em alguns tipos de CoDec de vídeo, MPEG-4 AVC/H.264 e MPEG-4 Visual, esta predição é feita com base em imagens futuras ou anteriores do vídeo. A eficácia desta predição pode ser melhorada fazendo compensações para diferenças entre imagens (*frames*), chamado predição por compensação de movimento (*motion compensated prediction*).

O bloco do modelo temporal recebe uma imagem de vídeo não comprimida e devolve uma imagem residual, a diferença entre a imagem actual e a imagem de prognóstico (predição), e um conjunto de vectores que descrevem de que forma o movimento foi compensado. Esta imagem residual possui menor energia do que a imagem actual.

O método de subtrair a imagem actual com uma imagem anterior é o mais simples, mas a redução de energia não é óptima, i.e., a imagem residual ainda contém muita informação. A maior parte da energia na imagem residual deve-se ao movimento de objectos que existe entre as duas imagens. Uma melhor predição pode ser obtida com compensação de movimento.

É possível estimar a trajectória de cada pixel entre imagens de vídeo sucessivas produzindo um campo de trajectórias de pixels conhecido como fluxo óptico (*optical flow*).

Se o fluxo óptico for suficientemente bom, então é possível fazer uma previsão dos pixels da imagem actual movendo simplesmente cada pixel para a posição adequada de acordo com o fluxo óptico.

Isto levanta um problema, pois é necessário obter-se vectores de movimento para cada um dos pixels da imagem, o que pode ser computacionalmente complicado ou mesmo impossível. Uma forma de contornar esse problema, consiste em fazer uma compensação de movimento para um dado conjunto de pixels, blocos. A isto chama-se compensação e estimação de movimento, baseada em blocos e funciona da seguinte forma [4]:

1. Procurar uma área no *frame* de referência (passado ou futuro) para encontrar a região M × N que coincide (*match*). Faz-se comparando o bloco M × N no *frame* actual em todas ou só em algumas posições possíveis da região M × N dentro da área de procura. Normalmente, um critério para o *match* é a energia contida no resíduo formado pela diferença entre a região candidata e o bloco M × N actual. A região candidata que minimiza a energia é a escolhida como *match*. Ao processo, dá-se o nome de estimação de movimento (*motion estimation*).
2. A região escolhida passa a ser a predição do bloco M × N actual e é subtraída do bloco actual M × N para se obter o bloco M × N residual, ou seja, compensação de movimento (*motion compensation*).
3. O bloco residual é comprimido e transmitido, como também é transmitida a informação acerca da posição da região candidata (vector de movimento).

Cada macrobloco (*macroblock*) corresponde a uma região de 16 × 16 pixels e é a unidade básica usada para a compensação da estimação de movimento em várias normas tais como H.261, H.263, MPEG-4 AVC/H.264, MPEG-1 Video, MPEG-2 Video e MPEG-4 Visual.

A estimação de movimento do macrobloco obtém-se, quando é encontrada uma região de 16 × 16 pixels da imagem de referência que mais se aproxima de um determinado critério de igualdade com o actual macrobloco (*best match*).

Após se ter encontrado a região de *best match*, essa região é subtraída do macrobloco, onde se produz um macrobloco residual que é codificado e transmitido em conjunto com o vector de movimento que descreve a posição do *best match*. Este processo denomina-se compensação de movimento.

No codificador, este macrobloco residual codificado é descodificado e adicionado à região de *best match* para reconstruir um macrobloco que é armazenado para posterior predição de compensação de movimento. Também é necessário que o codificador use o mesmo macrobloco residual descodificado para a reconstrução, para que tanto o codificador como o descodificador usem a mesma imagem de referência para a compensação de movimento. Caso tal não se verifique, o descodificador iria usar uma

imagem de referência diferente do codificador e o vídeo descodificado degradar-se-ia com o passar o tempo.

O codificador pode codificar, sem usar compensação de movimento, e a isto chama-se modo Intra (*Intra mode*), ou então, usando compensação de movimento e chama-se modo Inter (*Inter mode*). A escolha depende do tipo de imagens. Por exemplo, quando no vídeo se muda de uma cena para outra, há uma mudança significativa das imagens, o modo Intra pode ser mais eficiente.

Apesar de o tamanho do bloco ser 16 × 16 pixels, nem sempre este é o tamanho ideal. Verifica-se que blocos com tamanho inferior produzem melhores compensações de movimento.

Além de a compensação de movimento ser melhorada, a complexidade computacional também é aumentada. São necessárias mais buscas pelo melhor candidato (*best match*). Além disso, mais vectores de movimento têm que ser transmitidos. Com o aumento de mais vectores de movimento, mais bits têm que ser transmitidos, o que em certa medida, pode minorar a vantagem de uma imagem residual, energicamente mais fraca. A solução passa por adaptar o tamanho do bloco às características da imagem. Zonas onde a imagem é mais lisa, homogénea, devem-se usar blocos maiores, por seu lado em zonas onde existe mais detalhe, devem-se usar blocos menores. A norma MPEG-4 AVC/H.264 usa tamanho de blocos variáveis.

Uma situação que ocorre com o movimento dos objectos no vídeo é que eles raramente se movem de pixel para pixel, mas sim de uma fracção de pixel da imagem. Nestas situações, uma melhor previsão pode ser obtida interpolando a imagem de referência até ao nível de sub-pixel, antes de proceder à procurar do melhor candidato.

Um algoritmo usando a estimativa de movimento para uma resolução sub-pixel funciona da seguinte forma:

1. Procurar o melhor candidato sem usar a resolução sub-pixel.
2. Na zona onde se encontrou o melhor candidato anterior, encontrar um novo melhor candidato mas agora com uma resolução *half-pixel* (dobro da resolução original).
3. Após se ter obtido um novo melhor candidato, voltar a encontrar outro melhor candidato na zona do melhor candidato anterior, mas agora com resolução duplicada da anterior, *quarter-pixel* (quádrupla da original).
4. Se necessário, voltar a encontrar um novo melhor candidato no melhor candidato anterior duplicando a resolução, *eight-pixel* (oito vezes maior que a original).
5. O melhor candidato final é subtraído do macrobloco corrente.

Em geral, quanto maior é a interpolação melhor é a compensação do movimento. Não obstante, o com o aumento da interpolação é necessário enviar mais informação o que faz com que a compressão seja menor.

3.2- Modelo espacial

A função do modelo espacial, ou modelo de imagem (*image model*), tem como objectivo descorrelacionar a imagem (ou a imagem residual) e converter essa informação de forma a ser eficientemente comprimida pelo codificador de entropia.

As imagens de uma cena natural possuem um alto nível de correlação entre amostras vizinhas da imagem, não sendo portanto difícil a sua compressão. Fazendo a auto correlação bidimensional (2D) entre um pixel e outro pixel deslocado de algumas posições, verifica-se que assim que o pixel se desloca da posição original, o declive da correlação não é abrupto. Isto mostra a correlação que existe entre as amostras adjacentes. No entanto, a função de auto correlação 2D de uma imagem residual, onde houve compensação de movimento, mostra que o declive da curva é muito acentuado, ou seja, as amostras adjacentes são pouco correlacionadas.

O modelo espacial aplica uma transformada à imagem que não correlaciona e compacta a informação, seguido duma quantificação que reduz a precisão dos dados da transformada, e por fim reordena esses dados.

3.2.1 Transformada

A transformada num CoDec de vídeo converte os dados, informação, da imagem residual noutro domínio [9]. O tipo de transformada a ser usada depende dos seguintes factores:

- Os dados no domínio da transformada devem estar separados em componentes que tenham interdependência mínima, i.e., não correlacionados. E devem estar compactos, ou seja, a maior parte da energia deve estar concentrada e em poucos de valores.

- A transformada deve ser reversível.

- Por questões práticas a transformada escolhida deve ser computacionalmente implementável.

A transformada mais "popular" é a Transformada Co-senos Discreta (DCT – *Discrete Cosine Transform*) [10]. Esta transformada opera sobre blocos com dimensão N × N da imagem residual, sendo portanto a imagem processada por blocos. Este tipo de transformada, funciona bem para compressão de compensação de movimento de imagem residual baseado em blocos, mas tende a criar artefactos nas fronteiras desses blocos ("*blockiness*").

Existem também transformadas que operam sobre toda a imagem como a *Discrete Wavelet Transform* (DWT ou *wavelet*) [10] usada por exemplo num dos perfis do CoDec MPEG-4 Visual. Porém, a DWT tem requisitos maiores de memória, devido ao facto de processar a imagem por um todo, ao invés de blocos, e por isso não "encaixa" bem com compensações de movimento por blocos [4].

A transformada DCT, ou FDCT, é dada pela equação:

$$Y = AXA^T \tag{3.2.1}$$

e a inversa, IDCT, ou DCT^{-1}, por:

$$X = A^T YA \tag{3.2.2}$$

onde X é uma matriz com os valores das amostras e Y uma matriz com os valores dos coeficientes.

A matriz A é a matriz de transformação N×N:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} \tag{3.2.3}$$

$$C_i = \begin{cases} \sqrt{1/N} & , \quad i = 0 \\ \sqrt{2/N} & , \quad i > 0 \end{cases} \tag{3.2.4}$$

A equação (3.2.1) e a equação (3.2.2) podem ser reescritas na forma de somatórios:

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \tag{3.2.5}$$

$$X_{xy} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} C_x C_y Y_{xy} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \tag{3.2.6}$$

Os coeficientes obtidos após a transformada, Y, representam os dados do bloco da imagem no domínio da DCT. Estes coeficientes são denominados de pesos de um conjunto de padrões de teste base (*basis patterns*) [9]. Uma imagem pode ser reconstruída, combinando todos os padrões de testes N × N, sendo cada base multiplicada pelo coeficiente apropriado [4].

Apesar de a equação (3.2.3) (A_{ij}) ser relativamente complexa, se atendermos ao facto de a função co-seno ser simétrica e periódica, então a matriz formada por A pode ser simplificada. Por exemplo, para uma DCT 4 × 4 a matriz A pode ser reduzida à seguinte matriz:

$$A = \begin{bmatrix} \sqrt{1/2}\cos 0 & \sqrt{1/2}\cos 0 & \sqrt{1/2}\cos 0 & \sqrt{1/2}\cos 0 \\ \sqrt{1/2}\cos\frac{\pi}{8} & \sqrt{1/2}\cos\frac{3\pi}{8} & \sqrt{1/2}\cos\frac{5\pi}{8} & \sqrt{1/2}\cos\frac{7\pi}{8} \\ \sqrt{1/2}\cos\frac{2\pi}{8} & \sqrt{1/2}\cos\frac{6\pi}{8} & \sqrt{1/2}\cos\frac{10\pi}{8} & \sqrt{1/2}\cos\frac{14\pi}{8} \\ \sqrt{1/2}\cos\frac{3\pi}{8} & \sqrt{1/2}\cos\frac{9\pi}{8} & \sqrt{1/2}\cos\frac{15\pi}{8} & \sqrt{1/2}\cos\frac{21\pi}{8} \end{bmatrix} \quad \textbf{(3.2.7)}$$

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad \textit{sendo} \quad \begin{array}{c} a = 1/2 \\ b = \sqrt{1/2}\cos\frac{\pi}{8} \\ c = \sqrt{1/2}\cos\frac{3\pi}{8} \end{array} \quad \textbf{(3.2.8)}$$

Desta forma o cálculo da DCT é computacionalmente "mais leve" uma vez que basta calcular o valor de dois co-senos.

À primeira vista, a vantagem de usar uma DCT não é óbvia. No caso de blocos 4×4, têm-se 16 valores de amostras e após a DCT obtêm-se 16 coeficientes. Assim que o bloco 4×4 da imagem residual é reconstruído de um subconjunto dos coeficientes, verifica-se a vantagem de usar a DCT. O exemplo seguinte mostra essa vantagem:

- Os valores e os coeficientes dum determinado bloco 4 × 4 de imagem residual e a sua DCT (com arredondamento até à primeira casa decimal), respectivamente, estão representados na Figura 9.

85	109	124	126
64	103	126	126
50	93	122	107
46	76	57	41

Original

363.7	-57.1	-41.2	-5.3
79.5	-26.1	10.4	4.9
-31.7	34.7	7.2	-7.1
14.9	-9.5	-2.9	6.1

Coeficientes DCT

Figura 9 – Coeficientes da DCT de um bloco 4×4.

- Fazendo todos os coeficientes iguais a zero, excepto o mais significativo (coeficiente 0,0, chamado de coeficiente DC), e aplicando seguidamente a IDCT, obtém se a Figura 10-a).

- Se for calculada a IDCT para os dois coeficientes mais significativos, obtêm-se Figura 10-b).

- Com 3 e 5 coeficientes antes da IDCT, obtém-se a Figura 10-c) e a Figura 10-d) respectivamente.

91	91	91	91
91	91	91	91
91	91	91	91
91	91	91	91

a) 1 coeficiente

117	117	117	117
102	102	102	102
117	117	117	117
102	102	102	102

b) 2 coeficientes

128	122	112	106
90	97	106	113
69	75	85	92
76	70	60	54

c) 3 coeficientes

134	120	111	111
87	91	100	110
72	82	91	95
71	71	62	48

d) 5 coeficientes

Figura 10 – Reconstrução a partir de a) um, b) dois, c) três e d) cinco coeficientes.

A redução do número de coeficientes é feita através do quantificador que reduz os coeficientes de valores insignificantes, tal como se verá mais à frente. Quanto maior for o

passo de quantificação mais coeficientes são eliminados. Ora, a vantagem de se terem poucos coeficientes, implica um aumento da taxa de compressão obtido pelo codificador de entropia, porque a redundância de informação é maior. Contudo, uma redução elevada de coeficientes pode implicar uma apreciável redução da qualidade da imagem. Isto é, quantos mais coeficientes forem usados, mais o resultado da IDCT se aproxima da imagem original, uma vez que a matriz usada pela IDCT é o mais a similar possível à matriz produzida pela DCT.

3.2.2 Quantificador

O quantificador tem como objectivo converter uma escala de valores numa outra escala menor [11]. Desta forma, deve ser possível representar essa nova escala com menos bits do que a escala original, uma vez que a escala é menor.

Existem dois tipos de quantificadores: quantificador numérico (*scalar quantiser*) e quantificador vectorial (*vector quantiser*) [4].

O quantificador numérico converte uma amostra do sinal de entrada num valor quantificado na saída. Por exemplo, a conversão de valores reais em valores naturais. Esta conversão é uma conversão com perdas (*lossy*), uma vez que não é possível saber o valor exacto do valor real através do valor natural.

Um quantificador uniforme é definido da seguinte forma:

$$\begin{aligned} FQ &= round(\frac{X}{QP}) \\ Y &= FQ \cdot QP \end{aligned} \qquad \textbf{(3.2.9)}$$

A QP chama-se tamanho do degrau (passo) de quantificação. Todos os valores quantificados estão espaçados uniformemente em intervalos iguais a QP.

Existem quantificadores escalares lineares e não lineares, Figura 11.

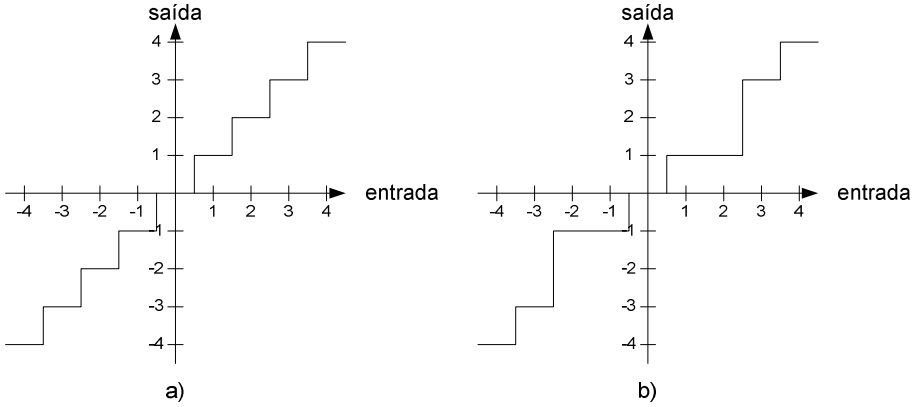

Figura 11 – Quantificador a) linear e b) não linear.

No CoDec de vídeo o processo de quantificação é feito em duas partes: quantificador directo, FQ (*forward quantiser*), e quantificador inverso IQ (*inverse quantiser*) [7]. Uma vez que a quantificação não é reversível, é usual chamar-se ao FQ *scaler* e ao IQ *rescaler*.

Quanto maior for o tamanho do QP maior será a compressão, mas pior será a qualidade de imagem e vice-versa.

O FQ deve ser desenhado para converter os valores insignificantes da DCT para zero, e ao mesmo tempo manter um reduzido número de valores significativos. A saída típica de um quantificador é um conjunto de valores (*array*), contendo quase só zeros e escassos coeficientes.

O quantificador vectorial converte um conjunto de amostras, como as amostras de um bloco da imagem, num só valor chamado de *codeword*, palavra de código. Os vários conjuntos de vectores guardados pelo codificador, bem como pelo descodificador, designam-se por *codebook*, registo de códigos.

Uma aplicação típica do quantificador vectorial à compressão da imagem é a seguinte [4]:

1. Dividir a imagem original em regiões.
2. Escolher um vector do *codebook* que se assemelhe o mais possível à região escolhida.
3. Transmitir um índice do vector escolhido para o descodificador.
4. No descodificador, reconstruir uma cópia aproximada, porque se trata de um processo de quantificação da imagem original, utilizando o índice fornecido.

Para implementar um quantificador vectorial, deve ser tomado em consideração o tipo de *codebook*, bem como métodos eficientes para procurar os vectores no *codebook*.

3.2.3 Reordenação e codificador de zeros.

À saída do quantificador obtém-se um conjunto de valores (*array*) que é constituído por escassos coeficientes diferentes de zero e muitos coeficientes nulos. Torna-se necessário proceder à reordenação desse *array*, agrupando os valores não nulos, e fazer uma representação eficiente do número de coeficientes nulos [4].

A forma como são reordenados os valores obtidos do quantificador depende da distribuição dos coeficientes não nulos da DCT. Os coeficientes mais significativos da DCT de um bloco de imagem residual estão normalmente em redor do coeficiente DC (0,0), ou seja, ocupam as posições de baixa frequência. Se for calculada a probabilidade da posição de cada coeficiente não nulo duma imagem residual, verifica-se que existe um aglomerado de coeficientes não nulos em torno do coeficiente DC e que existe uma distribuição aproximadamente simétrica tanto horizontalmente como verticalmente.

No caso do vídeo não progressivo, entrelaçado, a distribuição dos coeficientes não nulos é "enviesada", mais coeficientes não nulos estão na margem esquerda do gráfico da probabilidade. Isto ocorre porque as imagens por campos têm uma componente mais forte de alta-frequência no eixo vertical, devido à sub amostragem vertical.

Devido a estas características, a forma de reordenar os valores da DCT, após a quantificação, é fazer um varrimento em ziguezague dos coeficientes, Figura 12-a). Para vídeo entrelaçado, o processo de varrer em ziguezague os coeficientes, não é o ideal. Uma forma mais eficiente é proceder à varredura tal como mostra a Figura 12-b) [7].

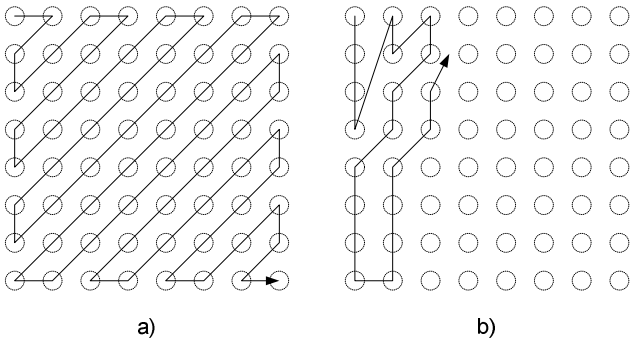

Figura 12 – Varrimento em ziguezague para a) quadro e b) campo.

Após o varrimento, o processo de reordenação, é necessário proceder à codificação da sequência de coeficientes obtida.

Uma forma de codificar essa sequência é a codificação *Run-Level*. É usada uma representação da forma (*run*, *level*), onde *run* representa o número de coeficientes nulos que precedem um coeficiente não nulo, e *level* a magnitude desse coeficiente não nulo. Por exemplo a sequência 10, 0, 0, -2, 8, 0, 0, 0, 0, 0, 0, -2, … é codificada da seguinte forma: (0, 10), (2, -2), (0, -8), (0, 6), (6, -2), …. Dado que o processo de quantificação faz com que os coeficientes da DCT de alta-frequência sejam normalmente nulos, após a reordenação, obtém-se uma sequência de valores que termina com um grande número de zeros [4]. Para tornar mais eficiente o processo de codificação, usa-se uma outra representação (*run*, *level*, *last*), onde *run* e *level* têm as mesmas funções e *last* indica o fim dos coeficientes não nulos. A sequência anterior fica agora codificada da seguinte forma: (0, 10, 0), (2, -2, 0), (0, -8, 0), (0, 6, 0), (6, -2, 1), onde o 1 na posição de *last* indica o último coeficiente não nulo.

3.3- Codificador de entropia

A função do codificador de entropia é converter uma série de símbolos num fluxo de bits (*bitstream*) comprimidos, apropriado para a transmissão/armazenamento [4].

Num codificador de vídeo, essa série de símbolos é composta pelos coeficientes da transformada quantificados e reordenados, vectores de movimento, marcadores, cabeçalhos e informação suplementar.

3.3.1 Codificação preditiva – *Predictive coding*

A compressão dos vectores de movimento pode ser melhorada, se for feita uma predição de cada vector de movimento em relação a vectores de movimento codificados anteriormente. Assim, em vez de transmitir todos os vectores de movimento, só é transmitida a codificação da diferença entre o vector de movimento predito e o vector de movimento actual. Este processo denomina-se MVD, Diferença entre Vectores de Movimento, do inglês *Motion Vector Difference*.

Em aplicações em tempo real, há necessidade de alterar o parâmetro do quantificador, o degrau de quantificação, devido, por exemplo, à necessidade de ajustar o débito ao canal de transmissão. Normalmente o novo degrau de quantificação difere ligeiramente do valor anterior. Sendo assim, e à imagem do MVD, é mais eficaz enviar a diferença que houve no degrau de quantificação (por exemplo, ±1 ou ±2) em vez de enviar o valor completo do novo degrau de quantificação.

Utilizando estas técnicas, menos bits são transmitidos, ou seja, há compressão, uma vez que é necessário menos bits para representar as diferenças do que seria para representar por completo um novo valor.

3.3.2 Códigos de comprimento variável – VLC

Os codificadores de códigos de comprimento variável (VLC) codificam uma série de símbolos em palavras-chave (*keywords*). Símbolos que ocorrem mais frequentemente são representados por VLCs curtos (menos bits), enquanto que símbolos mais escassos são representados por VLCs longos. Numa sequência de símbolos suficientemente longa isto conduz a uma compressão de informação.

- Códigos de Huffman

Nos códigos de Huffman [12], a cada símbolo é atribuído um VLC de acordo com a probabilidade de ocorrência de símbolos diferentes. De acordo com o esquema original proposto por Huffman, é necessário calcular a probabilidade de ocorrência de cada símbolo e construir um conjunto de palavras-chave de códigos variáveis. Se as distribuições de probabilidades forem exactas, então obtém-se uma representação relativamente compacta dos símbolos originais.

Uma desvantagem dos códigos de Huffman, quando aplicados ao CoDec de vídeo, é a necessidade de o descodificador ter a mesma tabela de códigos do codificador. Assim, é necessário transmitir toda a tabela de probabilidades contida no codificador para o descodificador, o que vai acrescentar informação extra (*overhead* extra), reduzindo a eficácia da compressão, principalmente para vídeos curtos. Outro problema é que a tabela de probabilidades não pode ser calculada, sem que o vídeo seja totalmente codificado, o que pode introduzir um atraso inaceitável no processo de codificação.

Para resolver os problemas associados ao uso de códigos de Huffman, entidades relacionadas com as normas de codificação de vídeo definiram um conjunto de *codewords* baseadas em distribuições de probabilidade de conteúdos de vídeo genérico. Uma vez que as tabelas de VLC são previamente calculadas, basta armazenar essas tabelas, tanto no codificador como no descodificador.

Uma desvantagem de usar VLC é que estes são muito sensíveis a erros de transmissão. Basta que ocorra um erro na sequência dos VLCs para que o descodificador perder a sincronização e falhar a descodificação de códigos subsecutivos. Por esta razão, existem os códigos de comprimento variável reversíveis, RVLC, que podem ser descodificados em ambas as direcções o que aumenta a imunidade a erros.

- Códigos aritméticos

Com códigos VLC pré calculados é necessário armazenar a mesma tabela de VLCs tanto no codificador, como no descodificador, o que aumenta os requisitos de memória. Os códigos aritméticos geram automaticamente códigos para cada símbolo de entrada. Além disso, os códigos aritméticos atribuem o número óptimo de bits para representar cada símbolo baseado no seu conteúdo, ao contrário dos VLC, onde existe sempre um número integral de bits para cada tipo de símbolo. Um codificador aritmético converte uma sequência de símbolos num único número fraccionário e consegue aproximar-se do número óptimo de bits para representar cada símbolo [4].

Os códigos exponenciais de Golomb, Exp-Golomb, usados na norma MPEG-4 AVC/H.264 são códigos aritméticos.

A codificação aritmética baseada no contexto, CAE (*Context-based Arithmetic Encoding*) usa as características espaciais/temporais para estimar a probabilidade de cada símbolo para codificação. A CAE é usada nas mais recentes normas de codificação de vídeo como o MPEG-4 Visual e MPEG-4 AVC/H.264.

3.4- Codificador DPCM/DCT

Quase todos os principais codificadores de vídeo, desde 1990, se baseiam no modelo híbrido DPCM/DCT, Figura 13 e Figura 14 [4]. Incorporam um bloco de estimação e compensação de movimento (DPCM) e um estágio de transformada (DCT) e codificação de entropia.

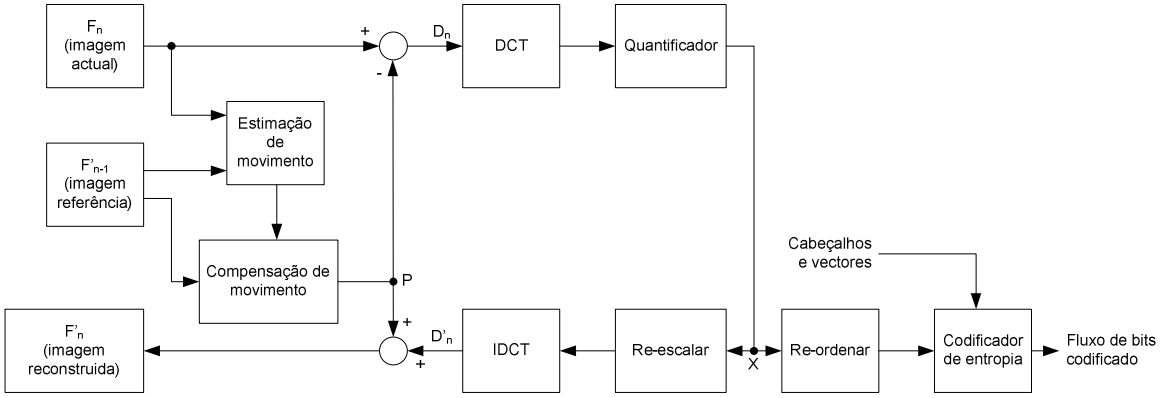

Figura 13 – Codificador DPCM/DCT.

A Figura 13 mostra o diagrama de blocos de um codificador DPCM/DCT. Por sua vez, a Figura 14 mostra um descodificador DPCM/DCT que é muito semelhante ao codificador. Observa-se que o codificador também inclui na sua malha (*loop*) um descodificador, tal como mostra a figura. Isto deve-se ao facto de o codificador necessitar de ter como referência para codificação as mesmas imagens que o descodificador vai usar.

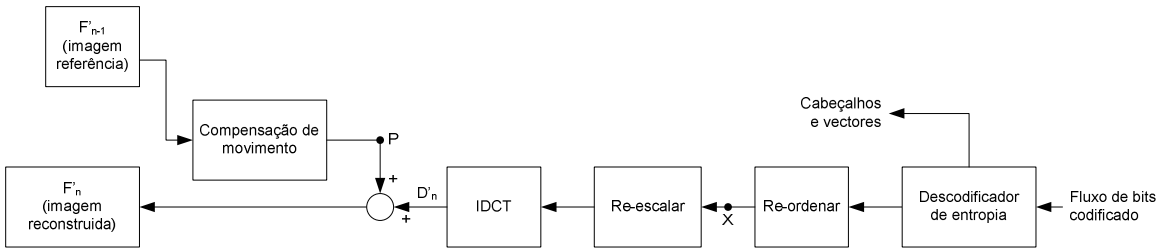

Figura 14 – Descodificador DPCM/DCT

O funcionamento do codificador é o seguinte [4]:

1. A imagem de vídeo (*frame*) F_n, Figura 15, que vai ser codificada é dividida em unidades de macroblocos. Estes macroblocos, na norma MPEG-4 Visual, têm um tamanho de 16 × 16 pixel para a luminância e 8 × 8 pixel para cada uma das componentes de crominância.

Figura 15 – Imagem actual F_n.

2. F_n é comparada com uma imagem de referência que pode ser uma imagem anteriormente codificada (F'_{n-1}), Figura 16. O estimador de movimento procura uma região 16 × 16 pixel em F'_{n-1}, ou numa interpolação de F'_{n-1}, que melhor coincide com o actual macrobloco de F_n dentro de um determinado critério. O desvio de posição entre o macrobloco actual e a região de referência encontrada é o vector de movimento, MV.

Figura 16 – Imagem de referência F'_{n-1}.

3. Com base no MV, uma predição P é gerada. P é a região 16 × 16 pixel seleccionada pelo estimador de movimento.
4. P é subtraído ao macrobloco actual para se obter o macrobloco residual D, Figura 17.

Figura 17 – Imagem residual F_n-F'_{n-1}.

5. D é transformado através da transformada DCT. Tipicamente D é separado em sub-blocos 8 × 8 ou 4 × 4 pixel, onde cada sub-bloco é transformado separadamente.
6. Cada sub-bloco é quantificado, o que origina X.
7. Cada coeficiente da DCT é reordenado e codificado pelo método *Run-Level*.
8. Os coeficientes, vector de movimento, Figura 18, e cabeçalhos associados são codificados pelo codificador de entropia para produzir o fluxo de bits, *bitstream*, comprimido.

Figura 18 – Vectores de movimento.

9. D é transformado através da transformada DCT. Tipicamente D é separado em sub-blocos 8 × 8 ou 4 × 4, onde cada sub-bloco é transformado separadamente.
10. Cada sub-bloco é quantificado, o que origina X.
11. Cada coeficiente da DCT é reordenado e codificado pelo método *Run-Level*.
12. Os coeficientes, vector de movimento, e cabeçalhos associados são codificados pelo codificador de entropia para produzir o fluxo de bits, *bitstream*, comprimido.

A reconstrução funciona da seguinte forma:

13. Cada macrobloco X é passado pelo quantificador inverso, *rescaler*, e pela transformada inversa IDCT. Uma vez que o quantificador elimina informação, o macrobloco X ao passar pelo *rescaler* não vai possuir os mesmos valores que tinha antes da quantificação. Por isso, em vez de se obter D obtém-se uma cópia aproximada, D', de D.
14. A predição da compensação de movimento P é adicionada a macrobloco residual D' para produzir o macrobloco reconstruído. Os macros blocos reconstruídos são guardados para produzir a imagem reconstruída F'_n. Esta imagem reconstruída pode ser usada como imagem de referência para a próxima imagem F_{n+1}.

Descodificador:

15. O fluxo de bits que chega é descodificado pelo descodificador de entropia. São extraídos os vectores de movimento, coeficientes e cabeçalhos para cada macrobloco.
16. Reordenação e descodificação *Run-Level* são efectuadas para se obter X.
17. X é inversamente quantificado e é-lhe aplicada a transformada inversa donde se obtém o residual D'. Não se obtém D pelos mesmos motivos anteriores.
18. O vector de movimento descodificado é usado para localizar a região 16 × 16 na cópia da imagem anterior F'_{n-1} residente no descodificador. Esta região torna-se a predição da compensação de movimento P.
19. P é adicionado a D' e o macrobloco é reconstruído. Os vários macroblocos são guardados para reconstruir a imagem $F'_{n.}$

É necessário que o codificador e o descodificador tenham a mesma imagem de referência F'_{n-1} para a predição da compensação de movimento. Por isso, no codificador existe também um caminho de descodificação como no descodificador.

Após se terem estudado os métodos genéricos da codificação de vídeo, e em particular o codificador DPCM/DCT, a compreensão da codificação de vídeo MPEG-4 AVC/H.264 torna-se mais simples. Como se observará no próximo capítulo, a norma MPEG-4 AVC/H.264 não é mais do que uma melhoria, apesar de ser muito significativa, do codificador DPCM/DCT.

Capítulo 4. NORMA MPEG-4 AVC/H.264

Neste capítulo, descreve-se a norma MPEG-4 AVC/H.264 [2]. No entanto, a camada do codificador de vídeo, VCL, MPEG-4 AVC/H.264 será o foco de estudo deste capítulo.

A norma MPEG-4 AVC/H.264 é uma norma de codificação de vídeo desenvolvido pelos grupos ISO/IEC *Moving Picture Expert Group* (MPEG, MPEG-4 *part* 10) e ITU-T *Video Coding Experts Group* (VCEG) [13]. Esta norma tem como principal objectivo aumentar o desempenho da compressão e prover facilidades de rede a aplicações conversação (vídeo telefonia) e aplicações de *streaming*, armazenamento ou difusão.

O processo de normalização, Figura 19, tem como objectivo definir, normalizar, só o descodificador, impondo restrições na sintaxe e no fluxo de bits (*bitstream)*, e definindo os elementos da sintaxe do processo de descodificação, para que todos os descodificadores que esteja em conformidade com a norma produzam uma saída similar para o mesmo *bitstream* codificado, que esteja de acordo com as restrições da norma. Uma vez que a norma foca só o processo de descodificação, isto permite um maior grau de liberdade para a implementação do codificador. No entanto, não há garantias de qualidade de reprodução, uma vez que são permitidas codificações mais rudimentares [13].

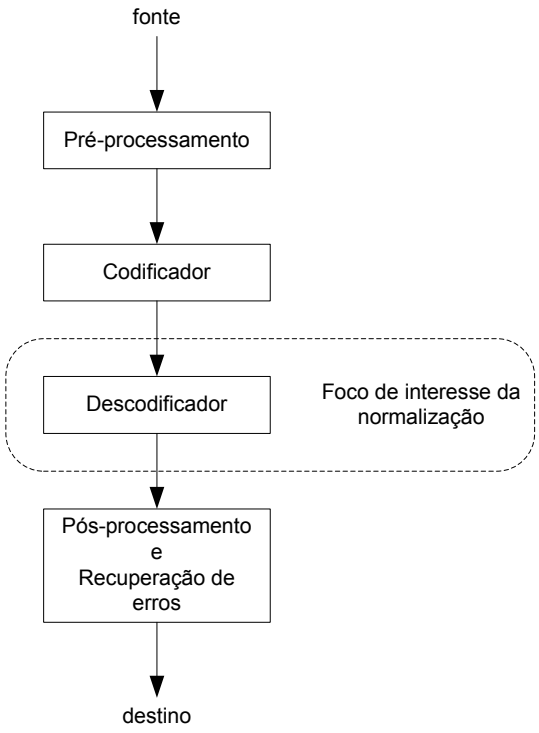

Figura 19 – Foco de interesse das normas de codificação de vídeo.

Esta norma está desenhada para solucionar as seguintes áreas de aplicação:

- Difusão sobre cabo, satélite, moduladores/desmoduladores (*modems*) para cabo, DSL, terrestre, etc.

- Aplicações interactivas ou de armazenamento em suportes ópticos ou magnéticos

- Serviços de voz sobre ISDN, Ethernet, LAN, DSL, rádio (wireless) ou redes móveis, entre outros.

- *Video-on-demand* ou serviços de *streaming* multimédia sobre DSL, cabo, ISDN, redes sem fios (*wireless*) ou redes móveis, etc.

- Serviço de mensagens multimédia (MMS) sobre ISDN, DSL, ethernet, LAN, *wireless* ou redes móveis, etc.

De forma a acomodar todos estes tipos de serviços, a estrutura do codificador de vídeo MPEG-4 AVC/H.264, Figura 20, é composta por: uma camada de codificação de vídeo VCL (*Video Coding Layer*) que representa eficientemente o conteúdo vídeo e uma outra camada de abstracção da rede NAL (*Network Abstraction Layer*) que formata o VCL e proporciona um cabeçalho de informação de forma apropriada a uma variedade de camadas de transporte e armazenamento.

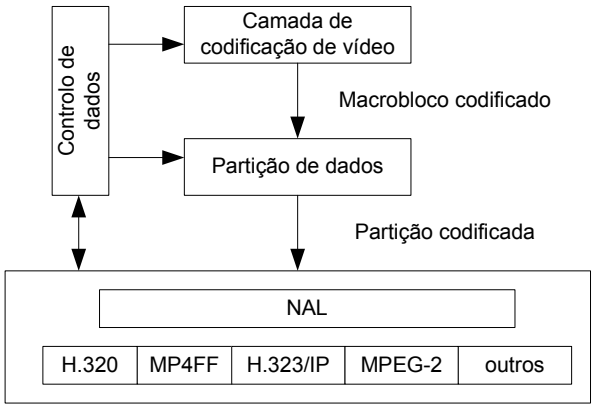

Figura 20 – Estrutura do codificador de vídeo MPEG-4 AVC/H.264.

As melhorias que o MPEG-4 AVC/H.264 apresenta para aumentar a eficiência de codificação em relação a codificadores anteriores como o MPEG-2 são [13]:

- Compensação de movimento com tamanho de blocos variáveis e de tamanho reduzido

- Compensação de movimento com precisão *quarter-sample*

- Vectores de movimento para além das fronteiras da imagem

- Compensação de movimento baseado em múltiplas imagens de referência

- Independência da ordem das imagens de referência em relação à ordem das imagens de apresentação

- Independência dos métodos da representação da imagem em relação à imagem de referência

- Predição ponderada

- Melhoramentos na inferência de movimento directo (*direct*) e indirecto (*skipped*)

- Predição espacial direccional para codificação Intra

- Filtro de redução do efeito de bloco (*deblocking filter*) incluído na estrutura do codificador

- Transformada efectuada sobre blocos de tamanho reduzido

- Transformada efectuada hierarquicamente sobre blocos

- Transformada de menor precisão

- Transformada inversa exacta

- Codificação de entropia aritmética

-Codificação de entropia adaptável ao contexto (CAE)

-Estrutura de parâmetros

-NAL

- *Slices* de tamanho flexível

- Ordem de macroblocos flexível (FMO)

- Ordem de *slices* arbitrária (ASO)

- Imagens redundantes

- Partição de informação

- Imagens SP/SI de sincronização/*switching*

4.1- Camada do codificador de vídeo – VCL

O diagrama de blocos do codificador MPEG-4 AVC/H.264 [2], mostrado na Figura 21, é similar ao diagrama de blocos DPCM/DCT apresentado anteriormente na Figura 13. No entanto, são os detalhes que o diferenciam que permitem melhorias significativas de ganho.

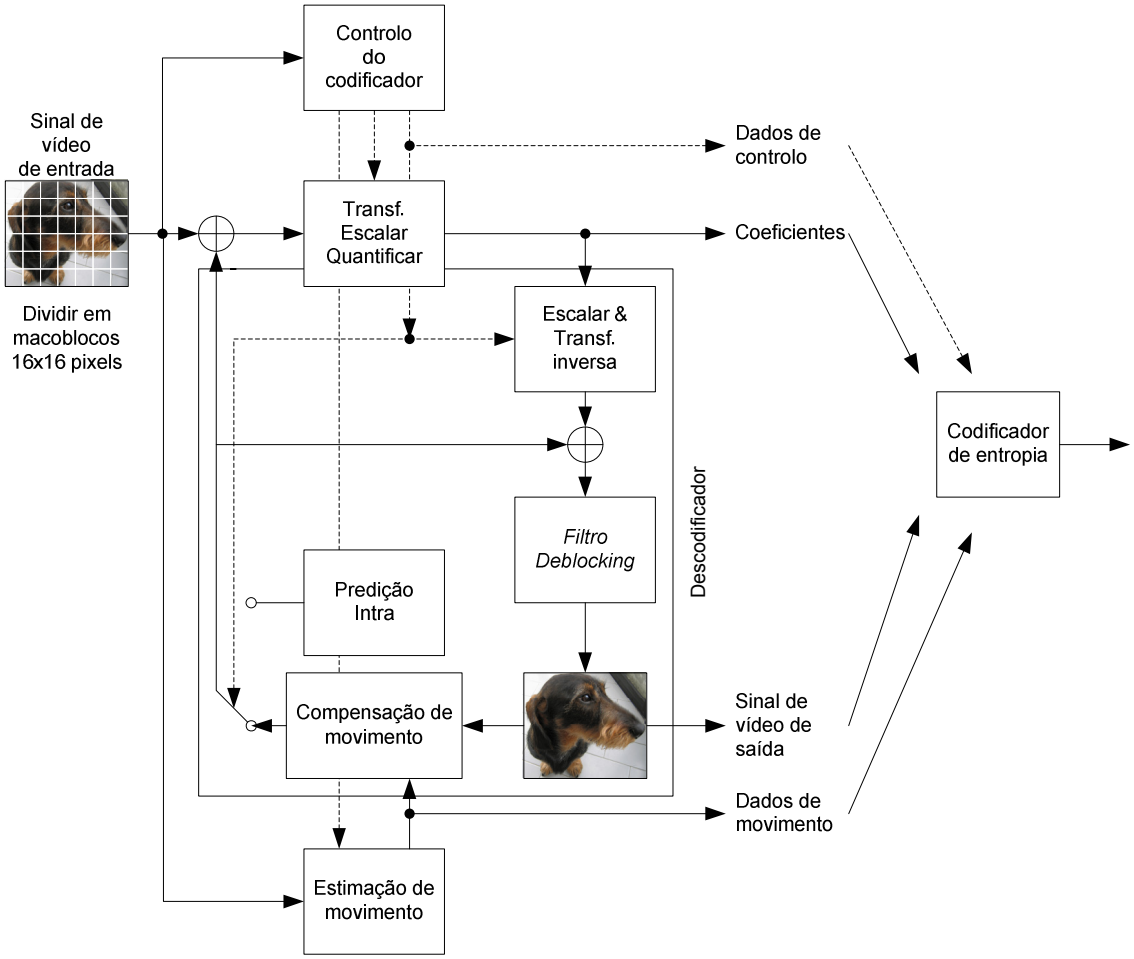

Figura 21 – Diagrama de blocos do codificador MPEG-4 AVC/H.264.

A norma MPEG-4 AVC/H.264 define três Perfis (*Profiles)*, cada um suportando um conjunto particular de funções e especificações de codificação, para que um codificador e um descodificador estejam de acordo com o Perfil, Figura 22.

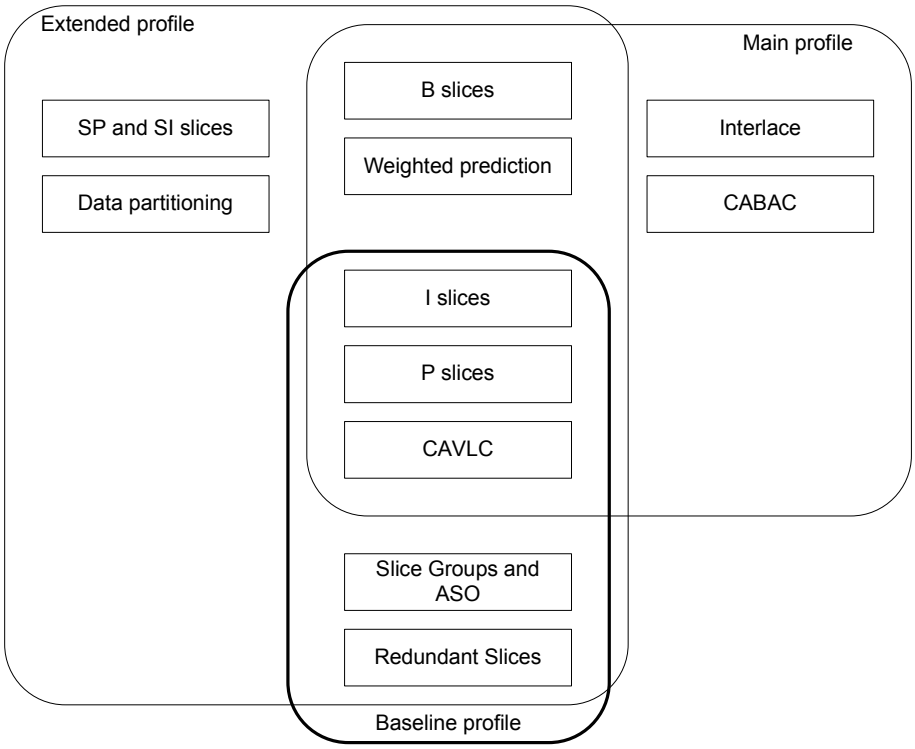

Figura 22 – Perfis do MPEG-4 AVC/H.264.

O Perfil Base (*Baseline Profile*) suporta codificação Intra e Inter, usando I-slices e P-slices, e o codificador de entropia é baseado no CAVLC, *Context-Adaptive Variable Length Codes*. O Perfil Principal (*Main Profile*) inclui o suporte para vídeo entrelaçado, codificação Inter usando B-slices, codificação Inter usando predição ponderada e codificação de entropia CABAC, *Context-Adaptive Based Arithmetic Coding*. Por fim, o outro Perfil Estendido é o (*Extendend Profile*) que não suporta vídeo entrelaçado nem CABAC, mas adiciona modos que tornam eficiente a troca (*switching*) entre *bitstreams* codificadas (SP e SI-slices) e melhorada imunidade a erros (partição de dados/informação, *data partitioning*).

Cada um destes perfis está principalmente desenvolvido para determinada aplicação [4]. O perfil base, *Base Profile*, está orientado para aplicações de videotelefone, videoconferência e comunicações sem fios (*wireless*). O perfil principal, *Main Profile*, vai de encontro a aplicações de difusão de televisão e armazenamento de vídeo, enquanto que o perfil estendido, *Extended Profile*, serve melhor aplicações de *streaming* de informação. De qualquer forma, cada um destes perfis pode ser usado em outras aplicações devido à sua flexibilidade.

4.1.1 Imagens e campos

Na norma MPEG-4 AVC/H.264 uma sequência de vídeo codificada é composta por uma sequência de imagens codificadas. O formato do vídeo pode ser progressivo, uma sequência de imagens, ou entrelaçado, uma sequência de campos, tal como na norma MPEG-2. Contudo, uma vez que no MPEG-4 AVC/H.264 a codificação é primeiramente baseada em conceitos geométricos do que em temporização, implica que, esta norma seja primordialmente agnóstica em relação ao tipo de vídeo [13].

4.1.2 Espaço de cor YC_bC_r e amostragem 4:2:0

Na norma MPEG-4 AVC/H.264 a representação da cor do vídeo é separada em três componentes: Y (luminância), Cb (crominância azul) e Cr (crominância vermelha). Como o sistema visual humano é mais sensível ao brilho (*luma*) do que à cor (*chroma*), a resolução das componentes de crominância corresponde a um quarto da componente de luminância, ou seja, é usada a amostragem de vídeo do tipo 4:2:0 com 8 bits de precisão para cada amostra. A norma MPEG-4 AVC/H.264 também prevê suporte para mais resolução nas componentes de cor e mais bits por amostra.

4.1.3 Divisão da imagem em macroblocos

A divisão da imagem é feita em macroblocos (MBs) de 16 × 16 amostras de luminância e 8 ×8 amostras para cada componente de crominância. Os macroblocos são blocos básicos da norma para os quais o processo de codificação está especificado.

4.1.4 Partições e grupos de partições – *Slices* e *Group Slices*

A imagem é dividida em áreas, fatias, que se denominam por *slices*. As *slices* são uma sequência de macroblocos que são processados ordenadamente pela ordem de varrimento *raster* ou pela ordem FMO (*Flexible Macroblock Order*) onde os macroblocos podem ser codificados fora da ordem *raster*. As *slices* são auto contidas, i.e., uma *slice* pode ser correctamente descodificada, sem fazer uso da informação de outra *slice* [4].

A ordem das *slices* pode ser arbitrária, *Arbitrary Slice Order* (ASO), o que significa que uma *slice* de uma *frame* codificada pode ser descodificada por qualquer ordem.

Entende-se que se está a usar ASO, se o primeiro MB de qualquer *slice* de um *frame* descodificado possuir um endereço de MBs menor do que o primeiro MB de uma *slice*, anteriormente descodificada da mesma imagem.

As *slices* podem ser agrupadas em *slice groups*. Os *slice groups* são um subconjunto de macroblocos da imagem codificada que contém uma ou mais *slices*. Os macroblocos de cada *slice* de um *slice group* são codificados pela ordem *raster*. Se só existir um *slice group* por imagem, então todos os macroblocos da imagem são codificados pela ordem *raster*, a menos que se esteja a usar ASO. Ao usar FMO, ou múltiplos *slice groups*, é possível passar a sequência de MBs codificados para a imagem descodificada de várias formas. A distribuição de MBs é determinada pelo mapa de MBs do *slice group*, o qual indica a que *slice group*, o MB pertence. A Tabela 3, mostra os diferentes tipos de mapas de *slice groups* para um MB, Figura 23.

Tabela 3 – Tipos de group slices.

Tipo	Nome	Descrição
0	*Interleaved*	Os MBs são atribuídos a cada *slice* à vez, Figura 23-a).
1	*Dispersed*	Os MBs em cada *slice group* são dispersados ao longo da imagem, Figura 23-b).
2	*Foreground and background*	Todos excepto o último *slice group* estão definidos como regiões rectangulares. O último *slice group* contém todos os MBs não contidos em qualquer outro *slice group*, Figura 23-c).
3	*Box-out*	Uma "caixa" é criada a partir do centro do *frame* e contém o grupo 0. Todos os outros MBs pertencem ao grupo 1.
4	*Raster scan*	Grupo 0 contém MBs pela ordem *raster* varridos desde o topo esquerdo e todos os outros MBs estão no grupo 1
5	*Wipe*	Grupo 0 contém MBs varridos na vertical desde o topo esquerdo e todos os outros MBs estão no grupo 1.
6	*Explicit*	Um parâmetro, *slice_group_id*, é enviado para cada MB indicando o grupo da sua *slice*.

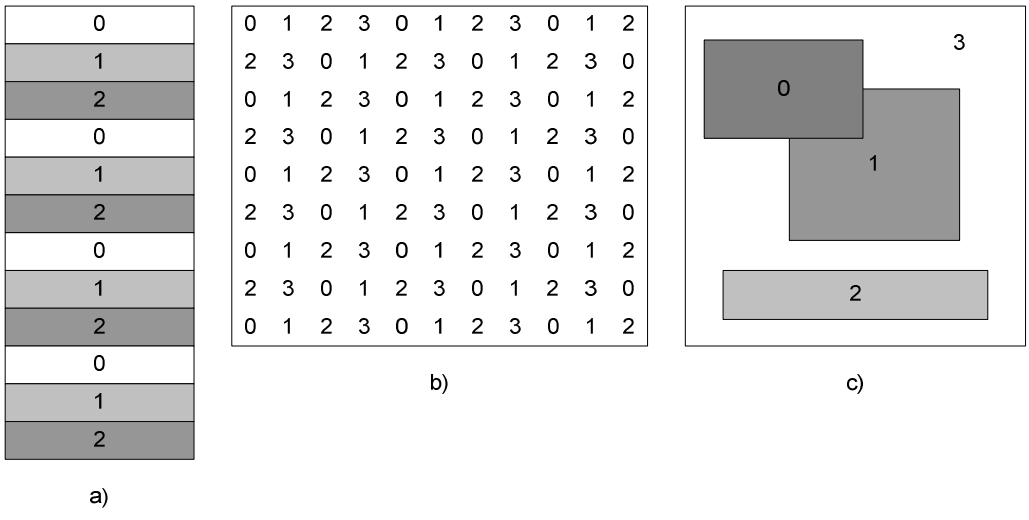

Figura 23 – Exemplos de group slices: a) interleaved, b) dispersed e c) foreground and background.

Independentemente de se estar a usar FMO ou não, cada *slice* pode ser codificada em vários tipos de codificação:

- I *slice* (Intra): todos os MBs da *slice* são codificados, usando o modo Intra

- P *slice* (*predicted*): em adição às I *slices*, alguns MBs das P *slices* podem ser codificados, usando o predição Inter, com pelo menos um sinal com compensação de movimento com predição por bloco.

- B *slice* (*bi-predictive*): além de se usar P *slices*, alguns MBs podem ser codificados, usando predição Inter com dois sinais com compensação de movimento com predição por bloco.

- SP (*Switching* P) *slice*: facilita a troca (*switching*) entre *streams* codificadas. Contém P e I macroblocos.

- SI (*Switching I*) *slice*: permite encontrar o MB exacto que coincide com a SP *slice* para recuperar de erros e permitir acesso aleatório.

4.1.5 Codificação adaptativa de Imagem/Campo (*Frame/Field*)

Quando se usa vídeo entrelaçado, duas linhas adjacentes têm uma dependência estatística reduzida, quando comparadas com as de vídeo progressivo. É mais eficiente codificar separadamente cada campo do vídeo entrelaçado. A norma MPEG-4 AVC/H.264 permite que o codificador decida qual decisão deve tomar para codificar o campo (*field*):

- Combinando o campo par e impar juntos e codificando-os como uma única imagem de um vídeo progressivo (modo imagem).

- Codificando separadamente os campos par e impar (modo campo).

- Combinando o campo par e impar juntos e comprimindo-os numa única imagem, mas aquando da codificação da imagem, para separar os pares de dois macroblocos verticalmente adjacentes em pares de macroblocos de dois campos ou uma imagem antes de codificá-los.

A opção entre os três modos pode se adaptativa para cada campo da sequência. Chama-se codificação *Picture-Adaptive Frame/Field* (PAFF), quando se escolhe entre as duas primeiras possibilidades.

Quando uma imagem é codificada em dois campos, então cada um deles é dividido em MBs e é codificado como se fosse uma imagem, excepto se:

- A compensação de movimento utilizar como referência campos, em vez de imagens.

- O varrimento em ziguezague dos coeficientes da transformada for diferente.

- Não for feita a filtragem horizontal de MBs dos campos.

Caso um quadro contenha uma mistura de regiões, onde existe movimento numas e noutras não, então, é tipicamente mais eficiente codificar as regiões estáticas no modo imagem, e as outras no modo quadro. Durante o desenvolvimento da norma MPEG-4 AVC/H.264 a codificação PAFF mostrou que reduzia o débito em 16% a 20% em relação à codificação só no modo quadro.

À técnica de seleccionar vários modos de codificação, campo ou imagem, na mesma cena, chama-se codificação *Macroblock-Adaptive Frame/Field* (MBAFF), Figura 24.

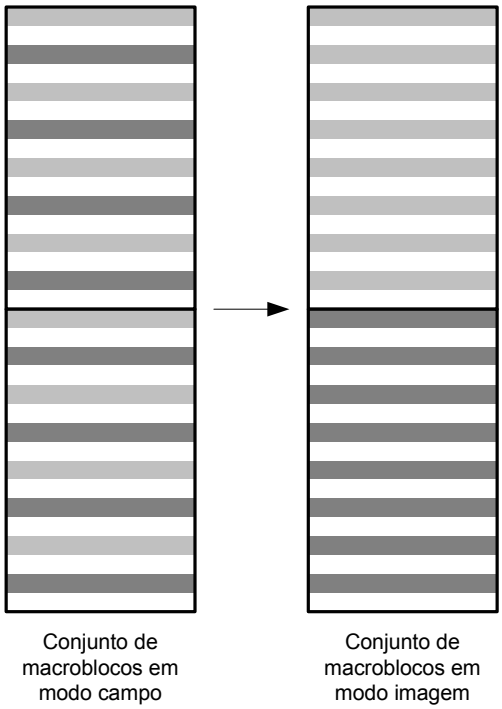

Figura 24 – Conversão campo/imagem.

4.1.6 Predição Intra – *Intra frame prediction*

- Intra_4×4

O modo Intra_4×4 baseia-se na predição de cada bloco de luminância 4 × 4, e está bem adaptado à codificação de partes da imagem com detalhe. Cada bloco 4 × 4 é predito das amostras vizinhas espaciais, Figura 25. As 16 amostras do bloco 4 × 4, *a* a *p*, foram preditas usando amostras previamente descodificadas em blocos adjacentes, A a Q.

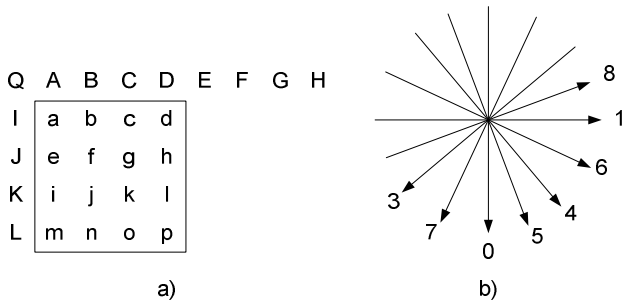

Figura 25 – a) Intra_4×4 e b) as oitos direcções.

No modo Intra_4×4 existem nove modos de predição, Figura 26.

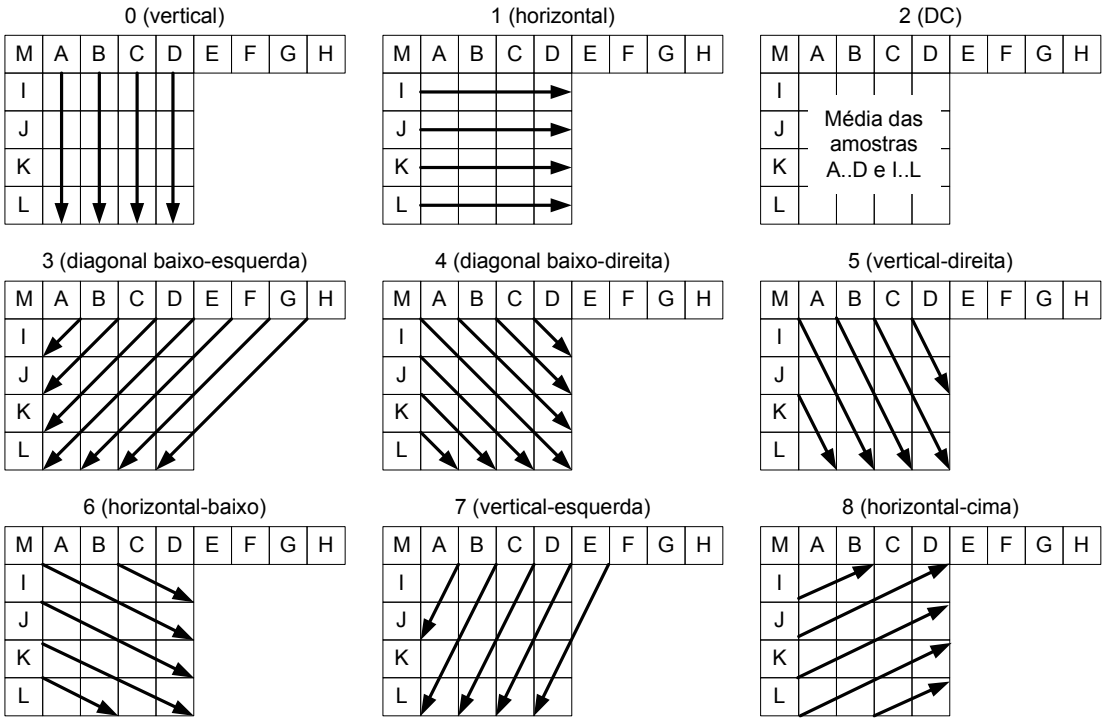

Figura 26 – Modos de predição Intra_4x4.

Estes modos estão adaptados para a predição direccional de estruturas na imagem tal como cantos de vários ângulos.

- Intra_16×16

Este modo faz a predição de blocos de luminância de 16 × 16 e está adaptado à codificação de áreas suaves da imagem. Existe também o equivalente para a crominância.

Neste modo existem só quatro modos de predição, Figura 27.

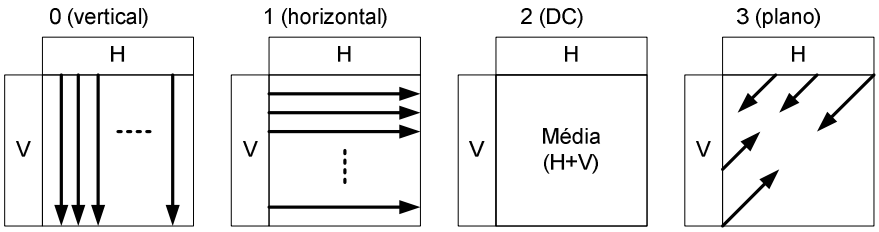

Figura 27 – Modos de predição Intra_16×16.

O modo Intra_16×16 também é utilizado para a predição Intra da crominância. Usa os mesmos quatro modos de predição, mas usa blocos de 8 × 8 em vez de 16 × 16, uma vez que a crominância normalmente é "lisa" (contínua) ao longo de grandes áreas.

- I_PCM

Além dos dois tipos de predição, Intra_4×4 e Intra_16×16, existe um tipo de codificação I_PCM. Este tipo de codificação permite contornar os processos de predição e transformação, enviando directamente os valores das amostras codificadas. O I_PCM serve para:

- Permitir que o codificador represente precisamente os valores das amostras

- Prover uma forma de representar os valores dos conteúdos anómalos da imagem, sem aumentar em muito os dados.

- Estabelecer um limite físico no número de bits que um descodificador deve processar para um MB sem degradar a eficiência de codificação.

Ao contrário de outras normas de codificação de vídeo que fazem a predição Intra no domínio da transformada, no MPEG-4 AVC/H.264, a predição Intra é feita sempre no domínio espacial, referindo as amostras vizinhas de blocos previamente codificados que estão à esquerda e/ou acima do bloco predito. Para evitar a propagação de erros, é inserido um modo Intra e assinalado, para que a predição seja feita só de MBs vizinhos codificados em Intra.

Os modos de predição Intra não passam as fronteiras das *slices*, de forma a manter todas as *slices* independentes de cada uma.

4.1.7 Predição Inter – *Inter predicton*

- Compensação de movimento em árvore.

A luminância de cada MB pode ser dividida em quatro formas, como se pode constatar na Figura 28:

- Uma partição do MB 16 × 16

- Duas partições de 16 × 8

- Duas partições de 8 × 16

- Quatro partições de 8 × 8

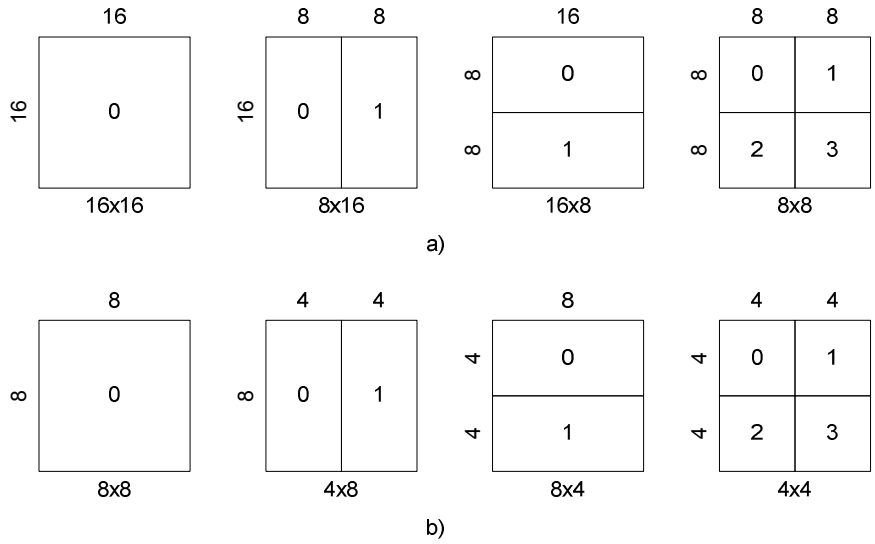

Figura 28 – partições de a) macroblocos e b) sub-macroblocos.

Se for escolhido o modo 8 × 8 então cada um dos quatro sub-macroblocos do macrobloco original pode ser dividido em duas partições de 8 × 4, ou quatro partições de 4 × 4. Estas partições aumentam o número de combinações possíveis em cada MB. A esta forma, de dividir MBs em sub-blocos, dá-se o nome de compensação de movimento em árvore (*tree structured motion compensation*).

É necessário que o codificador envie a informação relativa ao modo como está a dividir o MB.

Em cada sub-macrobloco com codificação Inter, a sua predição é feita a partir de uma área igual na imagem de referência. O desvio entre as duas áreas, vector de movimento, tem uma resolução de *quarter-sample* para luminância e *eight-sample* para crominância. Como essas amostras não existem, é necessário proceder à interpolação com base em amostras vizinhas, Figura 29.

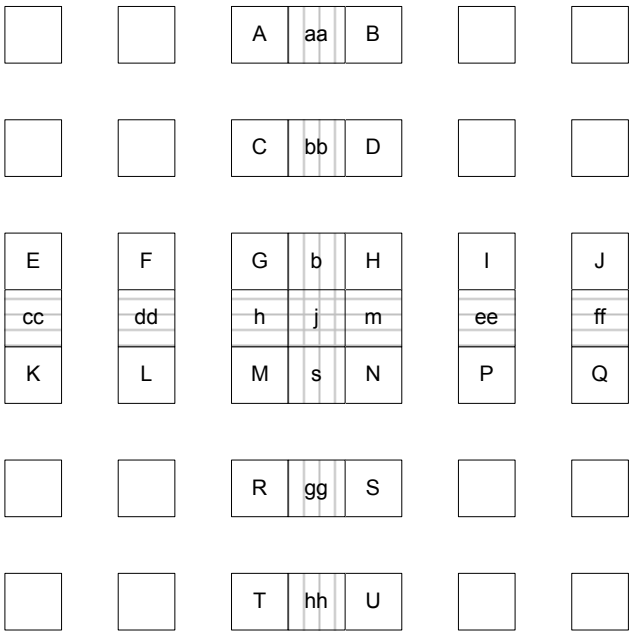

Figura 29 – Interpolação para luminância *half-sample*.

As amostras com resolução *half-sample* são obtidas aplicando um filtro de resposta impulsional finita (FIR) com 6 *taps* horizontalmente e verticalmente:

$$b = round\left(\frac{E - 5F + 20G + 20H - 5I + J}{32}\right) \quad \textbf{(4.1.1)}$$

$$h = round\left(\frac{A - 5C + 20G + 20M - 5R + T}{32}\right) \quad \textbf{(4.1.2)}$$

As amostras *quarter-sample* são obtidas por interpolação linear, Figura 30.

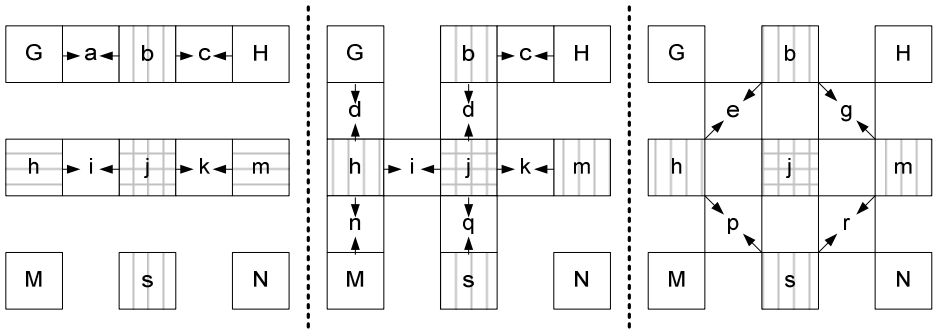

Figura 30 – Interpolação para luminância *quarter-sample*.

4.1.8 Transformada

Na norma MPEG-4 AVC/H.264 a transformada á aplicada a blocos de 4 × 4, e em vez de se usar uma DCT, é usado uma transforma inteira. São usadas três transformadas que dependem do tipo de informação residual, que vai ser codificada: transformada de Hadamard 4 × 4 para a luminância de MBs com predição Intra no modo 16 × 16, transformada de Hadamard 2 × 2 para crominância, e uma transformada baseada na DCT, para todos os outros blocos 4 × 4 residuais.

- Transformada residual 4 × 4 e quantificação

Esta transformada 4 × 4 opera sobre os dados dos blocos residuais, após a compensação de movimento com predição (predição Intra) [4]. Esta transformada é baseada na DCT, mas com as seguintes diferenças:

- É uma transformada baseada em números inteiros

- É possível obter uma transformada inversa exacta

- Bastam adições e deslocamentos lógicos binários (*shifts*) para a implementação

- A parte de escalonamento está integrada no quantificador

$$Y = AXA^T \tag{4.1.3}$$

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad sendo \quad \begin{matrix} a = 1/2 \\ b = \sqrt{1/2}\cos\frac{\pi}{8} \\ c = \sqrt{1/2}\cos\frac{3\pi}{8} \end{matrix} \tag{4.1.4}$$

Factorizando a equação $Y=AXA^T$ obtém-se:

$$Y = (CXC^T) \otimes E \tag{4.1.5}$$

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} X \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

$$onde \qquad d = c/b \qquad \textbf{(4.1.6)}$$

CXC^T representa o núcleo (*core*) da transformada 2D. Onde E é matriz de escalamento dos factores.

Para simplificar a implementação da transformada, fazem-se as seguintes aproximações:

$$a = 1/2 \quad b = \sqrt{2/5} \quad d = 1/2 \qquad \textbf{(4.1.7)}$$

Desta forma obtém-se o seguinte resultado:

$$Y = (C_f X C_f^T) \otimes E_f \qquad \textbf{(4.1.8)}$$

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \qquad \textbf{(4.1.9)}$$

Esta transformada aproxima-se de uma DCT 4 × 4, mas devido à alteração dos factores *d* e *b*, o resultado é diferente de uma verdadeira DCT 4 × 4.

O núcleo da transformada, $C_f X C_f^T$, pode ser calculado com aritmética de inteiros, usando só adições, subtracções e *shifts*, além de bastar um acumulador de 16 bits para efectuar todas as operações, uma vez que as amostras são todas de 8 bits. A outra parte da

transformada, $\otimes E_f$, requerer só uma multiplicação por cada coeficiente que pode ser embutida no processo de quantificação

A transformada inversa é dada pela equação [4]:

$$Y = C_i^T (Y \otimes E_i) C_i \tag{4.1.10}$$

$$Y = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \cdot \left(Y \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -2 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix} \tag{4.1.11}$$

$T^{-1}(T(X)) = X$, são ortogonais as tranformadas inversa e directa

A operação básica de quantificação é:

$$Z_{ij} = round(\frac{Y_{ij}}{Q_{step}}) \tag{4.1.12}$$

Sendo W= CXC^T e de modo a facilitar as operações de aritmética:

$$Z_{ij} = round(W_{ij} \frac{PF}{Q_{step}}) \tag{4.1.13}$$

Onde PF é a^2, $ab/2$ ou $b^2/4$ dependendo da posição (i, j).

De forma a simplificar as operações aritméticas, o factor PF/Q_{step} é implementado da seguinte forma:

$$\frac{MF}{2^{qbits}} = \frac{PF}{Q_{step}} \tag{4.1.14}$$

onde,

$$qbits = 15 + floor(QP/6) \tag{4.1.15}$$

e deste modo obtém-se:

$$\begin{aligned} |Z_{ij}| &= \left(|W_{ij}|MF + f\right) >> (qbits + 1) \\ sinal(Z_{ij}) &= sinal(W_{ij}) \end{aligned} \tag{4.1.16}$$

A operação >> indica um *shift* binário para a direita. No modelo de referência:

$$f = 2^{qbits}/3 \text{, codificação Intra} \tag{4.1.17}$$

$$f = 2^{qbits}/6 \text{, codificação Inter} \tag{4.1.18}$$

O processo de quantificação inversa é feito da seguinte forma:

$$Y'_{ij} = Z_{ij} Q_{step} \tag{4.1.19}$$

$$W'_{ij} = Z_{ij} Q_{step} \cdot PF \cdot 64 \tag{4.1.20}$$

Onde W'_{ij} é o coeficiente escalado que é transformada inversamente por $C_i^T W C_i$.

Uma vez que a norma MPEG-4 AVC/H.264 não especifica Q_{step}, nem PF directamente, é usado o parâmetro $V=Q_{step} \cdot PF \cdot 64$, e está definido entre $0 \leq QP \leq 5$ para cada posição do coeficiente, de tal forma que:

$$W'_{ij} = Z_{ij} V_{ij} 2^{floor\left(QP/6\right)} \tag{4.1.21}$$

- Transformada de coeficientes DC 4 × 4 para luminância

Se um MB é codificado no modo Intra_16×16 cada bloco residual 4 × 4 é transformado usando o "core" da transformada anterior, $C_f X C_f^T$. Cada coeficiente DC é transformado outra vez usando a transformada de Hadamard 4 × 4 [4].

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} W_D \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2 \tag{4.1.22}$$

Onde W_D é o bloco 4 × 4 dos coeficientes DC.

A transformada inversa é aplicada após o *rescaling*:

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} Z_D \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \quad \textbf{(4.1.23)}$$

- Transformada de coeficientes DC 2×2 para crominância [4]

  A transformada directa é dada pela equação:

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} W_D \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \textbf{(4.1.24)}$$

  E a transformada inversa:

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} Z_D \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \textbf{(4.1.25)}$$

4.1.9 Entropia

O MPEG-4 AVC/H.264 usa dois métodos de codificação de entropia CAVLC e CABAC [2].

- CAVLC

  Os códigos de comprimento variáveis usados no CAVLC são códigos Exp-Golomb. Estes possuem propriedades de descodificação simples e regular.

- Os códigos Exp-Golomb têm a forma de [M zeros][1][INFO], onde INFO é um campo de M bits que contém a informação. O tamanho de cada código Exp-Golomb é 2M+1 bits.

Os códigos são gerados, usando as seguintes expressões:

$$M = floor(\log_2(numero\_do\_codigo + 1)) \quad \textbf{(4.1.26)}$$

$$INFO = numero\_do\_codigo + 1 - 2^M \quad \textbf{(4.1.27)}$$

A tabela seguinte mostra os 9 primeiros códigos Exp-Golomb:

Tabela 4 – Códigos Exponenciais de Golomb.

numero_de_codigo	Código
0	1
1	010
2	00101
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001

A descodificação destes códigos é relativamente simples e obedece às seguintes regras, tendo em atenção que para *numero_de_codigo* = 0 → INFO = M = 0:

1. Ler o número de zeros em M seguido do valor 1.
2. Ler o campo INFO com M bits.
3. *número de_codigo* = 2^M + INFO – 1.

- CABAC

A eficiência do codificador de entropia pode ser melhorada usando o CABAC. Este tipo de codificação é muito eficiente para símbolos com probabilidades superiores a 50%. O uso de códigos adaptativos permite a adaptação a estatísticas não estacionárias dos símbolos.

Comparativamente ao CAVLC, o CABAC geralmente consegue uma redução no débito de transmissão de 5% a 15%, sendo os maiores ganhos quando se codifica sinais de TV entrelaçados.

4.1.10 Filtro de remoção de artefactos – *Deblocking filter*

Um dos problemas associados aos codificadores baseados em codificação de blocos é a produção de estruturas visíveis de blocos. Isto deve-se ao facto de as bordas dos blocos serem reconstruídas com menor precisão, que o interior do bloco. O facto de se verem estes blocos, é considerado um dos maiores artefactos visíveis, introduzidos por este tipo de codificação [13].

O MPEG-4 AVC/H.264 introduz um filtro que reduz este efeito nefasto. E o princípio deste filtro é o seguinte:

- Dadas as amostras p_0 e q_0 e as amostras p_1 e q_1, Figura 31.

- O parâmetro de quantificação QP, que depende dos limiares $\alpha(QP)$ e $\beta(QP)$, determina se estas amostras são filtradas

- A filtragem de p_0 e q_0 só acontece se:

1. $|p_0 - q_0| < \alpha(QP)$
2. $|p_1 - p_0| < \beta\ (QP)$
3. $|q_1 - p_0| < \beta\ (QP)$

Sendo $\beta(QP)$ consideravelmente menor que $\alpha(QP)$.

Desta forma, p_1 e q_1 só são filtrados se a condição for satisfeita:

$$|p_2 - p_0| < \beta\ (QP) \text{ ou } |q_2 q_0| < \beta\ (QP)$$

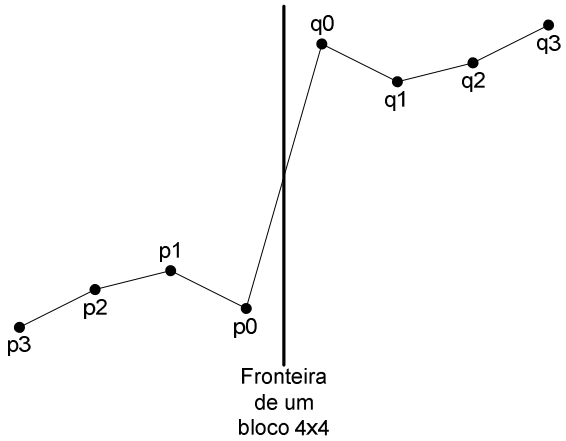

Figura 31 – Princípio de funcionamento do filtro de remoção de artefactos.

Com este filtro, os artefactos de blocos são reduzido e as altas frequências da imagem (*sharpness*) são mantidas. A Figura 32 mostra o efeito deste filtro.

Figura 32 – a) com filtro remoção de artefactos e b) sem filtro.

Tal como se verifica na Figura 32-a), a imagem apresenta um aspecto menos rugoso do que a imagem da Figura 32-b). Pode-se observar na Figura 32-b), a produção de estruturas visíveis de blocos (efeito "mosaico") tal como é referido em [13]. No entanto, o uso do filtro de remoção de artefactos na norma MPEG-4 AVC/H.264 minora em grande parte esta anomalia tal como comprova a Figura 32-a).

Após se terem estudado os conceitos e a codificação de vídeo e, em particular, a norma MPEG-4 AVC/H.264, está-se em óptimas condições para se proceder à implementação prática de algumas partes características de um codificador de vídeo. Deste modo, nos próximos dois capítulos são descritos os módulos desenvolvidos bem como a plataforma usada para a sua execução. Começa-se por descrever a plataforma de *hardware* usada, além de se assinalarem as ferramentas de *software* utilizadas para a elaboração dos módulos. Por fim, no capítulo seguinte, é apresentada uma descrição pormenorizada dos módulos desenvolvidos. São descritos os algoritmos utilizados, as interfaces dos módulos, diagramas temporais e o desempenho de cada um dos módulos desenvolvidos.

Capítulo 5. PLATAFORMA DE DESENVOLVIMENTO

O objectivo deste capítulo é descrever a plataforma de desenvolvimento usada para elaboração dos módulos apresentados nesta dissertação.

A plataforma de hardware usada foi uma placa PCMCIA designada por WILDCARD™II desenvolvida pela empresa Annapolis Micro Systems, Inc. Esta placa é composta por uma FPGA da família Virtex-II™ da Xilinx®, memórias ZBT-SRAM e DDR-SDRAM, ADC e interfaces de entrada e saída.

Para a programação da lógica programável foi usada a ferramenta de síntese de VHDL Synplicity® Synplify® Pro 7.6.1 e o pacote de desenvolvimento Xilinx® ISE™ 7.1. Para a simulação usou-se o simulador MentorGraphics® ModelSim™ 6.1e.

5.1- Wildcard™II

A placa Wildcard™II é uma placa PCMCIA do tipo II, Figura 33.

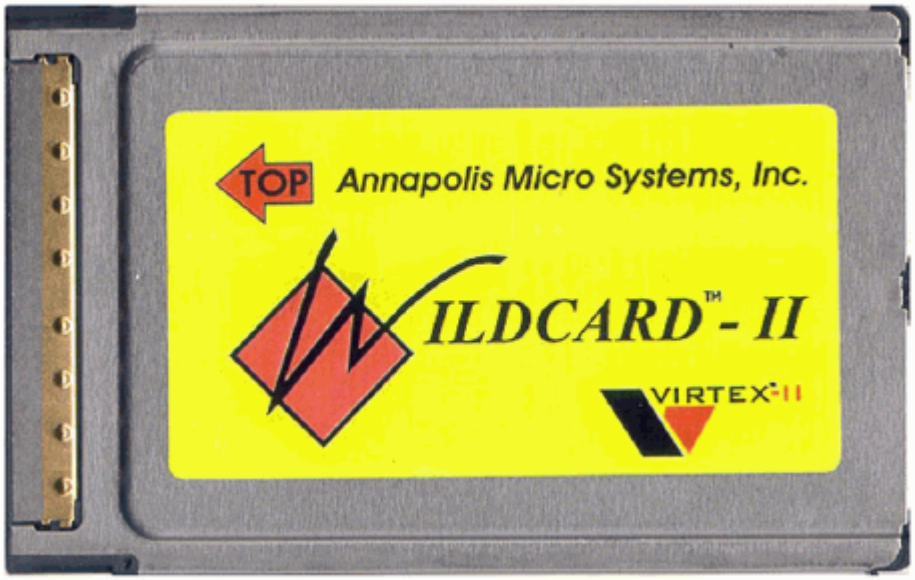

Figura 33 – Placa PCMCIA Wildcard™II.

Esta placa é constituída por uma FPGA Virtex-II™ XC2V3000, uma memória de 128MB ZBT-SRAM, uma memória de 64MB DDR-SDRAM, conversor ADC ADC9235, um controlador CardBus, interface PCI de 32bits@33MHz e interfaces I/O, Figura 34 [14].

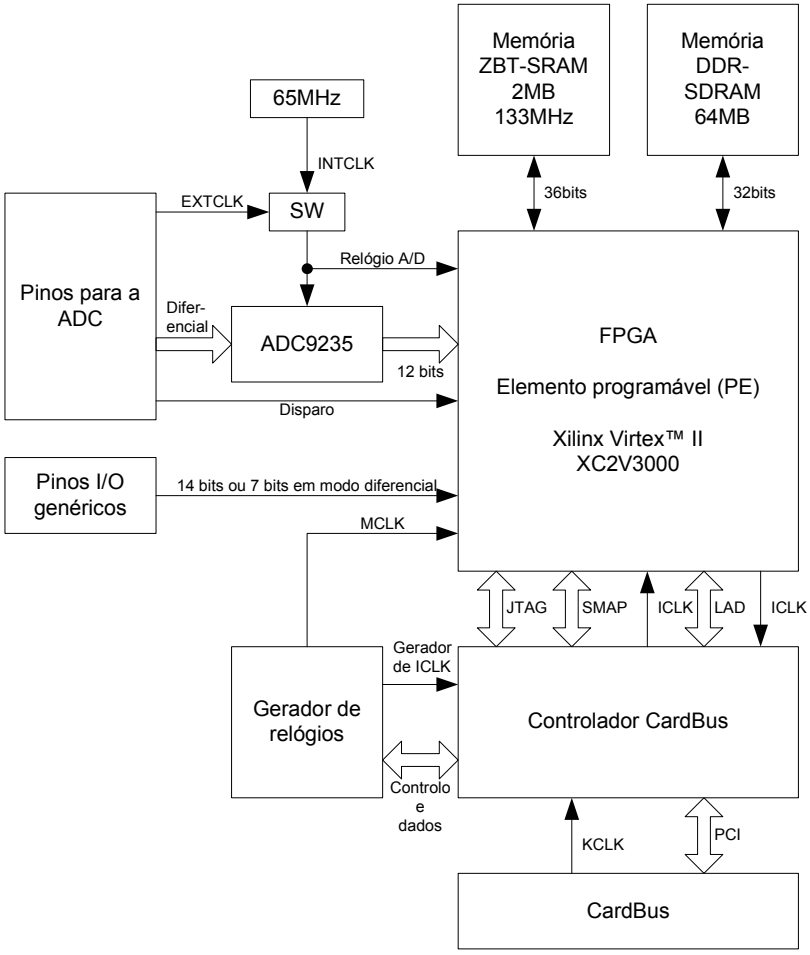

Figura 34 – Diagrama de blocos da WCII.

As funcionalidades relevantes da WCII para esta tese centram-se na FPGA, no controlador CardBus, interface PCI, gerador de relógios e nas memórias DDR-SDRAM e ZBT-SRAM.

O acesso a estas funcionalidades, bem como às restantes, é efectuado através de APIs de software e de modelos VHDL para a lógica programável, desenvolvidos pela Annapolis Micro Systems, Inc.

A API de software oferece um conjunto de funções em linguagem de programação C para aceder e controlar a WCII através do sistema anfitrião, que neste caso é um PC. Os

modelos VHDL permitem programar as funcionalidades da FPGA (PE – *Programming Elements*, elementos de programação) da WCII e fazer a sua verificação de funcionamento.

5.1.1 API de software

As bibliotecas da API, de software da WCII, dependem de determinados tipos de dados e estruturas responsáveis pela transmissão e recepção de informação de e para a placa especificada nas funções de chamamento. Estes tipos de dados então definidos no ficheiro *wcdefs.h* e devem ser incluídos nos ficheiros de programação C da aplicação desenvolvida [14].

Existem duas enumerações, WC_CardType e WC_MemBank. A enumeração WC_CardType serve para definir o tipo de família da Wildcard, isto é, se é WILDCARD ou WILDCARD_II, sendo para esta dissertação a última. A enumeração WC_MemBank está definida da seguinte forma:

```
typedef enum_WC_MemBank_
{
      WC_MemBank0=0,
      WC_MemBank1,
      WC_MemBank_Max
} WC_MemBank;
```

Na WCII o banco 0 representa a memória ZBT-SRAM, enquanto que o banco 1 representa a memória DDR-SDRAM.

Seguidamente mostram-se as definições das estruturas. De notar que a palavra-chave (*keyword*) DWORD é uma redefinição da palavra-chave **long int** da linguagem de programação C.

- WC_Version

```
typedef struct _WC_Version_
{
DWORD Hardware;
DWORD PeCore;
DWORD Api;
DWORD Driver;
DWORD Firmware;
} WC_Version;
```

Esta estrutura permite obter as versões da API, do *driver*, hardware, firmware e a versão da lógica programável.

- WC_Power

```
typedef struct _WC_Power_
{
FLOAT voltage; /* millivolts */
FLOAT current; /* milliamps */
FLOAT power; /* milliwatts */
} WC_Power;
```

Permite monitorar os consumos e alimentações da WCII.

- WC_DevConfig

```
typedef struct _WC_DevConfig_
{
DWORD PeXilinxType;
DWORD PeXilinxSpeedGrade;
DWORD PePackageType;
DWORD MemoryDwords[WC_NUMBER_MEM_BANKS];
DWORD MemorySpeedGrade[WC_NUMBER_MEM_BANKS];
} WC_DevConfig;
```

Permite obter informações acerca do hardware da WCII.

Um conjunto de APIs permitem controlar e comunicar com a WCII, através da aplicação desenvolvida. Cada API apresenta uma sintaxe de chamamento definida, que tem uma lista de parâmetros de passagem para as funções e códigos de retorno dessas funções.

A Tabela 5 contém uma breve descrição sobre as APIs da WCII. Para uma consulta mais detalhada consultar [14].

Tabela 5 – APIs de software da WCII.

API de software	Descrição
Funções gerais da API	
WC_Open	Abre o dispositivo WCII
WC_Close	Fecha o dispositivo WCII
WC_ErrorString	Retorna um texto descritivo do erro
WC_GetVersion	Devolve a versão dos vários componentes
WC_GetSerialNum	Devolve o número de série da WCII
WC_GetCardType	Determina se é um WILDCARD™ ou WILDCARD™-II
WC_DeviceInformation	Devolve informações sobre a WCII
Funções da API relacionadas com interrupções	
WC_IntReset	Reinicia a interrupção do PE
WC_IntQueryStatus	Obtém o estado das várias interrupções PE
WC_IntWait	Espera que aja uma interrupção
WC_IntEnable	Habilita as interrupções PE
Funções da API de memória	
WC_MemAlloc	Aloca memória para a WCII
WC_MemFree	Liberta a memória alocada pela WCII
Funções da API de programação da FPGA (PE)	
WC_PeProgram	Programa os PE
WC_PeProgramFromBuffer	Programa os PE a partir de um buffer
WC_PeRegWrite	Escreve para registos localizados no espaço de memória dos registos PE
WC_PeRegRead	Lê dos registos localizados no espaço de memória dos registos PE
WC_PeReset	Reinicia os PE
WC_PeGetPower	Reporta o consumo do PE
Funções da API relacionadas com os relógios	
WC_ClkSetFrequency	Estabelece a frequência do MCLK
Funções da API de monitoria da temperatura	
WC_PeGetTemperature	Obtém a temperatura de junção do PE
WC_PeGetTemperatureThresh	Obtém o limiar de temperatura estabelecido
WC_PeSetTemperatureThresh	Estabelece o limiar de temperatura
WC_PeQueryTemperatureEvents	Usado para verificar se ocorreram passagens pelo limiar de temperatura
WC_PeResetTemperatureEvents	Limpa o número de passagens pelo limiar de temperatura
Funções da API de monitoria da corrente eléctrica	
WCII_PeGetCurrentThresh	Obtém o limiar de corrente estabelecido
WCII_PeSetCurrentThresh	Estabelece o limiar de corrente
WCII_PeQueryCurrentEvents	Usado para verificar se ocorreram passagens pelo limiar de corrente
WCII_PeResetCurrentEvents	Limpa o número de passagens pelo limiar de corrente
Funções da API de DMA	
WCII_DmaMemAlloc	Aloca memória contígua para operações DMA
WCII_DmaMemFree	Liberta a zona de DMA
WCII_DmaUnbind	Desassocia o buffer DMA
WCII_DmaBind	Associa o buffer DMA
WCII_DmaErrorOp	Verifica erros no canal DMA

5.1.2 Modelos VHDL

Os modelos VHDL servem para definir a funcionalidade dos PEs reconfiguráveis, podendo depois ser usados para simular o comportamento do sistema para a sua verificação funcional.

Os modelos consistem em dois componentes principais, o ambiente do modelo de simulação VHDL (VHDL Model simulation environment) e as interfaces PE (PE interfaces).

- Ambiente do modelo de simulação VHDL

Este modelo, fornece um sistema completo de simulação. O ambiente de simulação é composto por três níveis: nível de desenho dos PE (PE Design), placa (Board), sistema (System). O nível de desenho dos PE é o primeiro e tem como objectivo simular o código VHDL desenvolvido. O nível seguinte é o da placa (Board level) e serve para simular as interligações entre os PEs e as memórias ZBT-SRAM e DDR-SDRAM, ADC e interfaces I/O. Por fim, o nível mais alto é o nível do sistema (System level) e este serve para simular as comunicações ente o sistema anfitrião e a WCII ou várias WCII.

Toda a simulação VHDL é controlada através de ficheiros modelo, Tabela 6. Estes modelos estão desenhados para funcionar no simulador ModelSim™.

Tabela 6 – Ficheiros modelo VHDL da WCII.

Ficheiro	Nível	Descrição
sim\pe_template_arch.vhd	Desenho PE	Arquitectura de topo do PE
sim\host_template_arch.vhd	Sistema	Arquitectura de topo do anfitrião
sim\io_template_arch.vhd	Sistema	Arquitectura de topo dos conectores I/O
sim\system_template_arch.vhd	Sistema	Arquitectura do sistema
sim\system_cfg.vhd	Sistema	Configuração do sistema
sim\project_vcom.de	-	Macro de compilação do ModelSim™
sim\project_vsim.do	-	Macro de simulação do ModelSim™
sim\pe.prj	-	Ficheiro de síntese do projecto para o Synplify™
sim\makefile	-	Makefile para colocação e routeamento no FPGA

A figura seguinte mostra as dependências e conteúdos dos três níveis de simulação.

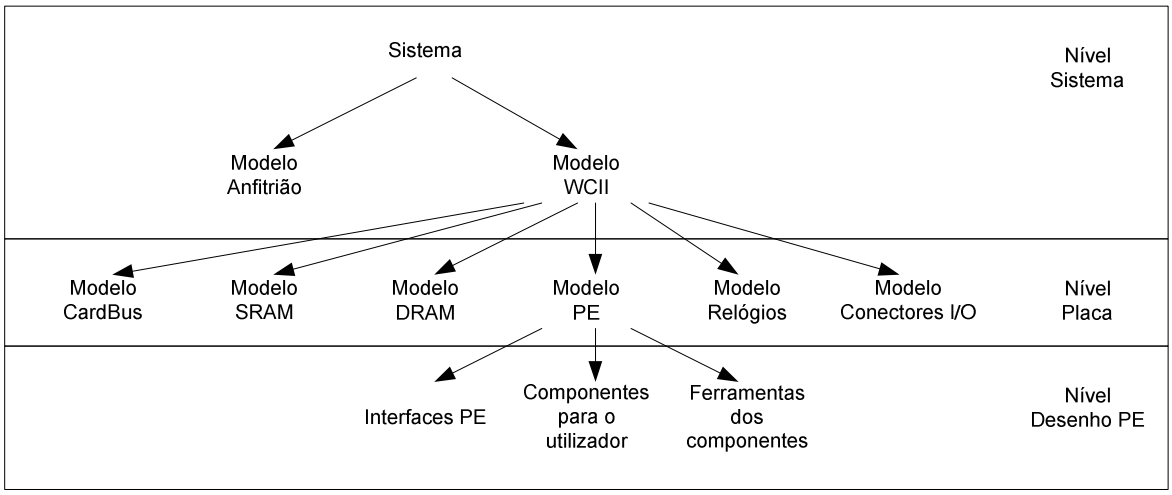

Figura 35 – Níveis de simulação da WCII.

5.1.3 Interfaces PE

As interfaces PE estão divididas em interfaces básicas e normalizadas. As primeiras permitem um acesso básico às funcionalidades dos PEs, enquanto que as outras permitem o controlo total dos PEs. Estes dois tipos de interfaces não podem ser usados em simultâneo para o mesmo PE.

O código gerado em VHDL é armazenado na FPGA da WCII, isto é, no bloco FPGA (Elemento programável - PE) mostrado na Figura 34. Para a programar, o anfitrião (*host*) acede ao controlador CardBus através do barramento PCI usando as funções disponibilizadas pela API.

Seguidamente resumem-se os tipos de interfaces PE disponibilizados na WCII. De notar, que só se referem os PE com relevância para esta dissertação. Uma descrição mais detalhada das interfaces encontra-se em [14].

- Clock_Std_IF

```
component Clock_Std_IF is
generic
(
      M_CLK_DLL_TYPE : Clk_DLL_Type := LOW_FREQ;
      CLKDV_DIVIDE : string := "2.0;
      DUTY_CYCLE_CORRECTION : string := "TRUE";
      CLOCK_FEEDBACK_2X : string := "FALSE";
      STARTUP_WAIT : string := "FALSE";
      DFS_FREQUENCY_MODE : string := "LOW";
      CLKFX_MULTIPLY : string := "4";
      CLKFX_DIVIDE : string := "1";
      CLKOUT_PHASE_SHIFT : string := "NONE";
      PHASE_SHIFT : string := "0";
      DSS_MODE : string := "NONE"
);
port
(
      Pads : inout Clock_Pads_Type;
      User_In : out Clock_Std_IF_In_Type;
      User_Out : in Clock_Std_IF_Out_Type
);
end component;

type Clock_Std_IF_In_Type is record
M_Clk : Clock_In_Type;
end record;

type Clock_Std_IF_Out_Type is record
M_Clk : Clock_Out_Type;
end record;

type Clock_In_Type is record
Clk : std_logic;
Clk90 : std_logic;
Clk180 : std_logic;
Clk270 : std_logic;
Clk2x : std_logic;
Clk2x180 : std_logic;
ClkDV : std_logic;
ClkFX : std_logic;
ClkFX180 : std_logic;
Locked : std_logic;
Status : std_logic_vector(7 downto 0);
PSDone : std_logic;
end record;
type Clock_Out_Type is record
Rst : std_logic;
DSSEn : std_logic;
PSIncDec : std_logic;
PSEn : std_logic;
PSClk : std_logic;
end record;
type Clk_DLL_Type is ( NONE, LOW_FREQ, HIGH_FREQ );
```

O Clock_Std_IF fornece acesso ao relógio da memória, M_Clk, e ao DCM da FPGA. O DCM, Digital Clock Manager (Controlador digital de relógio), é um elemento da família VIRTEX-II™ que têm como funcionalidades a síntese de frequências, mudanças de fase ao relógio e alinhamento de relógios (*de-skewing*). Mais à frente, apresenta-se uma descrição das funcionalidades da FPGA.

- LAD_Bus_Std_IF

```
Locked : std_logic;
Status : std_logic_vector(7 downto 0);
PSDone : std_logic;
end record;
type Clock_Out_Type is record
Rst : std_logic;
DSSEn : std_logic;
PSIncDec : std_logic;
PSEn : std_logic;
PSClk : std_logic;
end record;
type Clk_DLL_Type is ( NONE, LOW_FREQ, HIGH_FREQ );

type LAD_Bus_Std_IF_In_Type is record
Addr : std_logic_vector ( 18 downto 0 );
Data_In : std_logic_vector ( 31 downto 0 );
Reg_Strobe : std_logic;
DMA_Strobe : std_logic;
DMAInProgress : std_logic;
Write : std_logic;
PciRdy : std_logic;
BusGnt : std_logic;
Reset : std_logic;
RxClk_In : Clock_In_Type;
end record;
type LAD_Bus_Std_IF_Out_Type is record
Data_Out : std_logic_vector ( 31 downto 0 );
Strobe_Out : std_logic;
DMA_Init : std_logic;
PeRdy : std_logic;
IntReq : std_logic;
BusReq : std_logic;
RxClk_Out : Clock_Out_Type;
end record;
type Clock_In_Type is record
Clk : std_logic;
Clk90 : std_logic;
Clk180 : std_logic;
Clk270 : std_logic;
Clk2x : std_logic;
Clk2x180 : std_logic;
ClkDV : std_logic;
ClkFX : std_logic;
ClkFX180 : std_logic;
Locked : std_logic;
Status : std_logic_vector(7 downto 0);
PSDone : std_logic;
end record;
type Clock_Out_Type is record
Rst : std_logic;
DSSEn : std_logic;
PSIncDec : std_logic;
PSEn : std_logic;
PSClk : std_logic;
end record;
```

O barramento LAD Bus (barramento de dados, endereços e sinais de controlo entre o controlador CardBus e o bloco PE da WCII) é a forma primária de comunicar com o anfitrião do sistema. Através do LAD Bus, o PE pode receber e enviar informação ao anfitrião para as interfaces I/O e DMA.

O ciclo de escrita para o LAD Bus, para um espaço de registos é o seguinte:

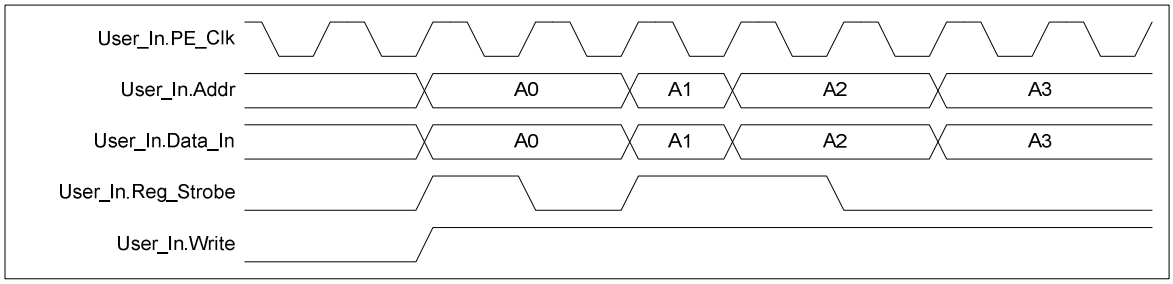

Figura 36 – Ciclo de escrita LAD Bus na WCII.

O início do ciclo começa quando o sinal User_In.Reg_Strobe está activo, isto é, está no nível alto (ou 1 lógico). Uma vez que o sinal User_In.Write está activo (1 lógico), então trata-se de um ciclo de escrita. São passados para os PE, os valores contidos em User_In.Addr (o endereço pretendido) e User_In.Data_In (o valor da escrita). O ciclo de escrita poderá ter no mínimo um ciclo de relógio, tal é mostrado no ciclo de escrita no endereço A1 na Figura 36.

E o ciclo de leitura para um espaço de registos é o seguinte:

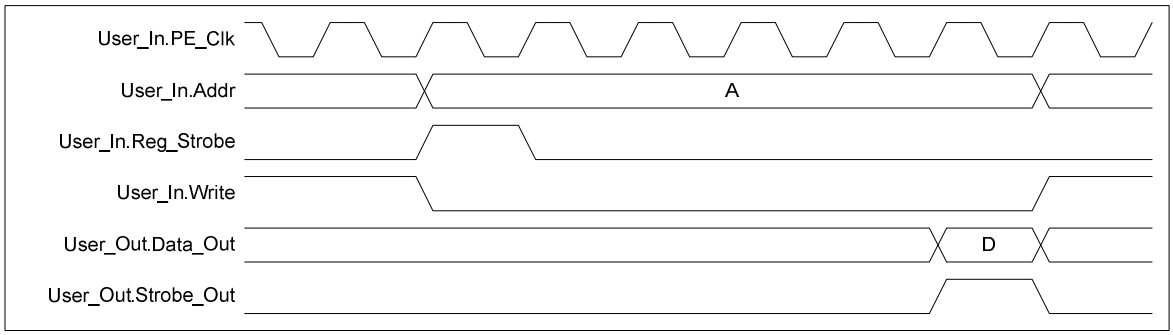

Figura 37 – Ciclo de leitura LAD Bus na WCII.

De notar que no ciclo de leitura, o fim de ciclo, User_Out.Strobe_Out = 1, é imposto pelo PE e não pelo anfitrião. Desta forma, a WCII pode retardar o ciclo de leitura para os casos onde o resultado de uma operação não seja imediato. No entanto, este atraso,

momento onde User_Out.Strobe_Out = 1, não deve exceder 64 ciclos de relógio. Isso poderia provocar bloqueios no barramento PCI do PC, o que poderia ser catastrófico.

Para uma descrição dos acessos DMA consultar [14].

- SRAM_Std_IF

```
component SRAM_Std_IF is
generic
(
CLK_DLL_TYPE : Clk_DLL_Type := LOW_FREQ
);
port
(
Pads : inout SRAM_Pads_Type;
Clk : in Clock_In_Type;
Global_Reset : in std_logic;
User_In : out SRAM_Std_IF_In_Type;
User_Out : in SRAM_Std_IF_Out_Type
);
end component;

type SRAM_Std_IF_In_Type is record
Data_In : std_logic_vector ( 35 downto 0);
Data_Valid : std_logic;
DMC_Clocked : std_logic;
end record;
type SRAM_Std_IF_Out_Type is record
Addr : std_logic_vector(20 downto 0);
Data_Out : std_logic_vector(35 downto 0);
Write : std_logic;
Strobe : std_logic;
Sleep : std_logic;
DCM_Reset : std_logic;
end record;
type Clk_DLL_Type is ( NONE, LOW_FREQ, HIGH_FREQ );
```

A interface SRAM_Std_IF provê o acesso à memória ZBT-SRAM. As seguintes figuras, mostram os acessos de escrita e leitura à ZBT-SRAM respectivamente, usando esta interface.

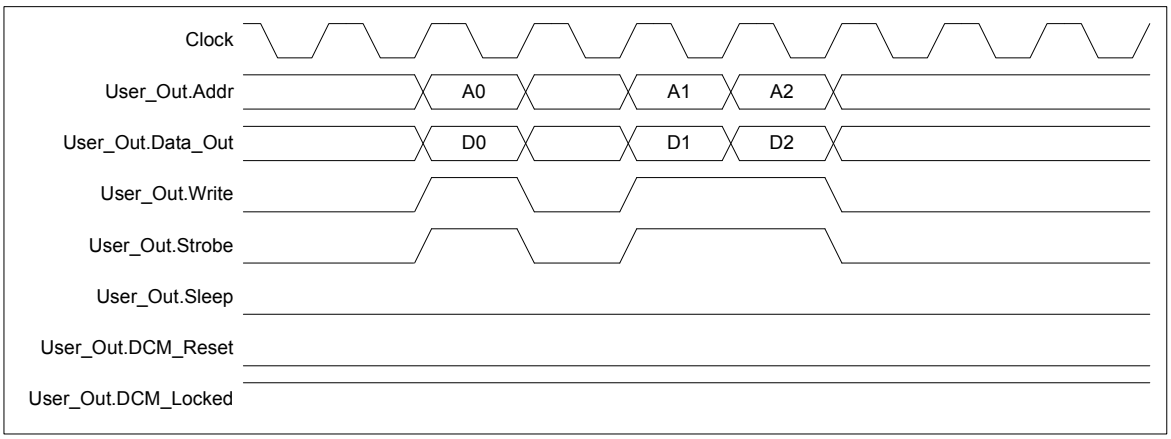

Figura 38 – Ciclo de escrita ZBT-SRAM na WCII.

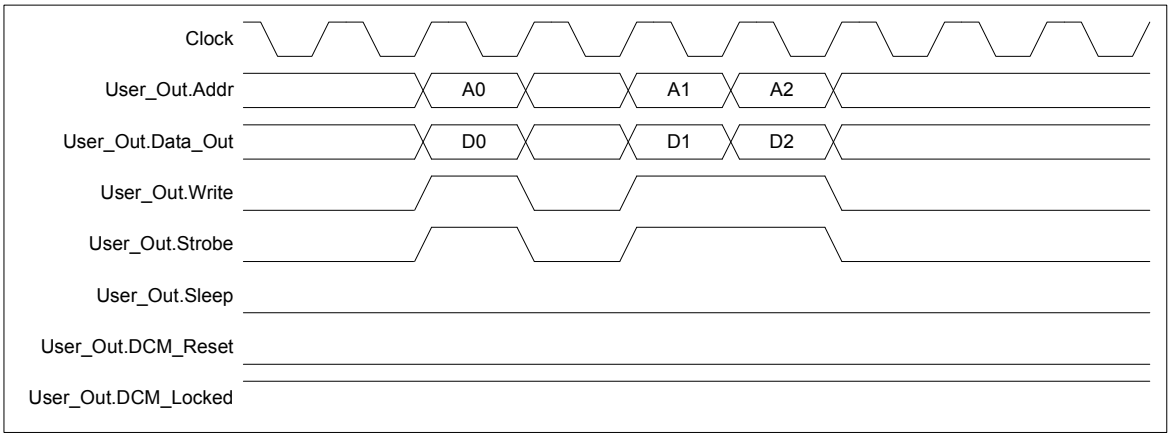

Figura 39 – Ciclo de leitura ZBT-SRAM na WCII.

- DRAM_Std_IF

```
component DRAM_Std_IF is
port
(
Pads : inout DRAM_Pads_Type;
M_Clk : in Clock_In_Type;
P_Clk : in std_logic;
Global_Reset : in std_logic;
User_In : out DRAM_Std_IF_In_Type;
User_Out : in DRAM_Std_IF_Out_Type
);
end component;

type DRAM_Std_IF_In_Type is record
Data_In : std_logic_vector(63 downto 0);
Data_Valid : std_logic;
Ready : std_logic;
Init_Done : std_logic;
end record;
type DRAM_Std_IF_Out_Type is record
Addr : std_logic_vector(22 downto 0);
Data_Out : std_logic_vector(63 downto 0);
Write : std_logic;
Strobe : std_logic;
end record;
```

Esta interface contém tudo o que é necessário para aceder e controlar a memória DDR-SDRAM. Desta forma, acede-se a esta memória sem qualquer preocupação com as questões de configuração, refrescamento e mudanças de bancos, linhas e colunas inerentes a todas as DDR-SDRAM. Isto é, esta interface fornece o acesso ao controlador DDR-SDRAM inserido na WCII.

As figuras seguintes mostram os ciclos de escrita e de leitura.

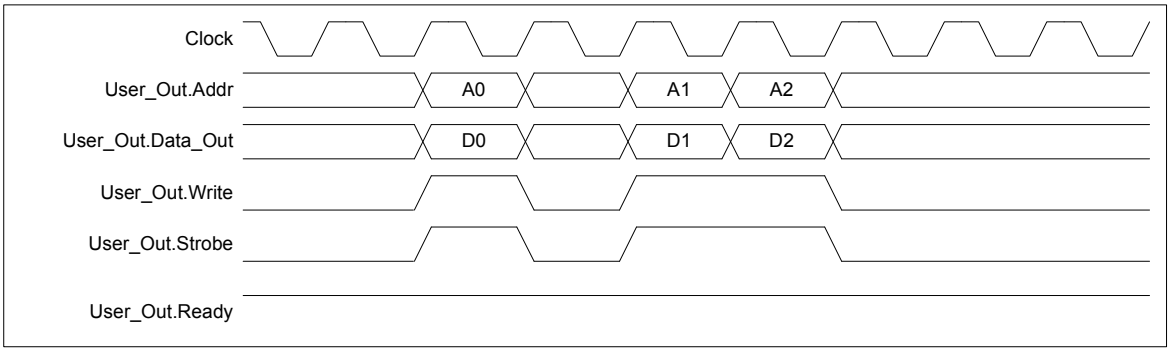

Figura 40 – Ciclo de escrita DDR-SDRAM da WCII.

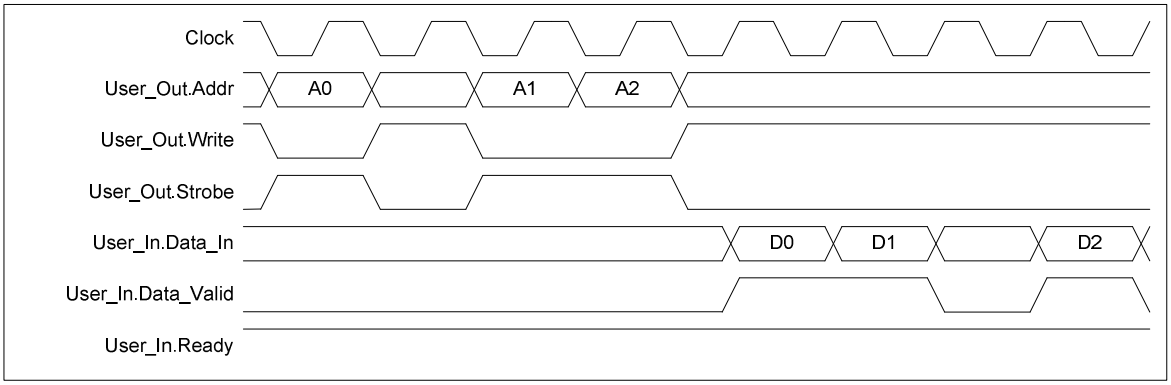

Figura 41 – Ciclo de leitura DDR-SDRAM da WCII.

Apesar de não serem relevantes para esta dissertação as outras interfaces disponíveis, refere-se aqui o seu nome e funcionalidade. Estas são: a interface IO_Std_IF que descreve a interface I/O e a interface AD_Std_If que descreve a interface ADC. Consultar [14] para uma descrição completa destas interfaces.

5.2- FPGA – Virtex-II™

A FPGA é um dispositivo semicondutor de lógica programável com uma estrutura muito semelhante a ASIC. Este tipo de dispositivos electrónicos são muito interessantes no desenvolvimento de protótipos de ASIC ou onde futuramente a FPGA vai ser substituída por uma ASIC. Tal verifica-se, porque uma FPGA permite o desenvolvimento relativamente rápido de uma aplicação onde o tempo é crucial, permitindo assim colocar no mercado mais rapidamente determinado produto (melhor *time-to-market*), podendo mais tarde substituir-se a FPGA por uma ASIC de menor custo [16][17].

Na WCII, encontra-se uma FPGA VIRTEX-II™ da Xilinx®, XC2V3000FG676-4 de 676 pinos [15]. A seguinte figura mostra a arquitectura desta família de FPGAs.

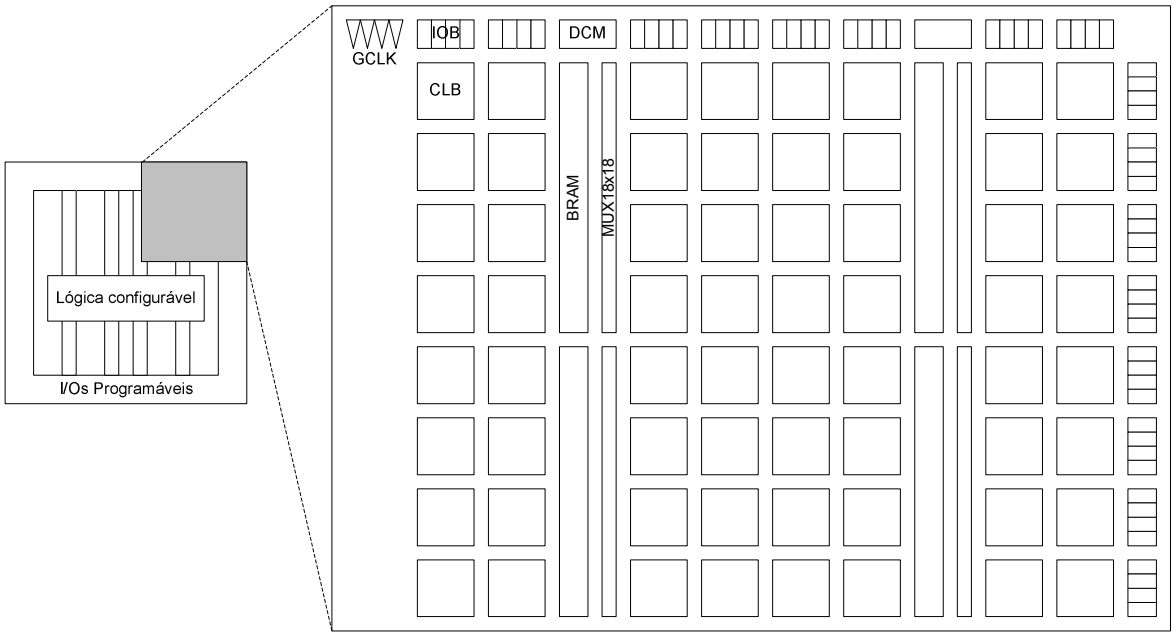

Figura 42 – Arquitectura de uma FPGA Virtex-II™ da Xilinx®.

A FPGA é composta por blocos I/O (IOBs) e blocos de lógica configurável (CLB). Os CLBs possuem elementos funcionais para lógica combinatória e sequencial, incluindo elementos básicos de memória. Além disso, esta FPGA contém blocos de memória dual-port, Block SelectRam (BRAM), blocos multiplicadores 18bit × 18bit dedicados, MUX18X18, e blocos para controlo de relógios, DCM (Digital Control Manager).

Os IOBs, Figura 43, são programáveis e são categorizados da seguinte forma:

- Blocos de entrada que podem ser SDR ou DDR

- Blocos de saída que podem ser SDR, DDR e ainda ser colocados em alta impedância (*buffers 3-state*)

- Blocos bidireccionais com qualquer combinação das categorias anteriores.

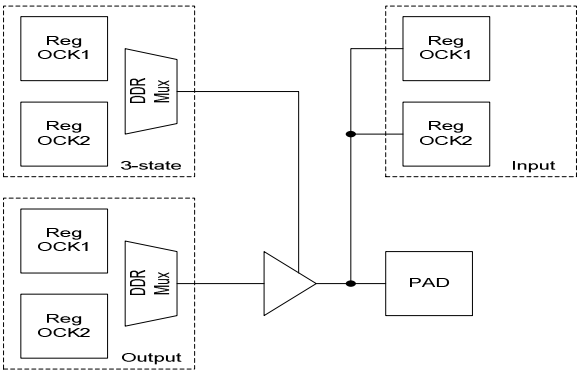

Figura 43 – Virtex-II™ IOB.

Os CLBs, Figura 44-a), são constituídos por: dois geradores de funções, dois elementos de armazenamento, portas de lógica aritmética, multiplexadores e outros tipos de arranjos lógicos. Os geradores de funções são configuráveis como LUTs de 4 entradas (4LUTs), registos de deslocamento de 16 bits ou ainda como memória distribuída de 16 bits.

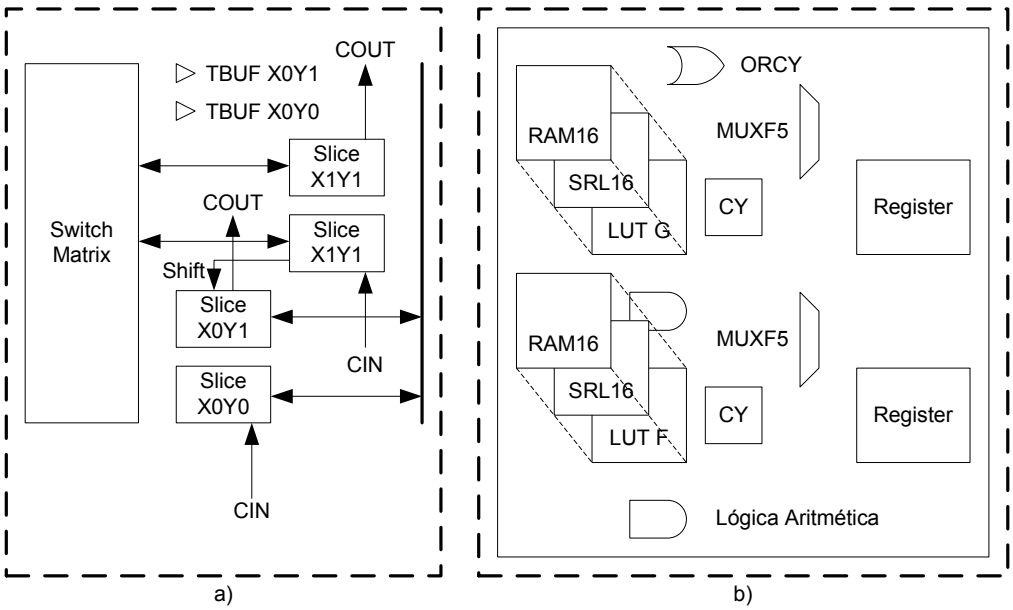

Figura 44 – a) Virtex-II™ CLB e b) Virtex-II™ *slice*.

Tal como mostra a Figura 44-a), um CLB é composto por quatro *slices*. Por sua vez, cada *slice* contém geradores de funções com quatro entradas, lógica aritmética com e sem transporte (*carry*), multiplexadores e dois elementos de armazenamento, Figura 44-b). Os geradores de funções de quatro entradas podem ser reconfigurados para funcionarem como: uma LUT de quatro entradas, registos de deslocamento (*shift registers*) de 16 bits ou ainda como memórias RAM (chamada também de memória distribuída) com 16 bits de espaço de armazenamento.

Os BRAMs são memórias RAM de acesso duplo (*dual-port* RAM), também podem ser memórias RAM de acesso simples (normal), com estruturas reconfiguráveis, i.e., a sua estrutura é flexível tal como mostra a seguinte tabela. Além disso, é possível interligar várias BRAMs em cascata para se obter determinada configuração.

Tabela 7 – Estruturas do BRAM da Virtex-II™.

Estrutura com acesso simples *Single-port* RAM	Estrutura com acesso duplo *Dual-port* RAM
16K × 1 bit	2K × 9 bits
8K × 2 bits	1K × 18 bits
4K × 4 bits	512K × 36 bits

Os DCMs são elementos do FPGA que permitem sintetizar frequências de relógios, fazer ajustes de fase a relógios e acertar os flancos de vários relógios de frequência idêntica (*de-skewing*).

A seguinte tabela mostra o número de elementos que a FPGA usada na WCII possui.

Tabela 8 – Constituição da FPGA VIrtex-II™ presente na WCII.

	CLB (1CLB = 4 slices = 128bits)				SelectRam		
Portas lógicas	Array Linhas × Colunas	Slices	RAM dist. (bits)	Mult.	BRAMs	Max. RAM	DCM
3M	64×56	14336	448	96	96	1728	12

O estudo pormenorizado das características da plataforma de desenvolvimento é fundamental para se desenvolverem os módulos aqui apresentados com eficiência. Assim, no próximo capítulo são descritos os seis módulos elaborados nesta dissertação. Os primeiros quatro módulos são dedicados à norma MPEG-4 Visual, enquanto que os últimos à norma MPEG-4 AVC/Visual. Os módulos desenvolvidos constituem algumas das funcionalidades mais representativas de cada uma destas normas.

Capítulo 6. MÓDULOS DESENVOLVIDOS

Nesta dissertação, foram desenvolvidos módulos em VHDL, de modo a funcionarem na WCII que efectuam alguns dos blocos típicos da codificação de vídeo MPEG-4 e MPEG-4 AVC/H.264.

Para a descodificação MPEG-4 foram desenvolvidos quatro módulos: VLD, IQ, IDCT e VLD+IQ+IDCT. O módulo VLD efectua a descodificação dos códigos de comprimento variável, VLC, na norma MPEG-4. O módulo IQ efectua o processo de quantificação inversa, e o módulo IDCT a transformada inversa de co-senos discreta. Estes três módulos foram projectados de forma a se interligarem uns aos outros para executarem algumas das partes características de um descodificador MPEG-4. O módulo VLD+IQ+IDCT representa isso mesmo.

Relativamente à norma MPEG-4 AVC/H.264 foram desenvolvidos dois módulos que representam uma grande fatia do peso computacional da codificação MPEG-4 AVC/H.264. Foi realizado o módulo *Deblocking* que implementa o processo de remoção de artefactos (*deblocking filter*) presente na norma MPEG-4 AVC/H.264. E por fim, o módulo de estimação de movimento, módulo Estimação de Movimento, de acordo com a norma MPEG-4 AVC/H.264. Os módulos desenvolvidos pela Universidade de Aveiro foram escolhidos em acordo com outros grupos da Univ. de Calgary, Xilinx, EPFL, Univ. Dublin, Univ. Dijon e Univ. Ghent. Estas instituições também contribuíram com outros módulos para a Paart 9 do MPEG-4, hardware de referência.

O teste comprovativo de funcionamento destes módulos foi efectuado, usando um software de referência para cada uma das normas de codificação de vídeo. Nesse software, as partes de código referentes às funcionalidades dos módulos, foram substituídas por chamadas de software ao módulo implementado na WCII.

O software de referência usado para os testes MPEG-4 foi o Microsoft® MPEG-4 Visual Reference Software, microsoft-v2.4-030710-NTU e o software de referência usado para o MPEG-4 AVC/H.264 foi o JM Reference Software, JM 10.2. Foram usadas várias sequências de vídeo para o teste de conformidade. O teste consistiu em comparar uma determinada sequência de vídeo descodificada, usando o software de referência, com a mesma sequência descodificada com o respectivo módulo presente na WCII, verificando, assim, se os resultados eram idênticos.

Nas secções seguintes, descreve-se detalhadamente o funcionamento, o algoritmo propriamente dito e a implementação de cada um destes módulos, sendo apresentada, no final de cada secção, uma tabela referente à ocupação de recursos do FPGA presente na WCII.

Em anexo, encontram-se os códigos VHDL de cada um dos módulos bem como as rotinas de software utilizadas para invocar os módulos implementados.

6.1- Módulo VLD

O módulo VLD desenvolvido em VHDL serve para descodificar códigos de comprimento variável em MPEG-4 num fluxo de dados (*stream*) [18]. Foi realizado de forma a poder ser integrado com o módulo IDCT e o módulo IQ, desenvolvidos nesta dissertação.

Este módulo é constituído por quatro sub-módulos responsáveis pela descodificação dos coeficientes DC, coeficientes não DC de macroblocos Intra, coeficientes não DC de coeficientes Inter, e um sub-módulo responsável pelo controlo do processo de descodificação.

6.1.1 Algoritmo

O algoritmo usado para este módulo está representado na próxima figura.

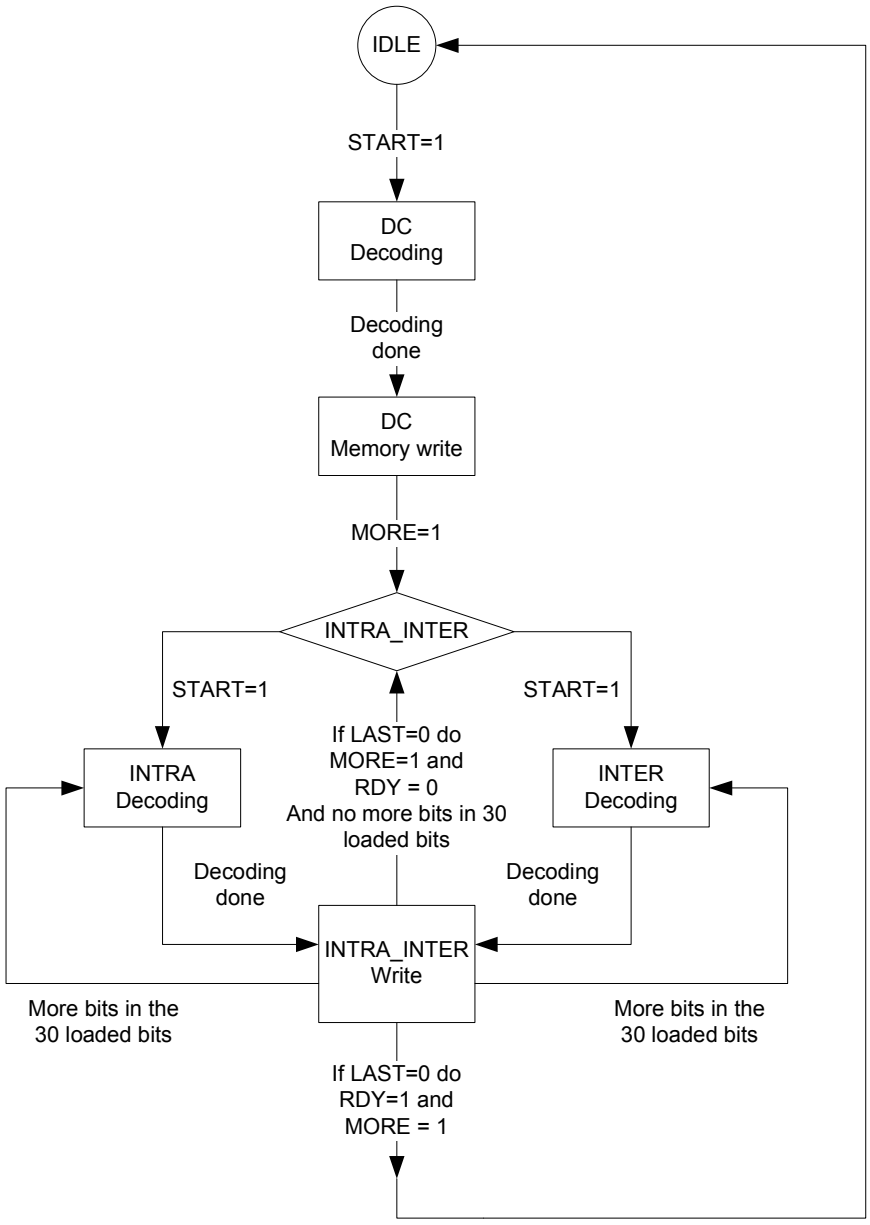

Figura 45 – Algoritmo do módulo VLD.

Inicialmente o módulo está na posição de repouso (IDLE). Assim que é detectado um sinal START = 1, é iniciado o processo de descodificação DC. Após ter sido descodificado o coeficiente DC, o módulo envia para a sua saída o coeficiente e assinala ao dispositivo responsável pela inserção da *stream*, para enviar mais informação, MORE = 1. Neste momento, o módulo está à espera de mais bits. Dependendo da sinalização

INTRA_INTER, o módulo passa à descodificação de coeficientes Intra ou Inter. Assim que seja detectado o último elemento, o módulo assinala que finalizou, RDY = 1.

No processo de descodificação dos coeficientes DC, todos os bits da *stream*, com significado para a descodificação, são analisados de uma só vez de acordo com uma LUT previamente construída. No entanto, o processo de descodificação dos outros coeficientes já não pode ser efectuado de uma forma tão simples. Deste modo, desenvolveu-se o seguinte algoritmo para a descodificação dos restantes coeficientes, quer sejam Intra ou Inter.

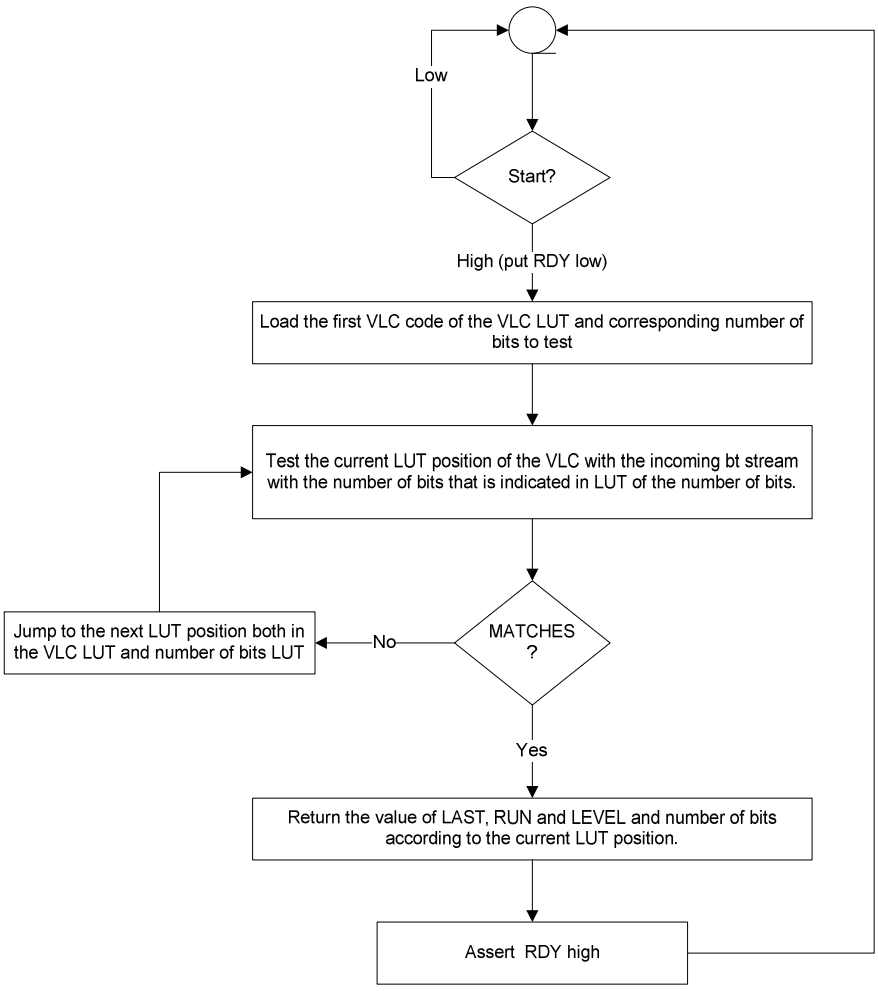

Figura 46 – Algoritmo para a descodificação de coeficientes não DC.

Apesar de não estar presente nas figuras, o módulo desenvolvido também procura por códigos de escape da norma MPEG-4. O módulo verifica se existe algum tipo de código de escape MPEG-4 na *stream* antes de proceder ao processo de comparação com as LUTs internas. Desta forma, aumenta-se o desempenho do módulo. Se é detectado um

código de escape MPEG-4 do tipo 1 ou 2 e não se está na presença de uma *stream* short_video_header, o processo de descodificação é o mesmo do da figura acima. Caso contrário, a descodificação é efectuado num só ciclo de relógio.

Seguidamente, explica-se melhor o funcionamento do módulo de descodificação.

6.1.2 Implementação

O módulo é composto por quatro sub-módulos, tal como foi descrito anteriormente. Cada um destes módulos é baseado em FSM e o processo de descodificação é feito por comparação com tabelas LUT, Figura 47.

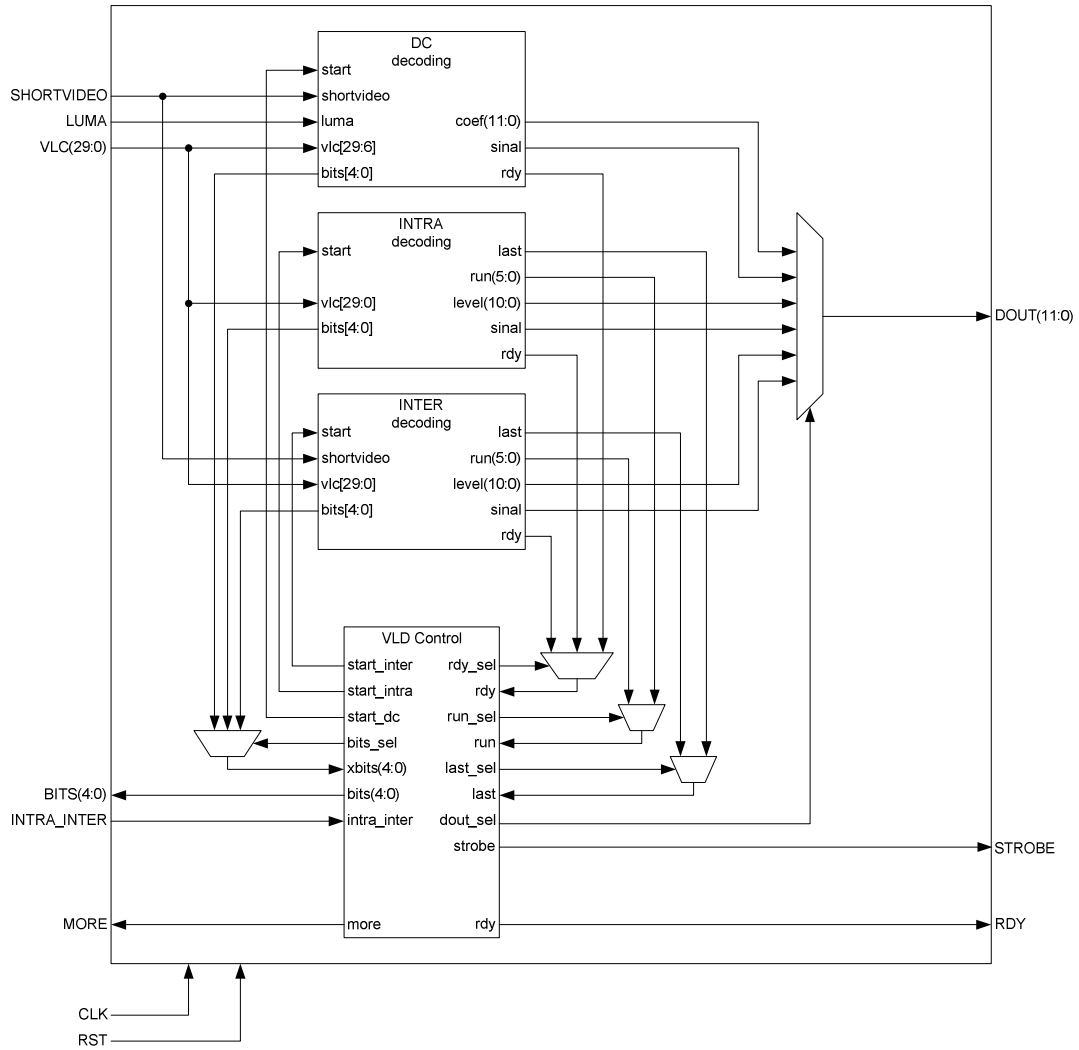

Figura 47 – Diagrama de blocos do módulo VLD.

Todos os sub-módulos são baseados em FSM. O processo de descodificação é baseado em comparações da *stream* de entrada, VLC(29:0), com LUTs previamente carregadas. Sempre que um código é detectado, o módulo converte a codificação LAST-RUN-LEVEL num conjunto de coeficientes. É normal que a grande maioria dos coeficientes sejam nulos. Durante o processo de descodificação, o módulo pede bits sucessivos ao dispositivo responsável pelo carregamento da *stream*, sendo carregados 30 bits de cada vez. Presume-se que a *stream* seja composta só por códigos VLC, representando coeficientes, i.e., sequências de blocos.

O processo de descodificação é iniciado quando START = 1 e são recolhidos 30 bits da *stream*, VLC(29:0). Depois, o módulo procura nas suas LUT, previamente construídas, um código VLC que coincida com os 30 bits de entrada, tendo em conta se é um macrobloco Intra ou Inter, sinal INTRA_INTER, e se é uma *stream* MPEG-4 short_video_header, sinal SHORTVIDEO. Assim que é encontrado determinado VLC, o módulo analisa mais bits dos 30 bits carregados. Se não for encontrado um código VLC nos restantes bits, o módulo requer mais 30 bits através do sinal MORE. Sempre que um código é descodificado, o módulo assinala o evento através da sua saída STROBE, indicando que DOUT[11:0] é válido. Se entretanto for detectado um código com sinal LAST = 1, o módulo termina o processo e assinala RDY = 1.

O módulo coloca na saída o número de bits descodificados, BITS(4:0), para o dispositivo encarregue de fornecer a *stream* saber a partir de que posição deve carregar mais 30 bits.

Seguidamente, apresenta-se o diagrama temporal deste módulo.

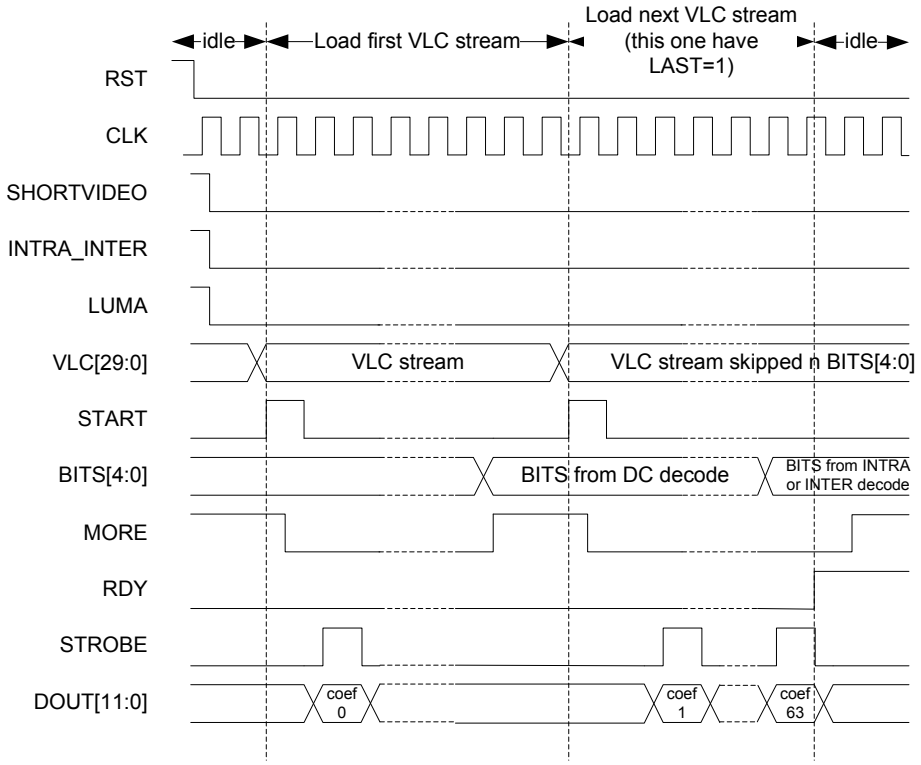

Figura 48 – Diagrama temporal do módulo VLD.

O diagrama mostra um exemplo de dois códigos VLC presentes numa *stream* para serem descodificados. O primeiro, corresponde ao código VLC referente ao coeficiente DC e o último a um código VLC onde é encontrado LAST = 1 terminando assim o processo de descodificação dos 64 coeficientes.

6.1.3 Resultados

A percentagem de ocupação e desempenho da FPGA é a seguinte:

Tabela 9 – Percentagem de ocupação da FPGA do módulo VLD.

CLB slices	1506
Slices Flip Flops	426
4LUTs	2727
BRAM	1
Freq. de relógio máxima	92MHz

A ocupação de recursos da FPGA é relativamente baixa e o desempenho elevado. É utilizado um recurso da FPGA, SelectRam (BRAM), usado para o armazenamento

temporário dos 30 bits da *stream*. O número relativamente alto de 4LUTs, deve-se ao facto de serem usadas LUTs com os vários códigos VLC, para acelerar o processo de busca.

Comparativamente, obtém-se ganhos ao nível da percentagem de ocupação de recursos da FPGA em relação os resultados obtidos [19]. Em [19] a percentagem de ocupação é de 51% dos elementos de lógica programável (ou CLBs). No módulo desenvolvido, a percentagem de ocupação é de (1506/14336) × 100% = 10.5%. O que resulta num ganho de aproximadamente 5 vezes do obtido em [19].

Relativamente à velocidade do módulo, verifica-se que o número máximo de símbolos por segundo é de 21Msimbolos/s (mega símbolos por segundo). E uma vez que os códigos VLC MPG-4 podem ter um comprimento de 30bits, o débito máximo do módulo é na ordem dos 630Mbps (21Msimbolos/s × 30bits = 630Mbps), apresentando velocidades muito próximas das obtidas por [19] e [20], que são de 720Mbps e 810Mbps, respectivamente.

Verifica-se, assim, que o módulo realizado, devido à sua baixa utilização de recursos de lógica pode ser uma solução interessante para aplicações de baixo consumo ou de baixo custo. Além disso, e como ocupa poucos recursos, é interessante para se integrar em circuitos integrados do tipo SoC (*System on Chip*), e como suporta velocidades de relógio até 92MHz, pode conviver relativamente bem dentro de um destes integrados.

6.2- Módulo IQ

Para a realização do módulo IQ [21], teve-se como principal objectivo desenvolver um módulo de quantificação inversa MPEG-4 de alto rendimento. Este módulo permite quantificar blocos de 64 elementos em apenas 600ns, usando 2% dos recursos da FPGA utilizada na WCII. Mais uma vez, no desenvolvimento deste módulo, foi tido em consideração o facto de este módulo vir a ser integrado com outros módulos para efectuar a descodificação MPEG-4.

6.2.1 Algoritmo

A quantificação em codificação de vídeo consiste em descartar selectivamente informação visual, sem introduzir perdas significativas de qualidade de imagem. Assim, o

quantificador remove informação visual que não seja perceptível ao observador. Esta forma de remover informação que é ignorada pelo olho humano, representa um dos aspectos fundamentais da compressão de vídeo. Reduz-se a necessidade de espaço de armazenamento, aumentando a largura de banda para a transmissão de vídeo. Além disso, a quantificação reduz o número de bits necessários para representar os coeficientes da DCT, provocando a principal perda de informação nos algoritmos de compressão de vídeo. Perdas menores só são obtidas com um "casamento" perfeito entre a função de quantificação e a informação estatística da fonte de dados, [22] e [23]. Um quantificador poder ser um valor constante a ser aplicado a um conjunto de coeficientes da DCT, ou então uma matriz, e neste caso MPEG-4 uma matriz 8×8, onde cada elemento é aplicado ao seu correspondente coeficiente espacial.

Quando a DCT é aplicada a um bloco de 8×8 pixel, o resultado é um conjunto de frequências espaciais. Uma vez que o olho humano é menos sensível a detalhes de alta frequência, a quantificação é maior sem que haja perda de qualidade notada e aumento da compressão. Similarmente, como o olho humano também é menos sensível às componentes de cor do que à de luminância, a quantificação nestas componentes também é maior.

Na norma MPEG-4 é possível aplicar dois processos de quantificação [1]. O primeiro, MPEG *Quantization*, é derivado da norma de vídeo MPEG-2, enquanto que o outro, H-263 *Quantization*, é usado na recomendação ITU-T H.263. O método a ser usado é decidido no codificador, que por sua vez envia essa informação ao descodificador. Na codificação MPEG-4 a quantificação do coeficiente DC é feita por passo de quantificação fixo.

O passo de quantificação é controlado pelo parâmetro quantizer_scale, Qp, que pode tomar valores entre 1 a $2^{quant\_precision}$-1 que é codificado uma vez por VOP. O parâmetro quant_precision, especifica o número de bits usados para representar os parâmetros de quantificação e toma valores de 3 a 9.

Antes de ser aplicada a quantificação inversa, os resultados da quantificação, F''[i][j] devem ser saturados da seguinte forma:

$$F'[i][j] = \begin{cases} 2^{bits\_per\_pixel+3} - 1, & \text{if } F''[i][j] > 2^{bits\_per\_pixel+3} - 1 \\ F''[i][j], & -2^{bits\_per\_pixel+3} \le F''[i][j] \le 2^{bits\_per\_pixel+3} - 1 \\ -2^{bits\_per\_pixel+3}, & \text{if } F''[i][j] < -2^{bits\_per\_pixel+3} \end{cases} \quad \textbf{(6.2.1)}$$

Os coeficientes DC de macroblocos Intra são quantificados, usando um método de quantificação não linear, optimizado de acordo com a seguinte tabela.

Tabela 10 – Valores de quantificação inversa MPEG-4 para o coeficiente DC.

quantizer_scale, Qp	1 - 4	5 - 8	9 - 24	25 – 31
dc_scaler(luminancia)	8	2Qp	Qp + 8	2Qp - 16
dc_scaler(crominância)	8	(Qp + 13)/2	(Qp + 13)/2	Qp - 6

O valor DC resultante da quantificação inversa é então dado por:

$$F''[0][0] = QF[0][0].dc\_scaler \quad \textbf{(6.2.2)}$$

onde QF[0][0] representa o coeficiente DC quantificado.

- MPEG *Quantization*

A vantagem da quantificação MPEG *Quantization* é que o quantificador toma em consideração as propriedades do olho humano. O quantificador MPEG permite a adaptação do passo de quantificação de cada coeficiente, através do uso de matrizes de pesos, Figura 49.

$$\begin{bmatrix} 8 & 17 & 18 & 19 & 21 & 23 & 25 & 27 \\ 17 & 18 & 19 & 21 & 23 & 25 & 27 & 28 \\ 20 & 21 & 22 & 23 & 24 & 26 & 28 & 30 \\ 21 & 22 & 23 & 24 & 26 & 28 & 30 & 32 \\ 22 & 23 & 24 & 26 & 28 & 30 & 32 & 35 \\ 23 & 24 & 26 & 28 & 30 & 32 & 35 & 38 \\ 25 & 26 & 28 & 30 & 32 & 35 & 38 & 41 \\ 27 & 28 & 30 & 32 & 35 & 38 & 41 & 45 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ 19 & 20 & 21 & 22 & 23 & 24 & 26 & 27 \\ 20 & 21 & 22 & 23 & 25 & 26 & 27 & 28 \\ 21 & 22 & 23 & 24 & 26 & 27 & 28 & 30 \\ 22 & 23 & 24 & 26 & 27 & 28 & 30 & 31 \\ 23 & 24 & 25 & 27 & 28 & 30 & 31 & 33 \end{bmatrix}$$

(b)

Figura 49 – Matriz de pesos para a quantificação inversa a) Intra e b) Inter.

A primeira quantificação inversão efectuada da seguinte forma:

$$F''[i][j] = \begin{cases} 0, & \text{if } QF[i][j] = 0 \\ ((2.QF[i][j] + k) \times W[i][j] \times \text{quantiser\_scale})/16, & \text{if } QF[i][j] \neq 0 \end{cases} \quad \textbf{(6.2.3)}$$

onde

$$k = \begin{cases} 0 & \text{for intra coded blocks} \\ \text{sign}(QF[i][j]) & \text{for inter coded blocks} \end{cases} \quad \textbf{(6.2.4)}$$

QF[i][j] representa coeficientes quantificados e W[i][j] representa a matriz de pesos.

Por fim, deve ser aplicado um controlo de desemparelhamento (mismatch) da seguinte forma:

$$F[7][7] = \begin{cases} F'[7][7], & \text{if sum is odd} \\ \left.\begin{cases} F'[7][7] - 1, & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1, & \text{if } F'[7][7] \text{ is even} \end{cases}\right\} & \text{if sum is even} \end{cases} \quad \textbf{(6.2.5)}$$

- H.263 *Quantization*

Neste método, não é aplicada a matriz de pesos, reduzindo assim a complexidade computacional e não se obtém o mesmo desempenho do método anterior.

$$F''[i][j] = \begin{cases} 0, & \text{if } QF[i][j] = 0 \\ (2.|QF[i][j]|+1)\times \text{quantiser\_scale}, & \text{if } QF[i][j] \neq 0, \text{quantiser\_scale is odd} \\ ((2.|QF[i][j]|+1)\times \text{quantiser\_scale})-1, & \text{if } QF[i][j] \neq 0, \text{quantiser\_scale is odd} \end{cases} \quad \textbf{(6.2.6)}$$

O sinal de QF[i][j] é dado da seguinte forma:

$$F''[i][j] = \text{Sign}(QF[i][j]) \times |F''[i][j]| \quad \textbf{(6.2.7)}$$

6.2.2 Implementação

A seguinte figura mostra o diagrama de blocos do módulo VHDL desenvolvido.

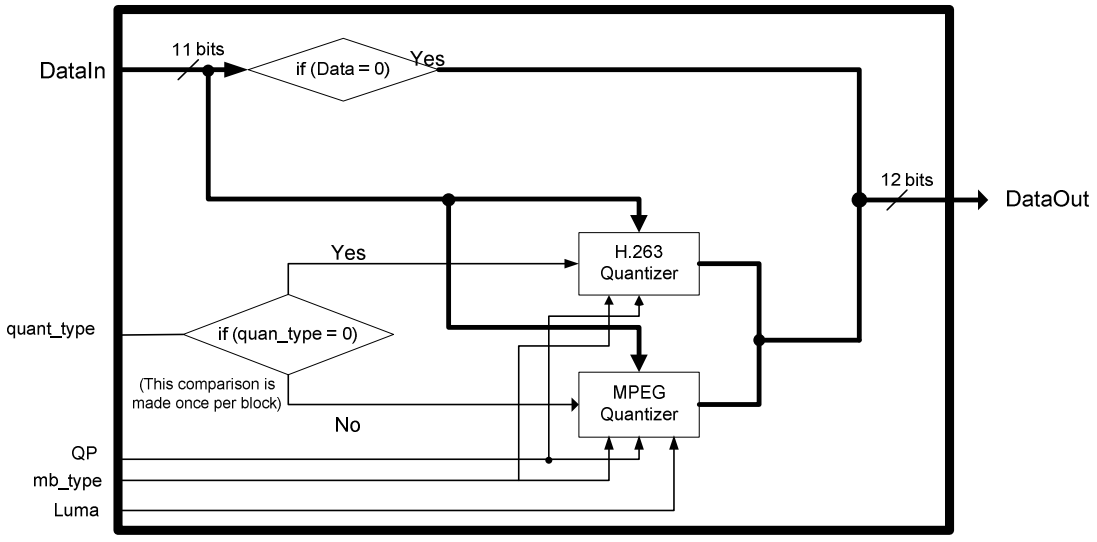

Figura 50 – Diagrama de blocos do módulo IQ.

Os sinais de entrada quant_type, QP, mb_type e Luma definem o tipo de quantificação (MPEG-4 ou H.263), o parâmetro de quantificação, o tipo de macrobloco (Intra ou Inter) e se é um bloco de luminância ou de crominância, respectivamente. Na

entrada DataIn são inseridos os coeficientes por quantificar e à saída DataOut obtém-se os coeficientes quantificados. Existe um sinal de relógio CLK usado pelo módulo.

O seguinte figura mostra o processo de quantificação MPEG-4 desenvolvido.

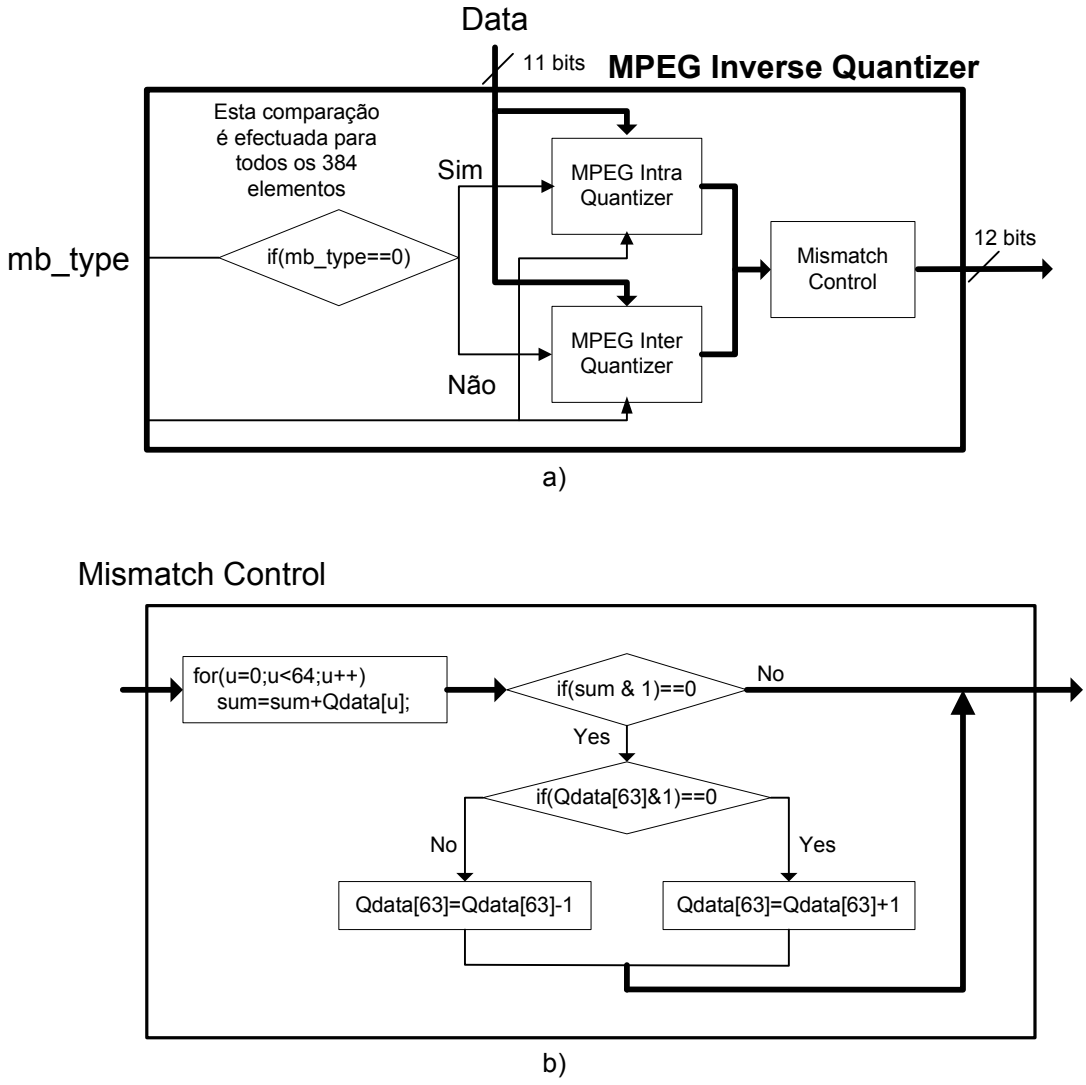

Figura 51 – a) Fluxograma de quantificação inversa para MPEG-4 e b) controlo m*ismatch*.

Seguidamente mostra-se o fluxograma para a quantificação H.263.

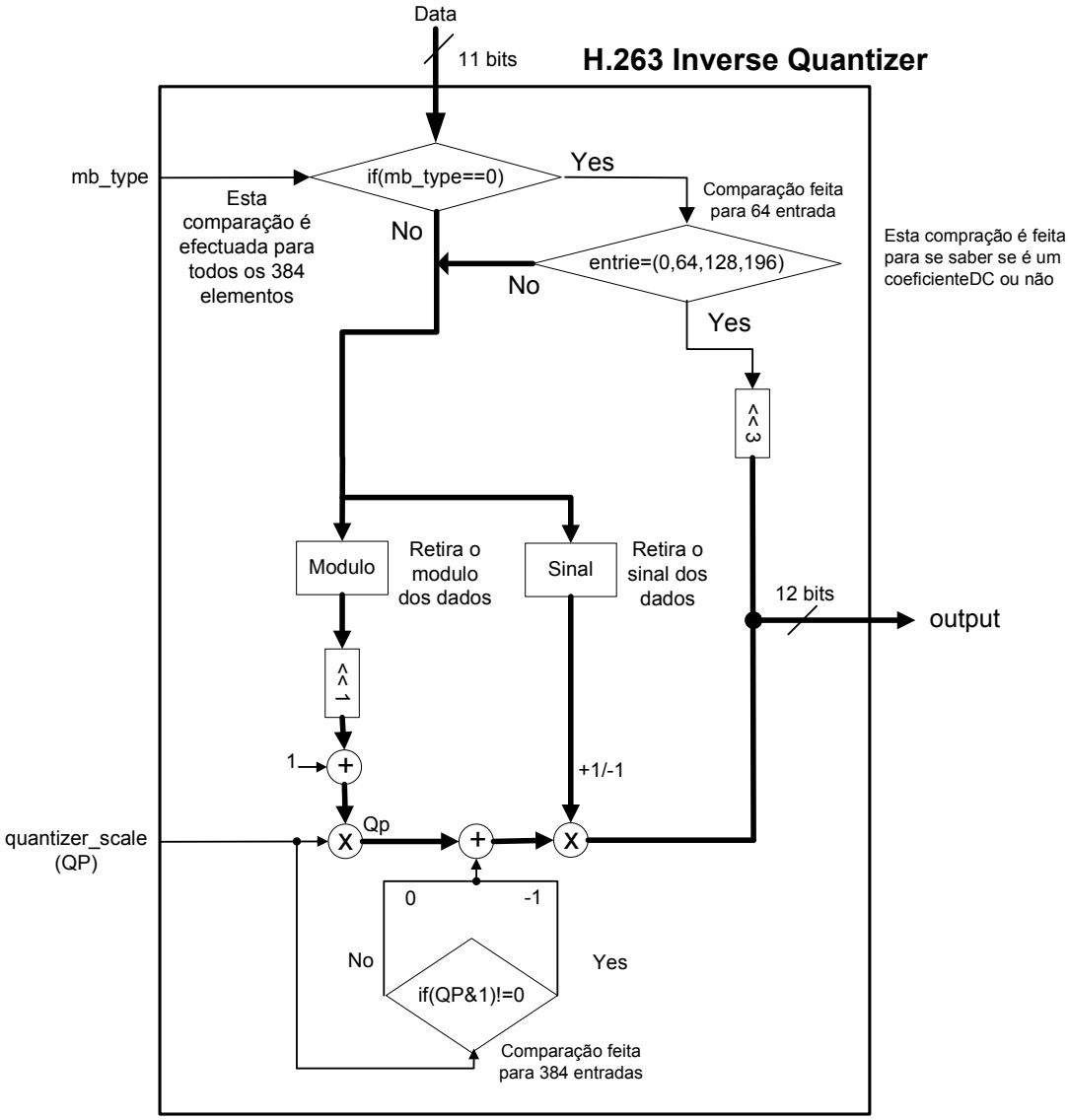

Figura 52 – Fluxograma para quantificação inversa H.263.

O diagrama temporal deste módulo está representado na seguinte figura.

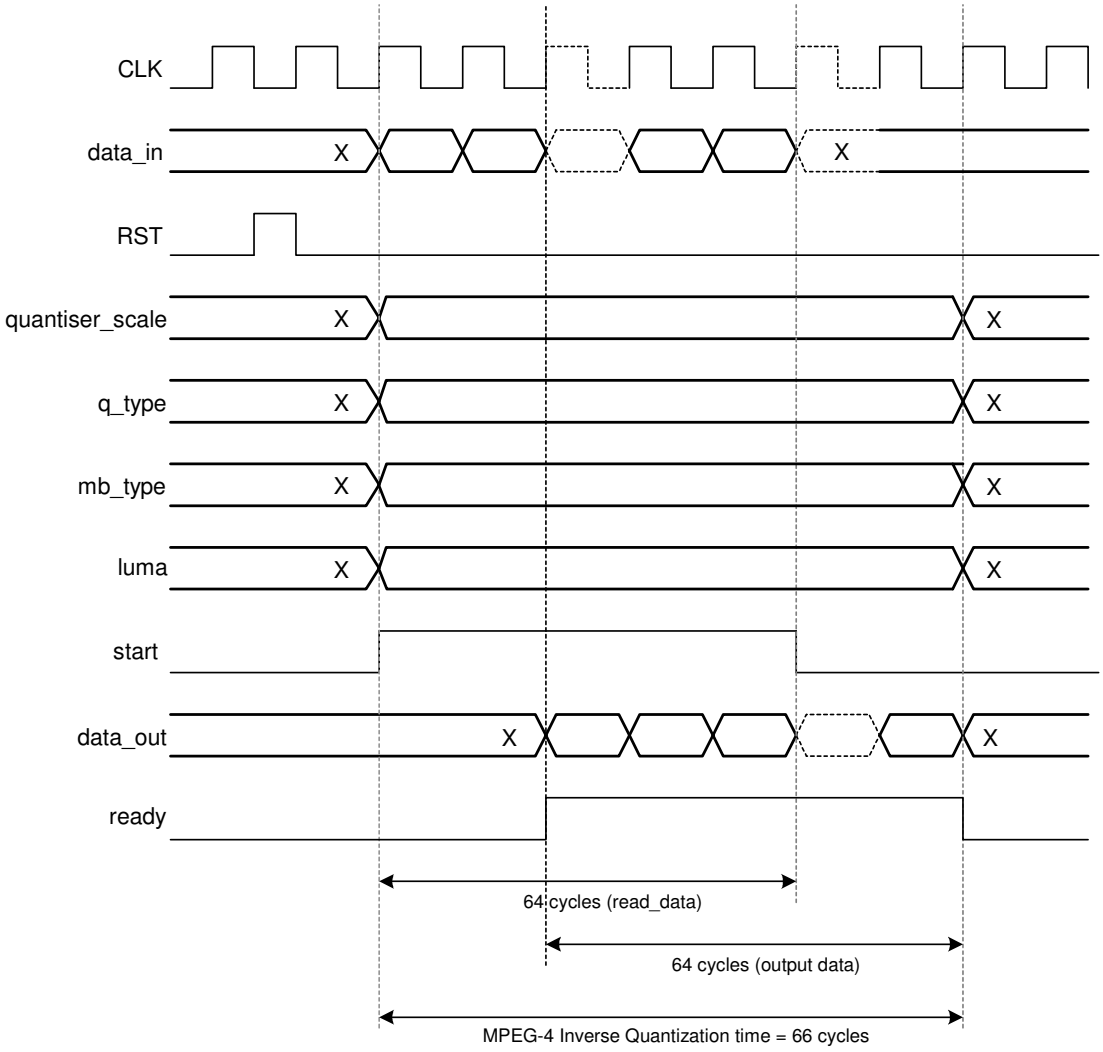

Figura 53 – Diagrama temporal do módulo IQ.

Verifica-se que existe uma latência de 2 ciclos de relógio CLK para que se dê a primeira quantificação.

6.2.3 Resultados

A seguinte tabela mostra a percentagem de ocupação da FPGA bem como o seu desempenho.

Tabela 11 – Recursos do FPGA ocupados pelo módulo IQ.

CLB slices	318
Slices Flip Flops	80
4LUTs	578
MUX18X18	11
Freq. de relógio máxima	98MHz

O módulo desenvolvido apresenta um desempenho de 1.48Mblocos/s e uma latência de 2 ciclos de relógio. E tal como em [24], são necessários 66 ciclos de relógio para processar um macrobloco. No entanto, em [24] são necessárias 5800 portas lógicas (*gates*) enquanto que no módulo desenvolvido, são necessárias 3028 portas lógicas (*gates*). Esta dissertação, apresenta um ganho no número de portas lógicas relativamente a [24]. Contudo, para este módulo são necessários 11 multiplicadores dedicados com dimensão 18×18 bits.

Verifica-se, assim, que apesar de não existirem ganhos de velocidade, o módulo desenvolvido apresenta um bom desempenho em termos dos recursos usados.

6.3- Módulo IDCT

No processo de descodificação/codificação de vídeo a IDCT é uma das partes que requer mais poder computacional [25]. Desta forma, para se obter vídeo MPEG em tempo real, é necessário obter-se uma IDCT rápida para acelerar o processo.

O algoritmo da IDCT inteira foi publicado em [26] e no ICIP 2005 embora tivesse sido baseado em [27]. A IDCT desenvolvida calcula muito rapidamente todos os 64 coeficientes de um macrobloco e está limitada unicamente pela velocidade máxima da lógica combinatória do FPGA utilizado.

6.3.1 Algoritmo

A grande maioria das normas de codificação de vídeo, baseados em modelos híbridos de compensação de movimento, usa a DCT no codificador para remover a redundância da imagem. Dado que a DCT é uma parte central da codificação de vídeo, todos os algoritmos de vídeo baseados na DCT beneficiam de uma DCT de computação rápida. Vários algoritmos de DCT (IDCT) *floating-point* foram propostos e normalmente estão classificados do seguinte modo: método indirecto e método directo. Sendo o primeiro caracterizado pelo calculo da DCT, através de FFT ou outras transformadas, e o método directo caracterizado pelo calculo da DCT, pela factorização matricial ou calculo recursivo.

A aproximação convencional para calcular uma 2D-DCT (DCT bidimensional) de dimensão N×N pontos através dos métodos directos, que segue o método linha-coluna, necessita de 2N conjuntos de 1D-DCT (DCT unidimensional) de N pontos. No entanto, técnicas que calculem realmente uma 2D-DCT, isto é, transformadas bidimensionais, são mais eficientes que os métodos linha-coluna. Deste modo, Feig e Winograd [27] propuseram um algoritmo de factorização matricial para 2D-IDCT, muito rápido.

De acordo com [27], a matriz DCT pode ser representada pelo seguinte produto matricial $C = D.P.B_1.B_2.M.A_1.A_2.A_3$.

Onde D é a matriz diagonal com a seguinte diagonal {0.3536; 0.2549; 0.2706; 0.3007; 0.3536; 0.4500; 0.6533; 1.2815}. A matriz M é composta por valores reais $\gamma(k)=\cos(\pi k/8)$ e P é a matriz de permutação. Por fim, as restantes matrizes são as seguintes:

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \qquad B_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\gamma_2 & 0 & -\gamma_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\gamma_6 & 0 & \gamma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\gamma_i = \cos\left(\frac{2\pi i}{32}\right)$$

O cálculo da 2D-DCT de 8×8 pontos envolve o produto da matiz $C \otimes C$ dada por:

$$C \otimes C = (D \cdot P \cdot B_1 \cdot B_2 \cdot M \cdot A_1 \cdot A_2 \cdot A_3) \otimes (D \cdot P \cdot B_1 \cdot B_2 \cdot M \cdot A_1 \cdot A_2 \cdot A_3) \qquad \textbf{(6.3.1)}$$

Usando tensores, permite refazer a equação (6.3.1) da seguinte forma:

$$C \otimes C = (D \otimes D)(P \otimes P)(B_1 \otimes B_1)(B_2 \otimes B_2)(M \otimes M) \cdot \\ \cdot (A_1 \otimes B_1)(A_2 \otimes A_2)(A_3 \otimes A_3) \tag{6.3.2}$$

A factorização matricial da 2D-DCT proposta por Feig-Winograd pode ser refeita de forma a processar computacionalmente a 2D-IDCT.

Dado que a matriz C é ortogonal, uma vez que a sua inversa é igual à sua transposta. E uma vez que D é diagonal e M é simétrica obtém-se:

$$C^{-1} = A_3^T \cdot A_2^T \cdot A_1^T \cdot M \cdot B_2^T \cdot B_1^T \cdot C^T \cdot D \tag{6.3.3}$$

Assim a 2D-IDCT pode ser calculada da seguinte forma:

$$(C^{-1} \otimes C^{-1}) \cdot X_{64} = (A_3^T \otimes A_3^T)(A_2^T \otimes A_2^T)(A_1^T \otimes A_1^T)(M \otimes M) \cdot \\ \cdot (B_2^T \otimes B_2^T)(B_1^T \otimes B_1^T)(P^T \otimes P^T)(D \otimes D) \cdot X_{64} \tag{6.3.4}$$

Onde X_{64} é um vector com os 64 elementos da DCT.

E uma vez que P é a matriz de permutação, a equação anterior pode tomar a seguinte forma:

$$(C^{-1} \otimes C^{-1}) \cdot X_{64} = (A_3^T \otimes A_3^T)(A_2^T \otimes A_2^T)(A_1^T \otimes A_1^T)(M \otimes M) \cdot \\ \cdot (B_2^T \otimes B_2^T)(B_1^T P^T \otimes B_1^T P^T)(D \otimes D) \tag{6.3.5}$$

Desta equação obtém-se um algoritmo para a computação da IDCT.

Observa-se que a multiplicação $D \otimes D$ são simples multiplicações ponto-a-ponto, $P^T \otimes P^T$ é a matriz de permutação e as multiplicações de $B_1^T \otimes B_1^T$, $B_2^T \otimes B_2^T$, $A_1^T \otimes A_1^T$, $A_2^T \otimes A_2^T$ e $A_3^T \otimes A_3^T$ por X_{64} envolvem apenas adições, sendo no total 416 adições. No todo, o algoritmo requer um total de 54 multiplicações, 6 operações de *shift* e 46 adições. Uma explicação mais detalhada da IDCT pode ser encontrada em [28] e [27].

Uma implementação eficiente da IDCT requer cálculo de vírgula fixa (*fixed-point*). Isto resulta em menos silício utilizado, e menor consumo energético. No entanto, existe um problema associado aos cálculos de vírgula fixa e devido ao tamanho fixo dos números, isto é, como os elementos da matriz M são números reais, as multiplicações $M \otimes M$ são substituídas por uma sequência de adições e *shifts*. Sendo assim, a matriz M é aproximada da seguinte forma:

$$\begin{aligned}
\gamma_2 &= 1 - 2^{-4} - 2^{-6} + 2^{-9} + 2^{-14} \\
\gamma_4 &= 1 - 2^{-2} - 2^{-5} - 2^{-7} - 2^{-8} + 2^{-14} \\
\gamma_6 &= 2^{-2} + 2^{-3} + 2^{-7} - 2^{-13} \\
\gamma_2 + \gamma_6 &= 1 + 2^{-2} + 2^{-4} - 2^{-8} + 2^{-9} \\
\gamma_2 - \gamma_6 &= 2^{-1} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-13} + 2^{-14} \\
\gamma_4 / 2 &= 2^{-2} + 2^{-3} - 2^{-6} - 2^{-8} - 2^{-9} + 2^{-13} \\
\gamma_4 \gamma_6 &= 2^{-2} + 2^{-6} + 2^{-8} + 2^{-10} + 2^{-14}
\end{aligned} \qquad \textbf{(6.3.6)}$$

O erro obtido com estas aproximações foi ajustado de forma a cumprir a norma IEEE 1180-1990 [29] com as modificações dadas pelo MPEG-4 Annex A de [28]. Assim, a seguinte tabela mostra a precisão da IDCT implementada.

Tabela 12 – Precisão da IDCT implementada.

IEEE 1180-1990 Test			**Results**
Test	**Interval**	**Error type**	
1	[-256, +255] Sign = +1	Peak Error	1
		Worst mse	0.022853
		Overall mse	0.019091
		Worst mean error	0.00053
		Overall mean error	1.08125e-05
2	[-5 ,+5] Sign = +1	Peak Error	1
		Worst mse	0.000535
		Overall mse	0.000421
		Worst mean error	0.00024
		Overall mean error	6.48438e-06
3	[-384, +383] Sign = +1	Peak Error	1
		Worst mse	0.022459
		Overall mse	0.018317
		Worst mean error	0.0004
		Overall mean error	2.4688e-06
4	[-256, +255] Sign = -1	Peak Error	1
		Worst mse	0.022819
		Overall mse	0.019072
		Worst mean error	0.000389
		Overall mean error	4.414e-05
5	[-5 ,+5] Sign = -1	Peak Error	1
		Worst mse	0.000552
		Overall mse	0.000419
		Worst mean error	0.000246
		Overall mean error	1.15e-05
6	[-384, +383] Sign = -1	Peak Error	1
		Worst mse	0.02247
		Overall mse	0.01832
		Worst mean error	0.00031
		Overall mean error	3.84688e-06

6.3.2 Implementação

Para a implementação do algoritmo proposto, foi feito em VHDL um módulo tal como mostra a Figura 54.

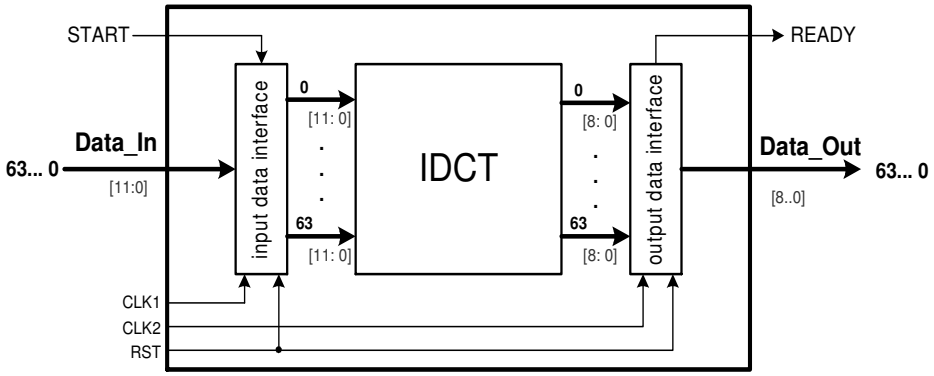

Figura 54 – Diagrama de blocos do módulo IDCT.

Os coeficientes são carregados na entrada Data_In síncronos com o relógio CLK1, assim que o sinal START esteja activo, ou seja, START = 1. O resultado da 2D-IDCT sobre os 64 elementos é obtido na saída Data_Out síncronos com o relógio CLK2, assim que o sinal READY fique activo, ou seja, READY = 1. O sinal RST quando activo, RST = 1, reinicia o módulo IDCT.

A interface de entrada é um conversor série-paralelo, Input Data Interface, que recebe 64 elementos de 12 bits em complemento para 2. Devido à estrutura do algoritmo apresentado, só quando os 64 elementos são carregados é que o cálculo da IDCT é propriamente efectuado.

Para a implementação do algoritmo Feig-Winograd, o sub-módulo IDCT da figura foi dividido em 8 sub-módulos OP0, OP1, OP2, OP3, OP4, OP5, OP6 e OP7, Figura 55.

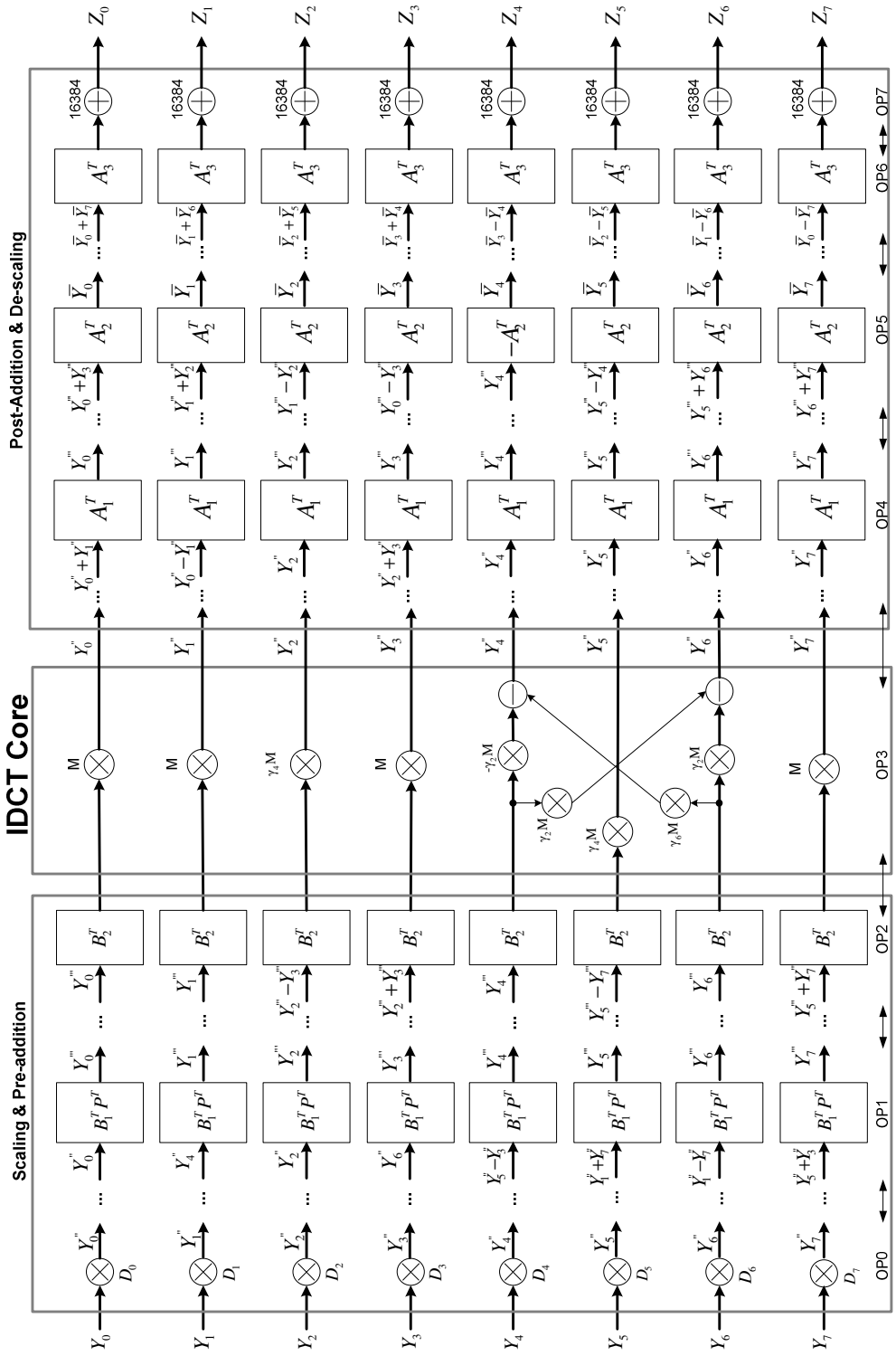

Figura 55 – Diagrama de blocos dos sub-módulos do módulo IDCT.

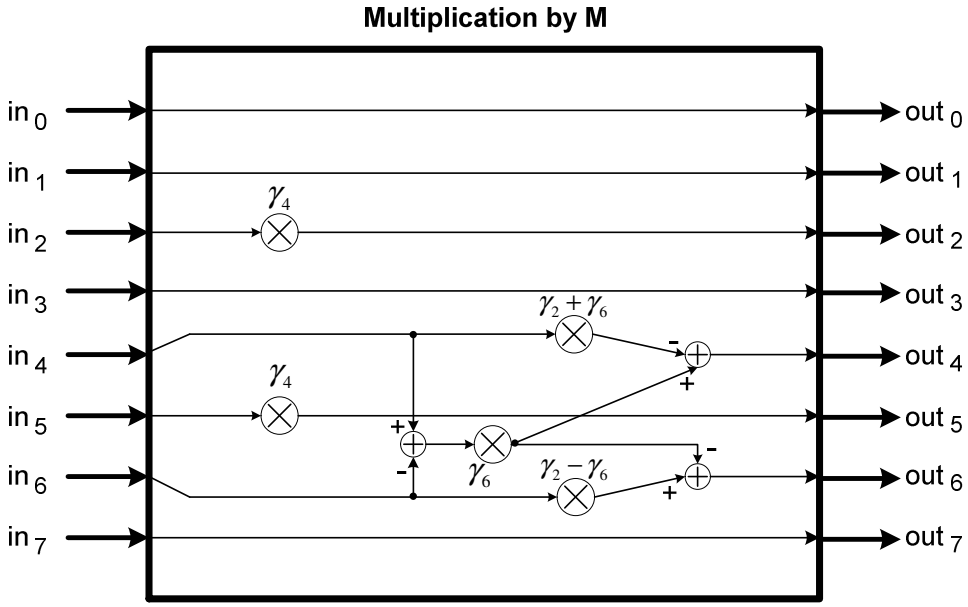

Figura 56 – Esquema da multiplicação M do sub-módulo OP3 do módulo IDCT.

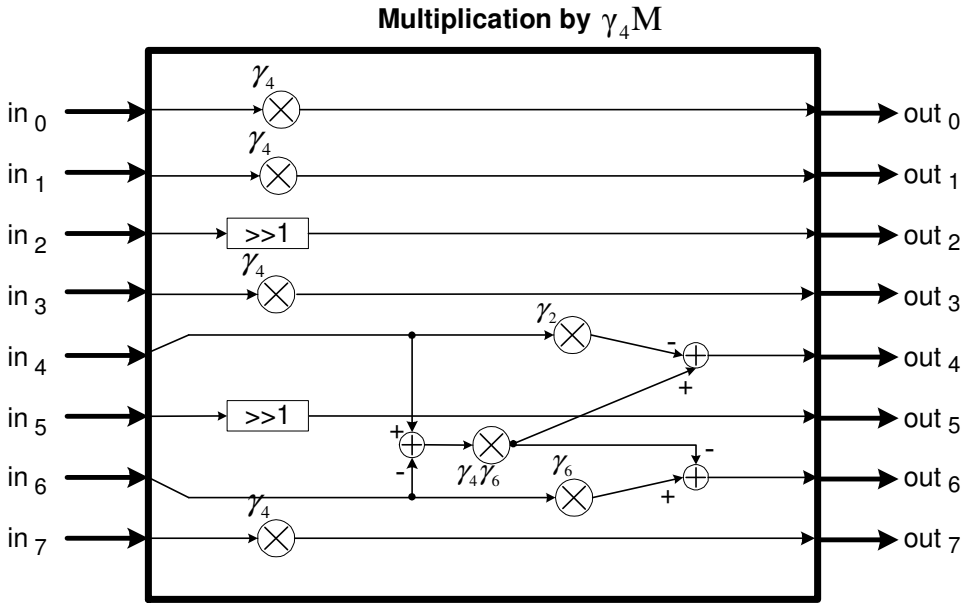

Figura 57 – Esquema da multiplicação γ_4M do sub-módulo OP3 do módulo IDCT.

Na Figura 55, cada Y_n, onde n=0, 1, …, 7, é um vector de 8 elementos de 26 bits em complemento 2.

O sub-módulo OP0 efectua a multiplicação dos coeficientes de entrada pela matriz diagonal D. A síntese VHDL deste sub-módulo resulta só em multiplicadores, MUX18X18 da Virtex-II™, ou seja, existe uma optimização dos recursos da FPGA. De outro modo,

caso as multiplicações fossem substituídas por adições e subtracções a utilização de recursos da FPGA era maior.

Os sub-módulos OP1 e OP2, em conjunto com o OP0, efectuam a fase de cálculo chamada de Scalling & Pre-Addiction, escalamento e pré-adição. O código VHDL destes dois sub-módulos, OP1 e OP2, resulta em lógica puramente combinatória que efectua só adições e subtracções. Cada um destes sub-módulos ocupa 5% das *slices* da FPGA e 5% de 4LUTs devido ao facto de se estar perante uma lógica puramente combinatória e porque os vectores Y_n são vectores com 8 elementos de 26 bits cada um. Isto resulta num espaço ocupado, relativamente grande, da FPGA, só para estes sub-módulos.

O sub-módulo OP3 é um módulo que efectua o efectivamente a IDCT e é um dos módulos que mais recursos ocupa da FPGA, porque efectua multiplicações, além de subtracções e adições. De forma a se optimizarem os recursos da FPGA, poder-se-ia ter utilizado os multiplicadores embebidos da FPGA, MUX18X18, para efectuar as multiplicações. No entanto, e como já estão utilizados 64 MUX18X18 dos 96 disponíveis pela FPGA pelo sub-módulo OP0, restou só a solução de se efectuarem as multiplicações com adições e shifts. Deste modo, os recursos da FPGA foram utilizados em larga medida.

Os sub-módulos OP4, OP5 e OP6 efectuam a penúltima operação a Post-Addiction & De-Scalling. O código VHDL destes sub-módulos resulta também em lógica combinatória pura o que resultou em velocidade em detrimento dos recursos da FPGA.

Por fim, o último sub-módulo OP7 foi incluído já no processo do conversor paralelo-série, obtendo-se assim uma optimização de recursos da FPGA na ordem dos 2%.

O último sub-módulo é o conversor paralelo-série, Output Data Interface, que devolve os coeficientes calculados a cada ciclo de relógio CLK2 sempre que READY = 1.

A seguinte tabela mostra a ocupação de recursos da FPGA dos sub-módulos OP0, OP1, OP2, OP3, OP4, OP5, OP6 e OP7.

Tabela 13 – Ocupação de recursos da FPGA pelos sub-módulos OP0, OP1, …, OP7.

FPGA recursos	OP0	OP1	OP2	OP3	OP4	OP5	OP6	OP7
Slices	0	832	832	3656	624	1536	1536	0
Slices FF	0	0	0	0	0	0	0	0
4LUTs	0	1664	1664	7196	1248	2700	3072	0
MULT18x18s	64	0	0	0	0	0	0	0

A seguinte figura mostra o diagrama temporal do módulo IDCT desenvolvido.

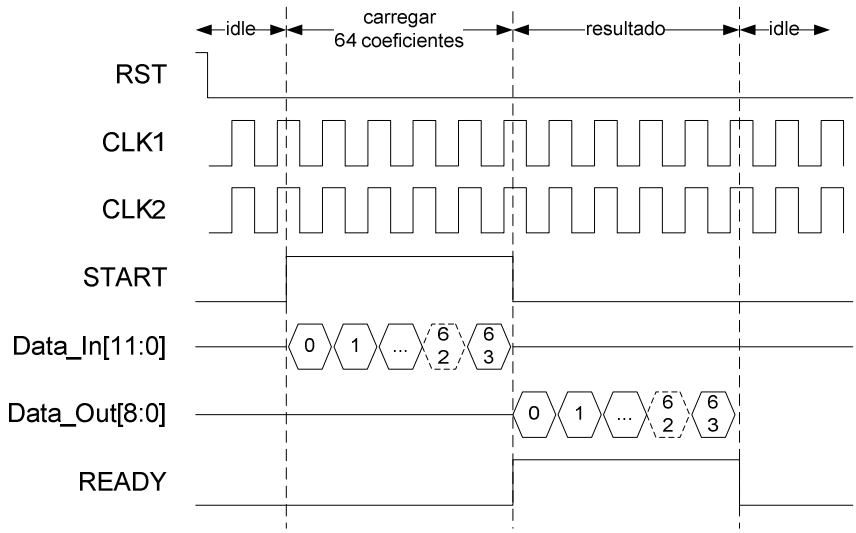

Figura 58 – Diagrama temporal do módulo IDCT.

O módulo deve ser reiniciado, RST = 1, para que o módulo fique num estado conhecido. Assim que o módulo detecta o sinal START = 1, este começa a carregar os 64 elementos no seu conversor série-paralelo em cada flanco ascendente do relógio CLK1. Após 64 ciclos de relógio CLK1, o módulo assinala na sua saída READY = 1, indicando que os coeficientes foram calculados. Os 64 elementos são lidos em cada flanco ascendente de CLK2. Após 64 ciclos de relógio de CLK2, o módulo coloca o sinal READY = 0 e espera por outro START = 1 para reiniciar o processo. O sinal START = 1 deve manter-se durante todo o processo de carregamento dos 64 coeficientes.

6.3.3 Resultados

A próxima tabela mostra os recursos totais utilizados pelo módulo da IDCT desenvolvido na FPGA. De notar que GLCK (Global Clock – relógio global), apresenta uma funcionalidade finita da família Virtex-II™, reservada para um melhor encaminhamento e distribuição dos sinais de relógios.

Tabela 14 – Ocupação de recursos pelo módulo IDCT.

FPGA recursos	Módulo IDCT	
Slices	9421	65%
Slices FF	853	2%
4LUTs	18218	63%
MULT18x18s	64	66%
GLCKs	2	12%

Uma vez que o algoritmo foi desenhado só com lógica combinatória, com excepção dos conversores série-paralelo e paralelo-série, o cálculo da IDCT não tem latência associada, ou seja, é extremamente rápido e está só limitado pela velocidade máxima da lógica do FPGA utilizado.

A velocidade máxima de relógio para este módulo é de 219.25MHz, ou seja, tem um débito aproximadamente de 19.3Mblocos/s.

Comparando o módulo desenvolvido por [30], verifica-se que o módulo desenvolvido é 42 vezes mais rápido pelo facto de não apresentar latência. No entanto, e como foi analisado anteriormente, a percentagem de ocupação é elevada. Conclui-se que o módulo aqui apresentado pode ser uma boa solução para aplicações onde se requer grande velocidade para o cálculo da IDCT, tal como televisão de alta definição, HDTV (*High Definition Television*). Outras implementações, tais como as apresentadas por [31] e [32], reduzem os recursos usados mas, apresentam um elevada latência.

6.4- Módulo VLD+IQ+IDCT

É um módulo desenvolvido em VHDL que consiste nos três módulos MPEG-4 anteriormente descritos, VLD, IQ e IDCT, [33]. Portanto, descreve-se aqui só a implementação deste módulo, uma vez que o funcionamento individual dos sub-módulos VLD, IQ e IDCT já foi documentado.

6.4.1 Implementação

Este módulo consiste na integração dos módulos VLD, IQ e IDCT, anteriormente descritos num único módulo, Figura 59.

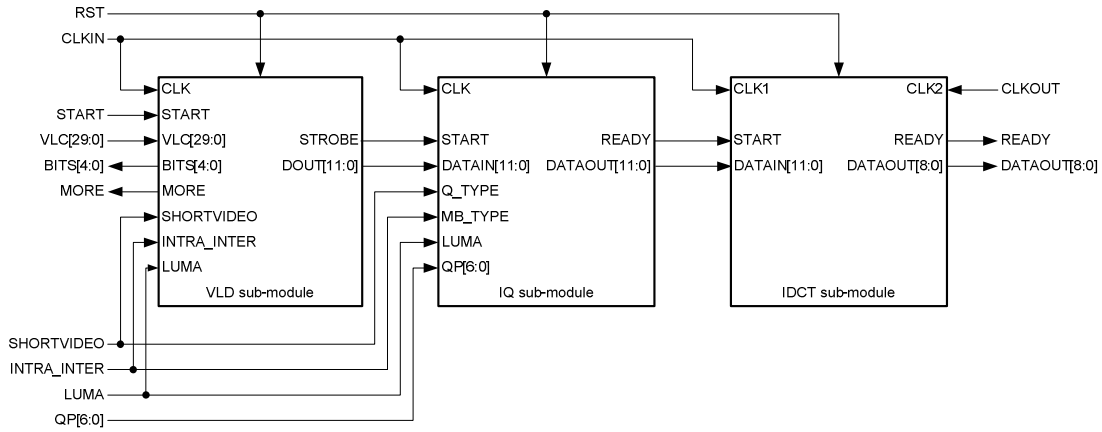

Figura 59 – Diagrama de blocos do módulo VLD+IQ+IDCT.

O funcionamento baseia-se no encadeamento dos módulos VLD, IQ e IDCT de tal modo que a descodificação MPEG-4 é quase completa.

O diagrama temporal deste módulo é mostrado na seguinte figura.

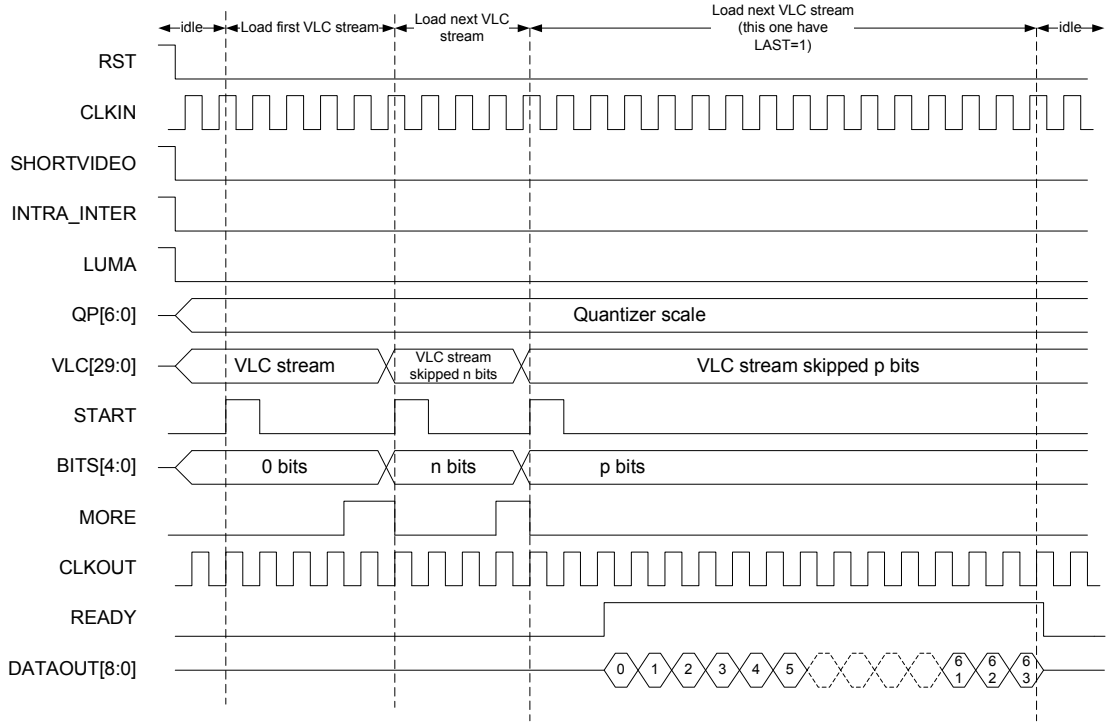

Figura 60 – Diagrama temporal do módulo VLD+IQ+IDCT.

O diagrama mostra um exemplo de três VLC *streams* para serem descodificadas. A primeira, corresponde ao código VLC, referente ao coeficiente DC, e a última contém um código VLC, onde é encontrado LAST = 1 terminando assim o processo de descodificação.

6.4.2 Resultados

A percentagem de ocupação e desempenho da FPGA é o seguinte:

Tabela 15 – Percentagem de ocupação do módulo VLD+IQ+IDCT.

4LUTs	26569
MUX18X18	7
BRAM	1
Freq. de relógio máxima	65MHz

A ocupação de recurso da FPGA é relativamente alta e o desempenho moderado. O sub-módulo IDCT é o responsável pela quantidade de 4LUTs usadas, tal como foi analisado anteriormente. O agravamento da frequência máxima de relógio deve-se ao sub-módulo IQ que é quase só lógica combinatória.

6.5- Módulo *Deblocking*

Foi desenvolvido um módulo em VHDL para efectuar o processo de *Deblocking* (efeito de "mosaico") [34] presente em codificações de vídeo baseadas em IDCT e compensações de movimento por blocos tal como na norma MPEG-4 AVC/H.264.

A norma de codificação de vídeo MPEG-4 AVC/H.264, como é baseado em IDCT inteira, produz um efeito de "bloco" (mosaico) indesejado na imagem, [35]. Também, a previsão da compensação de movimento é uma das fontes deste efeito indesejado. E, apesar de no MPEG-4 AVC/H.264 a IDCT apresentar um tamanho reduzido de 4×4 elementos que reduz de certa forma este efeito, o uso de um filtro de remoção de artefactos (*deblocking filter*) pode maximizar o desempenho da codificação.

Como o filtro de remoção de artefactos no MPEG-4 AVC/H.264 está na malha (*loop*) de codificação, é imperativo que todos os blocos 4×4 sejam filtrados por este filtro. Isto requer um cálculo computacional intensivo, devido ao filtro ser adaptativo. Desta forma, o módulo desenvolvido tenta remover algum do tempo que seria necessário para o cálculo do filtro, libertando o processador principal para outras tarefas.

Este módulo recebe todos os pixels não filtrados de um bloco e aplica o filtro de remoção de artefactos definido na norma MPEG-4 AVC/H.264 [2].

6.5.1 Algoritmo

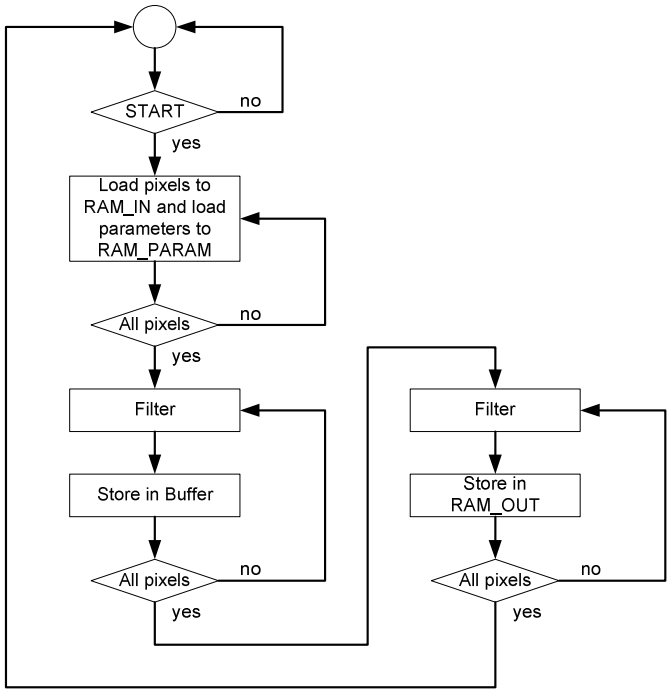

Figura 61 – Algoritmo do módulo *Deblocking*.

Quando é detectado o início do processo, START = 1, o módulo armazena internamente os pixels a serem filtrados numa memória RAM, RAM_IN, estrategicamente organizada. A organização da RAM_IN consiste em armazenar em cada posição da memória 8 pixels ordenados por p3, p2, p1, p0, q0, q1, q2 e q3 (ou 4 se for um bloco de crominância, p1, p0, q1 e q0) correspondentes às fronteiras horizontais, tal como é definido em [2]. Paralelamente, também é armazenada informação referente às condições do filtro, na memória RAM_PARAM.

Quando todos os pixels estão carregados, o processo de filtragem inicia-se e o resultado da filtragem das fronteiras horizontais é armazenado num *buffer*. Este *buffer* é um *buffer* especial, uma vez que quando se passa à fase de filtragem das fronteiras verticais, os pixels já estão organizados, de tal forma que não é necessária nenhuma reorganização. Assim, poupam-se ciclos de relógio necessários para ler os pixels com a filtragem horizontal, e depois reorganizá-los para a filtragem vertical, i.e., aumenta-se o desempenho do módulo.

Assim que a filtragem vertical termine, o módulo assinala que a mesma terminou.

6.5.2 Implementação

O diagrama de blocos do módulo *Deblocking* está representado na Figura 62.

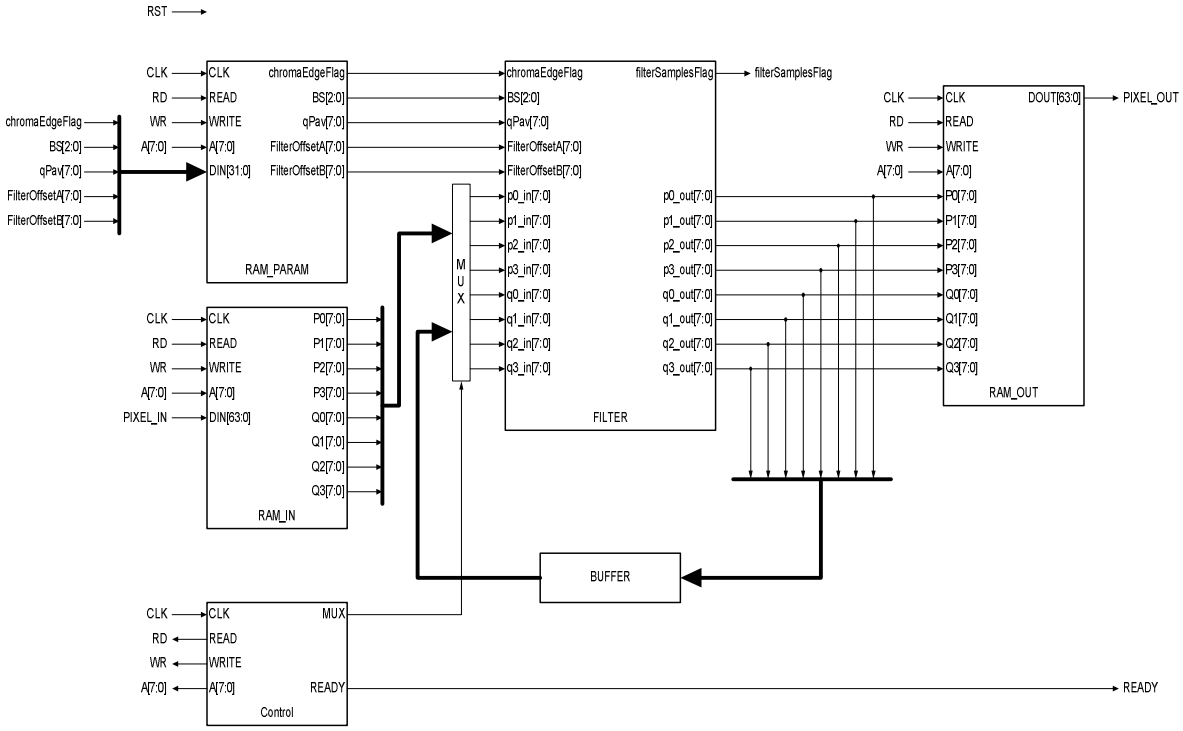

Figura 62 – Diagrama de blocos do módulo *Deblocking*.

O processo de descodificação é iniciado quando START = 1. São então armazenados os pixels dos blocos, PIXELS_IN, e as condições de filtragem, chromaEdgeFlag, BS[2:0], qPav[7:0], FilterOffsetA[7:0] e FilterOffsetB[7:0] para o Deblocking. Os pixels são armazenados em memórias RAM de forma estratégica tal como em [36]. Assim, e porque o filtro propriamente dito é lógica combinatória pura, só são gastos ciclos de relógio para guardar e ler das RAMs, que influenciam a latência do módulo.

Primeiro, as fronteiras horizontais dos blocos são filtradas tal como é recomendado por [2] e o resultado é armazenado num *buffer* (zona temporária de memória) especial, tal como descrito anteriormente. Por fim, é efectuada a filtragem vertical das fronteiras dos blocos, é armazenado o resultado numa RAM, RAM_OUT, e é assinalado o fim do processo, READY = 1.

O processo de filtragem é controlado por FSM e algumas LUTs são usadas para reduzir o número de cálculos.

O sinal de entrada PIXEL_IN é um porto de 64 bits dividido em duas secções de 32 bits: PIXEL_IN_P e PIXEL_IN_Q. A secção PIXEL_IN_P é composta por 4 pixels de 8 bits cada um, referentes aos pixels p3, p2, p1 e p0, tal como definidos em [2]. A outra secção, tem uma organização semelhante à de PIXEL_IN_P, mas agora refere-se aos pixels q3, q2, q1 e q0 definidos em [2]. O sinal de saída PIXEL_OUT é semelhante ao sinal PIXEL_IN só que este contém os pixels totalmente filtrados.

O seguinte diagrama mostra as relações temporais dos sinais do módulo desenvolvido.

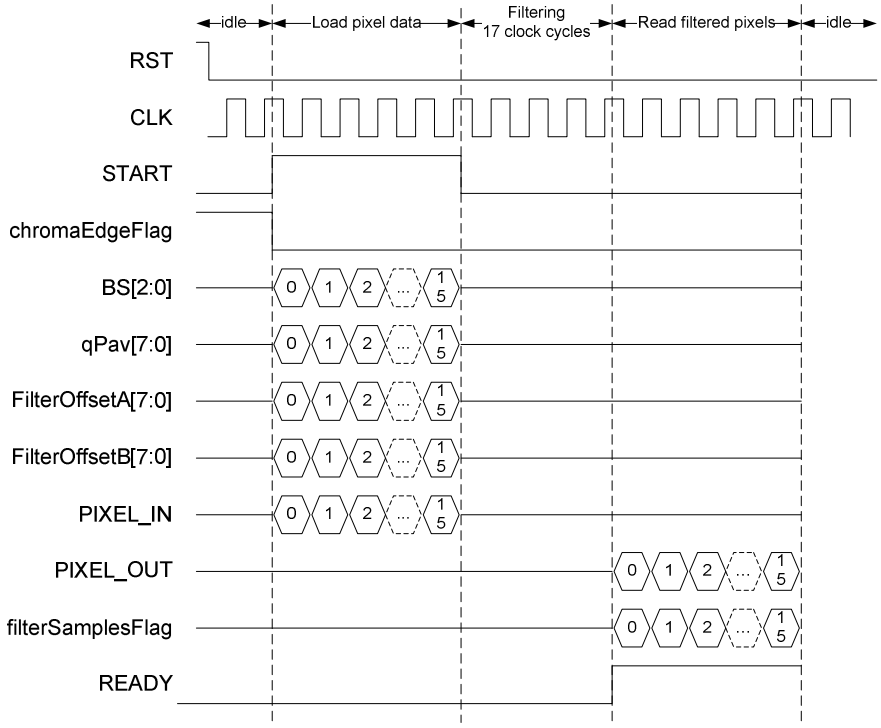

Figura 63 – Diagrama temporal do módulo Deblocking.

O diagrama mostra o exemplo da remoção de artefactos (*deblocking*) para pixels de crominância (chromaEdgeFlag = 0). Nos primeiros 16 ciclos de relógio e quando START = 1, os pixels são armazenados na memória RAM_IN interna, iniciando-se assim o processo de filtragem. Ao fim de mais 17 ciclos de relógio (a latência do módulo), o módulo assinala que o processo de filtragem está finalizado, READY = 1, e que os dados em PIXEL_OUT estão válidos.

6.5.3 Resultados

A seguinte tabela mostra a percentagem de ocupação da FPGA, bem como o seu desempenho.

Tabela 16 – Percentagem de ocupação da FPGA do módulo *Deblocking*.

4LUTs	1456
BRAMs	3
MUX18X18	0
Freq. de relógio máxima	40MHz

A limitação da velocidade de operação deve-se ao bloco FILTER que é composto unicamente por lógica combinatória. Não obstante, este bloco efectua a filtragem quase instantaneamente, estando limitado unicamente à velocidade de propagação do caminho mais longo da lógica implementada.

Apesar de a frequência máxima de funcionamento ser de 40MHz, os recursos utilizados são inferiores aos utilizados em [37]. Enquanto que em [37] são utilizados 3909 geradores de funções (4LUTs), no módulo desenvolvido são utilizados 1456 4LUTs, ou seja, existe um ganho de 2.6 vezes, em recursos.

Uma vez que o filtro é efectuado num só ciclo de relógio, a velocidade de cálculo é mais rápida que em [37]. Mesmo com uma frequência de funcionamento máxima mais baixa, 40MHz em relação a 72MHz de [37], o módulo aqui apresentado faz a remoção de artefactos de um bloco 4×4 muito mais rapidamente. Bastam 17 ciclos de relógio, mais 16 para carregar os pixels e mais 16 ciclos para ler o resultado, para processar completamente um bloco 4×4.

6.6- Módulo Estimação de Movimento

Na norma MPEG-4 AVC/H.264 a estimação de movimento apresenta novas características tais como: blocos de tamanho variável, imagens de referência múltiplas, vectores de predição de movimento, resolução de *quarter-pixel*, etc. Estas novas características requerem mais poder computacional e complexidade, relativamente à norma MPEG-2. Resultados práticos demonstram que a estimação de movimento em MPEG-4 AVC/H.264 consome cerca de 60% do tempo para a codificação com uma imagem de

referência, e até 80% para várias imagens de referência [38]. Deste modo, o uso de um módulo em hardware dedicado pode acelerar significativamente o processo de estimação, fazendo uso de processamento paralelo.

A estimação de movimento explora a redundância temporal existente numa sequência de vídeo. Entre vários algoritmos de estimação de movimento, o algoritmo *full-search block-matching* é o que produz melhores resultados [38] na procura dos vectores de movimento (MV), Figura 64.

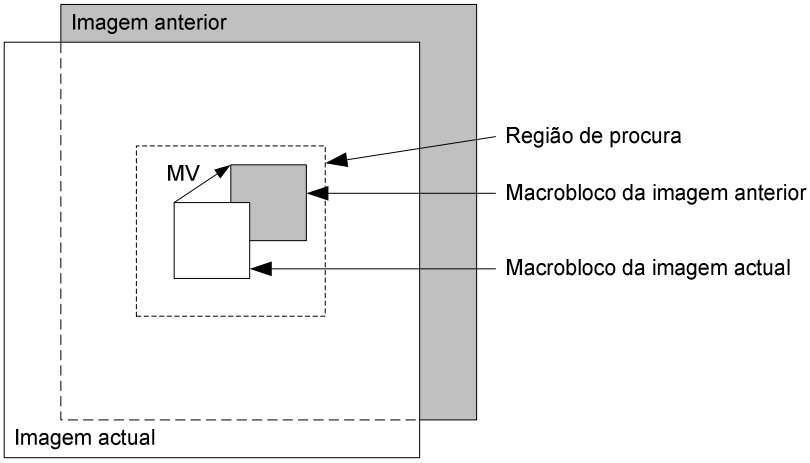

Figura 64 – Processo de procura *block-matching*.

Tal como mostra a figura anterior, a estimação de movimento é efectuada em dois passos. Primeiro, é calculado a soma de diferenças absolutas, SAD (*Sum of Absolute Differences*), de cada vector de deslocamento. Seguidamente, usam-se métodos para encontrar a menor das SADs anteriormente calculadas, SAD_{MIN}.

$$SAD_{(i,j)} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\left|C(k,l) - R(i+k, j+l)\right| \quad \textbf{(6.6.1)}$$

$$SAD_{MIN} = \min(SAD(i, j)) \quad \textbf{(6.6.2)}$$

Na equação (6.6.1) *C(k,l)* representa o macrobloco actual e *R(i+k,j+l)* e macrobloco de referência.

As melhorias na estimação de movimento em MPEG-4 AVC/H.264 devem-se ao uso de blocos de tamanho variável para efectuar a estimação, ao contrário das normas MPEG anteriores. O macroblocos, na norma MPEG-4 AVC/H.264, são segmentados em dois modos: modo macrobloco, Figura 65-a), e modo 8×8, Figura 65-b) [38].

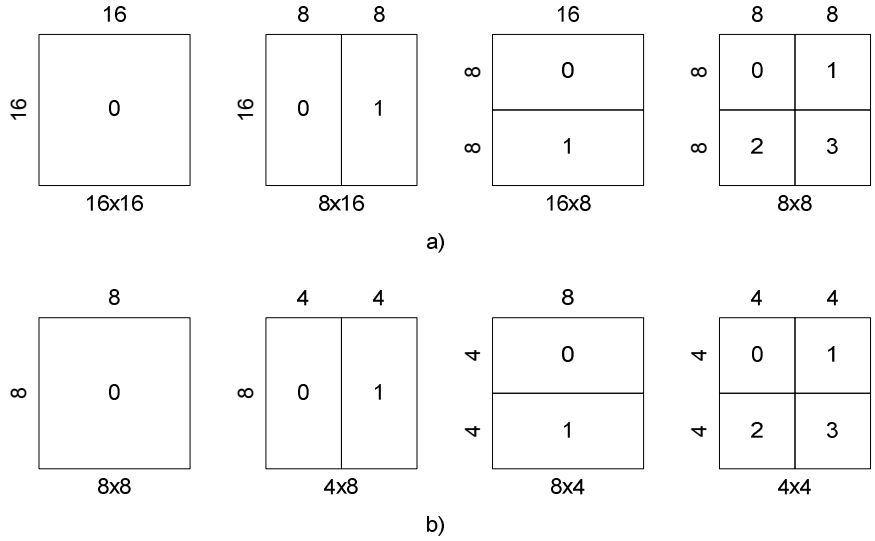

Figura 65 – Modo a) macrobloco e b) 8×8.

Esta segmentação resulta num total de 41 sub-blocos, i.e., cada macrobloco contém um total de 41 vectores de movimento. Deste modo, o módulo aqui apresentado, Estimação de Movimento, permite calcular os 41 vectores de movimento associados a um macrobloco da norma MPEG-4 AVC/H.264, usando o algoritmo *full-search block-matching* com terminação antecipada. Apesar de este algoritmo ser mais lento do que o método usado em [39], ele garante que é encontrada a região que melhor coincide com a imagem de referência, traduzindo-se numa melhoria dos valores da PSNR, e mais compressão. No entanto, o uso da terminação antecipada reduz o tempo necessário para o processo de procura, isto é, se durante o processo de procura, o valor actual da SAD for superior ao valor anteriormente obtido, então o módulo não necessita de efectuar todas as pesquisas necessárias para o cálculo da actual SAD, prosseguindo imediatamente para a próxima SAD.

6.6.1 Algoritmo

O módulo Estimação de Movimento aqui apresentado usa o algoritmo *full-search block-matching* com terminação antecipada [40]. Assim, se durante o processo de procura, determinação da área de *best matching*, o valor actual da SAD for superior ao calculado até ao momento, então é terminada a busca nessa área, e passa-se à próxima. Desta forma, o desempenho do módulo é melhorado, reduzindo assim o número de ciclos de relógio para efectuar a procura.

O tamanho da janela de procura (*search window*) é 48×48 pixels, o que corresponde a três vezes o tamanho do macrobloco 16×16, e por sua vez, cada macrobloco 16×16 é sub dividido em sub-blocos 4×4, Figura 66.

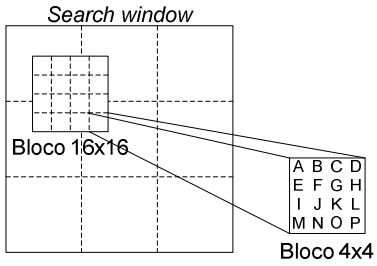

Figura 66 – Janela de procura (*search window*).

A cada bloco 4×4 os pixels são etiquetados por ordem alfabética, tal como mostra a figura. E a *search window* contém os pixels da imagem de referência, onde deve ser encontrado o bloco 16×16 que melhor coincide com o bloco 16×16 actual.

A figura seguinte ilustra o algoritmo desenvolvido para detectar a zona que melhor coincide (*best match*) com a imagem de referência.

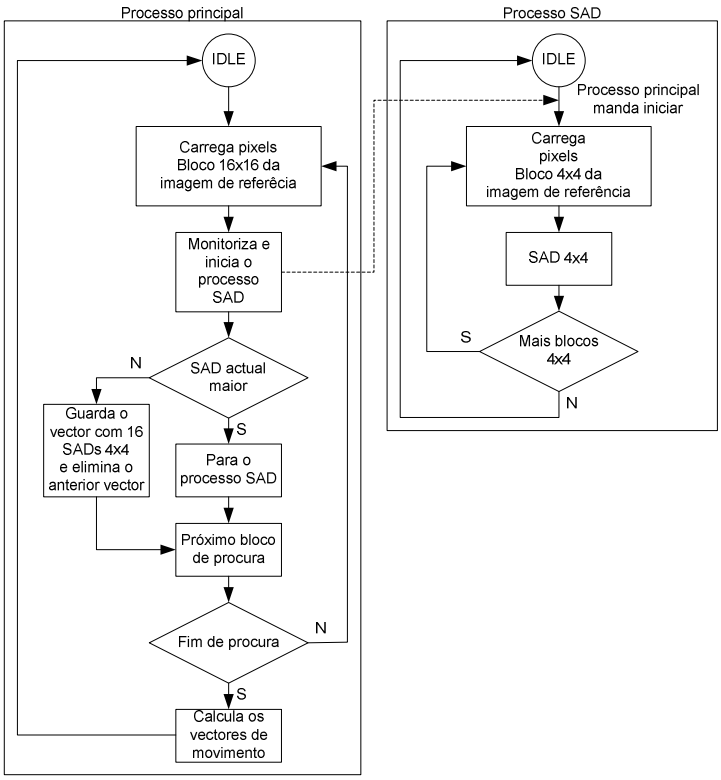

Figura 67 – Algoritmo do módulo Estimação de Movimento.

Existem dois processos a serem executados em paralelo, o processo principal e o processo SAD. O primeiro começa por carregar um bloco de 16×16 pixels da janela de procura, *search window*, e inicia o processo SAD. Por sua vez, o processo SAD carrega um dos 16 blocos de 4×4 pixels do bloco 16×16 de referência e calcula num só ciclo de relógio o valor da SAD com tamanho 4×4 efectuando posteriormente o cálculo dos restantes 15 blocos 4×4. Entretanto, o processo principal armazena num vector de 16 posições as sucessivas SAD 4×4 calculadas. Note-se, que o bloco 16×16, correspondente à imagem corrente, foi previamente armazenado.

Internamente, o processo SAD vai sucessivamente calculando a soma de todas as 16 SADs 4×4 de um bloco 16×16 pixels. Esse valor é constantemente passado para o processo principal, de modo a ser usado na terminação antecipada do processo de busca. Sempre que este valor for superior ao valor anteriormente registado, o processo SAD é terminado e o processo principal carrega os pixels do próximo bloco de procura.

O processo principal contém dois vectores com 16 posições cada um. O primeiro, que guarda as 16 SADs 4×4 actuais e o segundo, as 16 SADs 4×4, anteriormente calculadas. Se a SAD actual é numericamente inferior à SAD anterior, então o vector que

contém a SAD actual toma os valores do vector da SAD anterior e vice-versa. Desta forma, controla-se qual o vector que possui os valores da SAD válidos.

Assim que o processo SAD calcule todas as 16 SADs de um bloco 16×16 da imagem de referência, o processo principal verifica se existem mais blocos 16×16 na janela de procura. Caso ainda existam blocos, o processo principal carrega o próximo bloco 16×16, caso contrário, o processo principal procede ao cálculo dos vectores de movimento.

Uma vez que o vector final contém as menores SADs 4×4 encontradas, i.e., os valores das SADs onde foi encontrado o *best match*, o processo principal procede ao cálculo das restantes SADs 4×8, 8×4, 8×8, 16×8, 8×16 e 16×16, Figura 68.

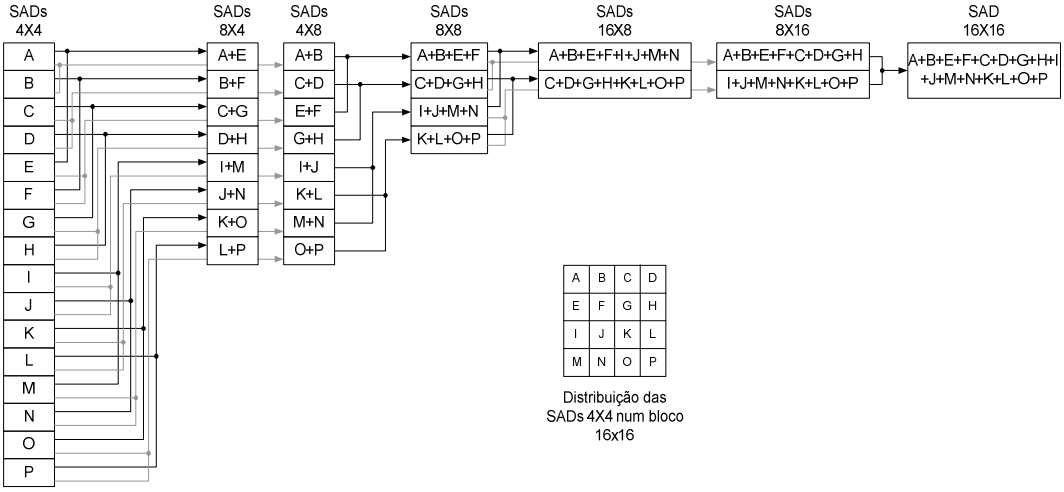

Figura 68 – SADs 4×4, 8×4, 8×8, 16×8, 8×16 e 16×16.

O cálculo destas SADs é efectuado num só ciclo de relógio.

6.6.2 Implementação

O módulo é composto por vários sub-módulos onde cada um é responsável por uma determinada função, Figura 69.

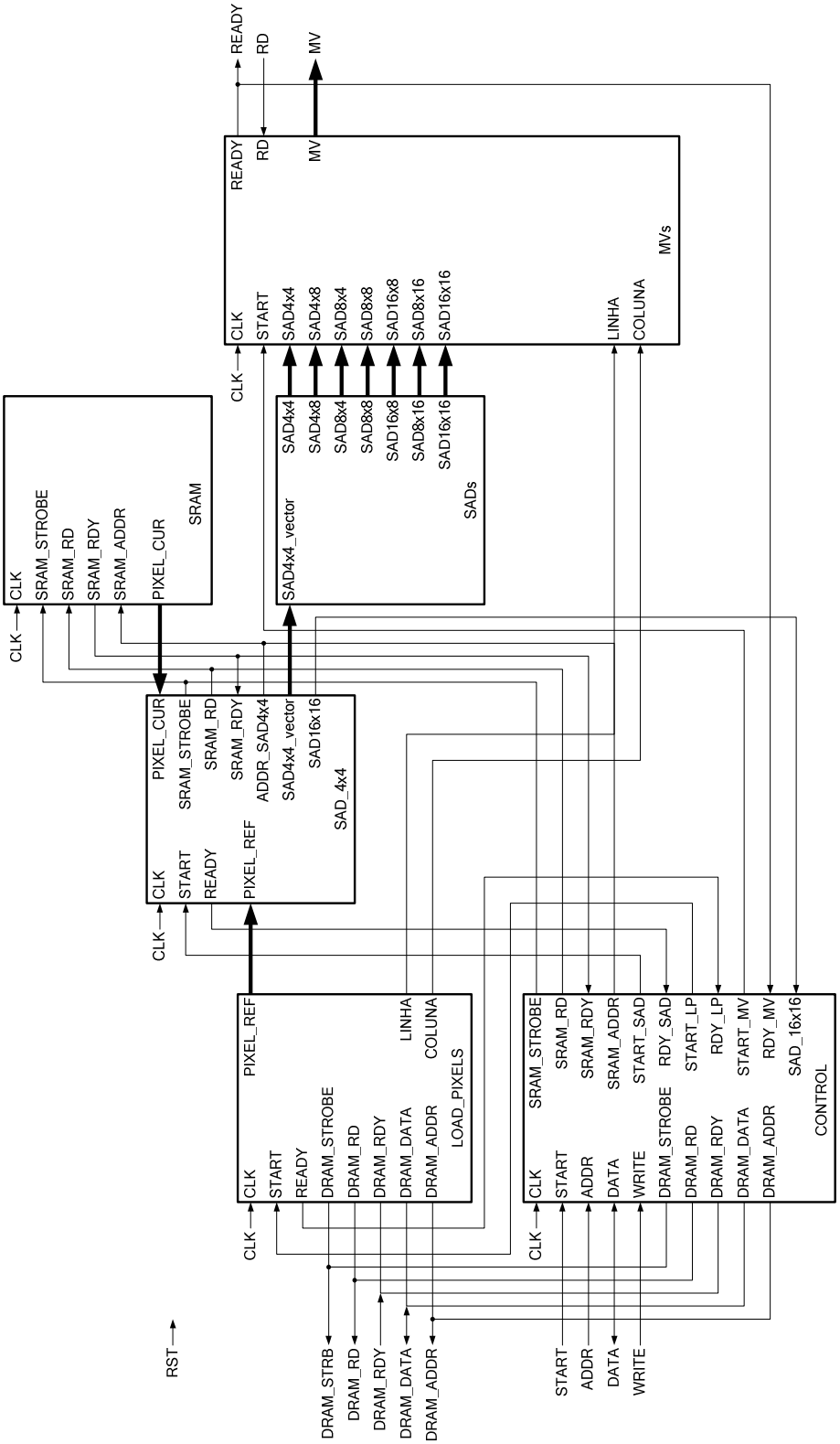

Figura 69 – Diagrama de blocos do módulo Estimação de Movimento.

O processo de estimação de movimento é iniciado quando o sinal START no sub-módulo CONTROL é activado, START = 1. O software deve então começar por fornecer a

informação dos pixels da janela de procura e do bloco de corrente (actual). Os pixels da janela de procura são armazenados na memória DDR-SDRAM da WCII e os pixels do bloco actual são armazenados numa memória estática dentro da FPGA (BRAM). Após todos os pixels terem sido armazenados, inicia o processo de estimação de movimento.

Primeiro, o sub-módulo CONTROL coloca o sinal START_LP no nível lógico alto (START_LP = 1), para que o sub-módulo LOAD_PIXELS proceda à busca dos pixels correspondentes ao bloco 16×16, pretendido na janela de procura (bloco 48×48 pixels). Este sub-módulo LOAD_PIXELS contém toda a lógica necessária para endereçar a memória DDR-SDRAM, através do sinal DRAM_ADDR e por um barramento de dados com 64 bits denominado DRAM_DATA. O controlo da memória DDR-SDRAM é deixado a cargo do controlador embebido na WCII. O armazenamento dos pixels é considerado terminado, assim que o sinal DRAM_RDY esteja no nível lógico alto (DRAM_RDY = 1). Nesse momento, a informação nos pinos PIXEL_CUR é válida. Consequentemente, o sub-módulo CONTROL vai detectar no seu pino RDY_LP = 1, o que faz com que se dê início ao processo do cálculo da SAD, START_SAD = 1.

O sub-módulo CONTROL, a partir de agora, monitoriza o seu sinal SAD_16x16 que indica como está a decorrer o *best match* actual. Este é o sinal usado para que a procura seja interrompida, caso o valor SAD_16x16 seja superior ao valor anteriormente obtido, i.e., a terminação antecipada. Caso tal se verifique, o sinal START_SAD é colocado ao nível lógico baixo (START_SAD = 0) e o sinal START_LP passa ao nível lógico alto (START_LP = 1). Evita-se assim, uma procura desnecessária, o que se traduz numa procura mais rápida.

Após se ter obtido o *best match*, o sub-módulo CONTROL inicia o sub-módulo MVs que calcula os vectores de movimento, colocando sinal START_MV no nível alto (START_MV = 1). O sub-módulo MVs, com base no resultado obtido pelo sub-módulo SADs e os valores dos sinais LINHA e COLUNA, calcula os vectores de movimento associados à deslocação do macrobloco 16×16 actual, em relação à janela de procura.

Quando o sinal READY do sub-módulo MVs é válido (READY = 1), o processo de estimação de movimento está finalizado. Para se obterem vectores de movimento resultantes, o pino RD deve ser colocado no nível lógico alto (RD = 1). Os 41 vectores de movimento são lidos no pinos MV deste sub-módulo.

De notar que, o sub-módulo SRAM é composto por 4 memórias estáticas (16×32) com 16 posições de memória com 32 bits cada uma. Cada uma destas memórias guarda os pixels do bloco 16×16 actual.

Estas memórias estão organizadas de tal forma que o cálculo de uma SAD 4×4 é efectuado num só ciclo de relógio, Figura 70.

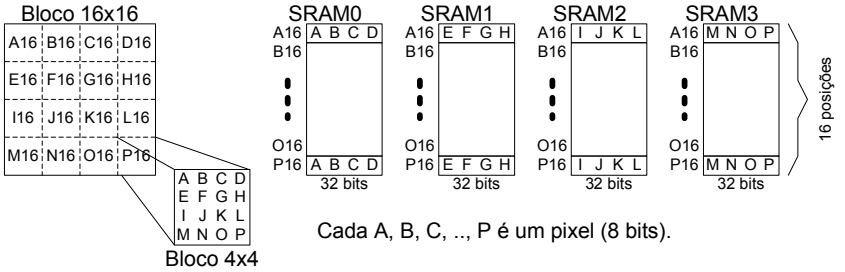

Figura 70 – Organização das memórias SRAM no módulo Estimação de Movimento.

De igual forma, o sub-módulo LOAD_PIXELS recolhe os pixels da memória DDR-SDRAM de tal modo que, a informação nos pinos PIXEL_REF do sub-módulo SAD contém a mesma estrutura e ordem existente nos pinos PIXEL_CUR.

Durante o processo de recolha de pixels na DDR-SDRAM, o sub-módulo LOAD_PIXELS vai guardando nos seus pinos COLUNA e LINHA, a informação da localização do actual teste de *best match* que será usada para o cálculo dos vectores de movimento.

Seguidamente, apresenta-se o diagrama temporal do módulo Estimação de Movimento.

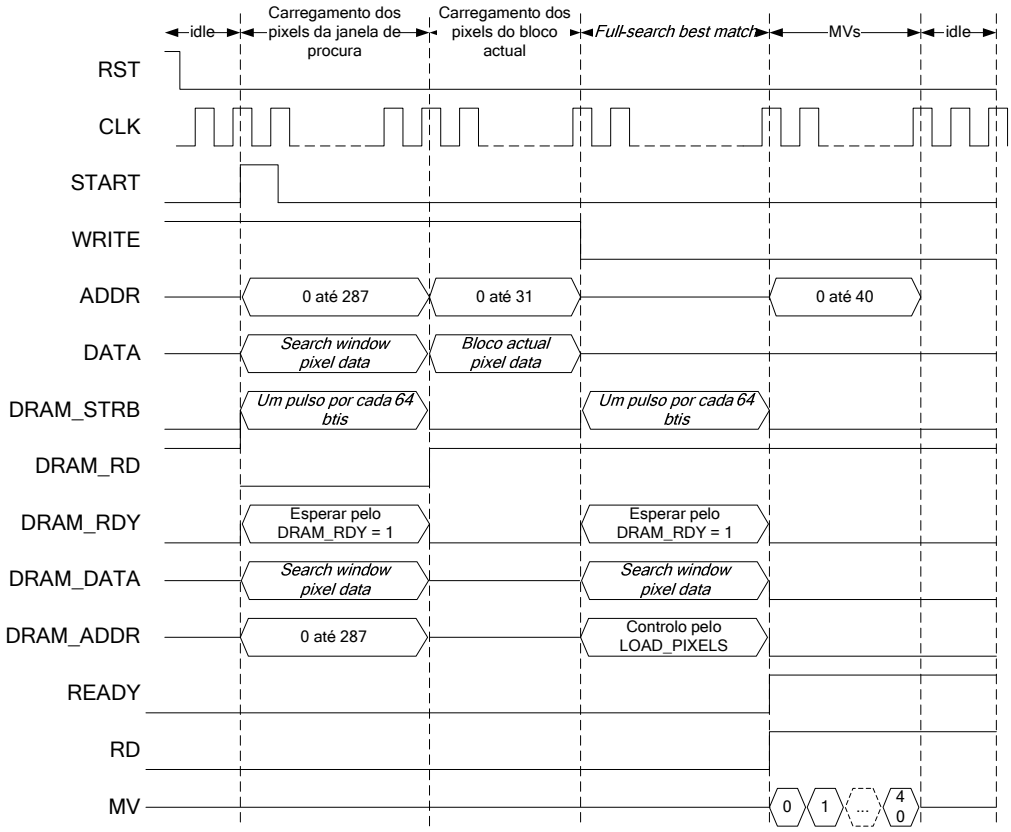

Figura 71 – Diagrama temporal do módulo Estimação de Movimento.

Para efectuar o algoritmo *full-search best match* são necessárias 2048 procuras numa janela de procura 48×48 pixels para um bloco de 16×16 pixels. Se, para cada procura, excluindo a latência da memória DDR-SDRAM, são gastos 8 ciclos de relógio, no máximo, então são necessários 2048×8=16364 ciclos de relógio para a procura total. Todavia, uma vez que é empregue o processo de terminação antecipada (*early termination*), o número de ciclos de relógio será muito menor. Assim, como a procura é iniciada, começando no centro da janela de procura para as periferias (em espiral), então é muito provável que o número de casos de terminação antecipada seja elevado, ou seja, reduz-se o número de ciclos de relógio necessários para a procura.

6.6.3 Resultados

A seguinte tabela mostra a percentagem de ocupação da FPGA e o seu desempenho.

Tabela 17 – Percentagem de ocupação do módulo Estimação de Movimento.

4LUTs	3590
BRAMs	4
MUX18X18	0
Freq. de relógio máxima	50MHz

A limitação da velocidade está principalmente imposta pelo sub-módulo SADs, uma vez que este é exclusivamente composto por lógica combinatória. No entanto, desta forma não se perdem ciclos de relógio para o cálculo intermédio das SADs.

A velocidade máxima de funcionamento do módulo desenvolvido é de 50MHz, ou seja, é mais rápida do que a velocidade apresentada em [39]. Além disso, este módulo utiliza o algoritmo de procura total, o que garante uma SAD mínima, enquanto que em [39] não são verificados todos os pixels do bloco.

O módulo aqui desenvolvido pode ser usado como um co-processador, num sistema de codificação. Deste modo, o processador principal do codificador não precisa de ser tão poderoso, uma vez que o processo de estimação de movimento pode ocupar até 80% do processo de codificação, ficando esta tarefa relegada para o módulo desenvolvido.

Capítulo 7. CONCLUSÕES

A codificação de vídeo, ou seja, a compressão de vídeo encontra inúmeras aplicações que vão desde aplicações de consumo a aplicações de telecomunicações, passando pelas aplicações de estúdio.

A codificação MPEG-4 AVC/H.264 representa uma melhoria significativa na codificação de vídeo. A eficácia conseguida é feita à custa de um aumento significativo da complexidade em relação a codificadores MPEG anteriores. No entanto, a qualidade de imagem obtida e a taxa de compressão são comparativamente melhores.

O uso de mais do que uma imagem de referência para a predição de movimento, faz com que sejam necessários maiores recursos de memória do sistema, bem como uma gestão eficiente dessa memória. Porém, o uso de códigos Exp-Golomb pode gerar facilmente códigos VLCs sem haver a necessidade de armazenar uma tabela de VLCs como no MPEG-4 Visual. O uso de códigos CABAC também apresenta essa vantagem.

Outra vantagem do MPEG-4 AVC/H.264 é a sua transformada ser de fácil implementação, uma vez que bastam operações de adições, subtracções e *shifts* para o seu cálculo. Além disso, para a transformada usada consegue-se implementar a transformada inversa exacta. Todavia, no MPEG-4 Visual tal não é possível.

O uso do filtro de redução de artefactos, *deblocking filter*, faz com que os problemas associados aos codificadores baseados em blocos, MBs, sejam minorados.

O módulo VLD desenvolvido apresenta um desempenho aceitável. Uma forma de aumentar o seu desempenho seria, em vez de procurar um símbolo VLC, de cada vez na *stream* de 30 bits, procurar paralelamente os códigos de vários comprimentos que são previstos. Desta forma, o desempenho seria muito mais elevado. Pelo facto de só serem examinados 30 bits de cada vez, o mesmo implica que se gaste algum tempo entre cada

passagem de 30 bits. O ideal seria se o módulo recebesse toda a *stream* de uma só vez, ou então até encher uma das memórias ZBT-SRAM ou DDR-SDRAM da WCII, e aí proceder à procura dos códigos VLC. Tal procedimento não foi efectuado devido a problemas com a integração com o software de referência. No entanto, os recursos ocupados por este módulo são baixos, o que o torna interessante para implementar em FPGA, ou sistemas de menor capacidade.

O módulo IQ não apresenta muita mais margem para melhorar. A velocidade máxima de funcionamento está limitada pelo caminho mais longo da lógica combinatória implementada. O seu desempenho está a par de outros módulos idênticos desenvolvidos mas com menos ocupação de recursos.

O módulo IDCT apesar de calcular a IDCT de uma forma muito rápida, ocupa muito os recursos da FPGA, ou seja, mais de 60%. Isso deve-se ao facto de serem calculados todos os coeficientes simultaneamente. Dada a existência de oito vectores Y_n, onde cada um é composto por oito coeficientes de 26 bits, origina um total de 1664 bits o que corresponde 55,46% das 3000 gates do FPGA. Uma forma de reduzir os recursos, seria calcular um Y_n de cada vez, mas assim perder-se-ia o principal objectivo deste módulo, que é calcular uma IDCT rapidamente. No entanto, o módulo desenvolvido apresenta um poder de cálculo muito elevado. O cálculo de uma IDCT MPEG-4 num só ciclo de relógio acelera significativamente o processo de codificação.

O módulo *Deblocking* apresenta um filtro rápido, para a remoção de artefactos (*deblocking filter*) presente na norma MPEG-4 AVC/H.264. A forma como os pixels são internamente organizados representa em grande medida o desempenho deste módulo. Para melhorar o desempenho poder-se-ia carregar os pixels e fazer a sua filtragem paralelamente com o desfasamento de um ciclo de relógio, reduzindo assim a latência do módulo.

O módulo Estimação de Movimento, desenvolvido usa o método exaustivo de procura do melhor bloco que coincide com o bloco de referência. Deste modo, conseguem-se melhorias na qualidade de imagem, e consequentemente maior compressão, uma vez que se garante o mínimo de energia residual da imagem. Com o uso da terminação antecipada, *early termination*, a velocidade do cálculo dos vectores de movimento é melhorada.

Neste trabalho, apresenta-se implementações eficientes de diversos módulos em FPGA. Estes podem ser utilizados na codificação de vídeo, encontrando aplicações onde existam poucos recursos devido à sua baixa percentagem de ocupação da FPGA: módulo VLD e módulo IQ, e também em aplicações onde se requer maior velocidade usando por exemplo o módulo IDCT ou o módulo *Deblocking*. Para além disso, o módulo Estimação de Movimento pode ser usado como um co-processador de baixo custo para a estimação do movimento.

REFERÊNCIAS

[1] ISO/IEC 14486. Generic coding of audio-visual objects – Part2: Visual.

[2] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC), Mar. 2003.

[3] Charles Poynton, "Digital Video and HDTV", Morgan Kaufmann Publishers, 2003, ISBN 1-55860-792-7.

[4] Iain E. G. Richardson, "H.264 and MPEG-4 Video Compression", John Wiley & Sons Inc., 2003, ISBN 0-470-84837-5.

[5] Recommendation ITU-R BT.601-5, "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratio", ITU-T, 1995.

[6] Recommendation ITU-T BT.500-11, "Methodology for the subjective assessment of the quality of television pictures", ITU-T, 2002.

[7] Keith Jack, "Video Demystified: A Handbook for the Digital Engineer", Third Editon, LLH-Technology Publishing, 2001.

[8] ISO/IEC 14495-1:2000 Information Technology – lossless and near-lossless compression of continuous-tone still images: Baseline, (JPEG-LS).

[9] John Proakis, Dimitris G. Manolakis, "Digital Signal Processing", Third Edition, Prentice-Hall International, Inc., 1996, ISBN 0-13-394338-9.

[10] K. R. Rao, P.C. YIP, "The Transform and Data Compression Handbook", CRC Press, 2001, ISBN 0-8493-3692-9.

[11] William K. Pratt, "Digital Image Processing: PIKS Inside", Third Edition, John Wiley & Sons Inc., 2001, ISBN 0-471-37407-5.

[12] D. Huffman, "A method for the construction of minimum redundancy codes", Proc. of the IRE, 40, pp. 1098-1101, 1952.

[13] Thomas Wiegang, Gary J. Sullivan, Gile Bjontegaard, et al., "IEEE Transactions on Circuits and Systems for Video Technology", VOL. 13, No. 7, 2003.

[14] Annapolis Micro, Inc. "WILDCARD™-II Reference Manual", 12968-0000 Revision 2.2, 2004.

[15] VirtexTM-II Platform FPGAs: Complete Data Sheet, DS031, October 14, 2003.

[16] Bob Zeidman, "Introduction to CPLD and FPGA Design", The Chalkboard Network.

[17] U. Meyer-Baese, "Digital Signal Processing with Field Programmable Gate Arrays", Springer, 2001, ISBN 3-540-413413-3.

[18] M. Santos, A. Silva, A. Navarro, "MPEG-4 VLD implementation on a VIRTEX-II", ISO/IEC JTC1/SC29/WG11, M12792, January 2006.

[19] Mihai Sima, Sorin Cotofana, Stamatis Vassiliadis, et al., "MPEG-compliant Entropy Decoding on FPGA-augmented TriMedia/CPU64", IEEE Symposium on Field-Programmable Custom Computing Machines", April 2002.

[20] Bai-Jue Shieh, Yew-San Lee and Chen Yi-Lee, "A High-Troughput Memory-Based VLC Decoder with Codeword Boundary Prediction", IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, no.8, pp. 1514-1521, December 2000.

[21] A. Navarro, A. Silva, M. Santos, et al., "MPEG-4 Inverse Quantizer Hardware Accelerator Implementation in Virtex II", ISO/IEC JTC1/SC29/WG11, M11460, July 2004.

[22] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers", IEEE Trans on ASSP, Vol. 36, N. 9, Sept. 1988, pp. 1445-1453.

[23] A. Navarro, P. Gouveia, and A. Silva, "Delta Rate Control for DV Coding Standard", IEEE Symposium on Consumer Electronics, Sept. 2004, Reading-UK.

[24] Bin Sheng, Wen Gao, Di Wu, "An Implemented VLSI Architecture of Inverse Quantizer for AVS HDTV Video Decoder", IEEE Conference Proceeding, vol. 1, pp. 306-309, October 2005.

[25] A. Navarro, A. Silva, M. Santos, et al., "2-D IDCT hardware accelerator implementation for VIRTEX-II", ISO/IEC JTC1/SC29/WG11, M11459, July 2004.

[26] A. Silva, P. Gouveia, A. Navarro, "Fast Multiplication-free QWDCT for DV coding standard", IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, Feb 2004.

[27] E. Feig, "A fast Scaled-DCT algorithm", Image Algorithms and Techniques, Proc. SPIE Vol. 1244, pp. 2-13, 1990.

[28] "Information Technology—Coding of Audio/Visual Objects", ISO/IEC 14496-2:1999, 1999.

[29] "IEEE Standard Specifications for the implementations of 8x8 Inverse Discrete Cosine Transform", IEEE Std. 1180-1990.

[30] Mihai Sima, Sorin Cotofana, Stamatis Vassiliadis, et al., "An 8x8 IDCT Implementation on a FPGA-Augmented TriMedia", IEEE Symposium on Field-Programmable Custom Computing Machines", pp. 160-169, 2001.

[31] Xilinx, "An Inverse Discrete Cosine Transform (IDCT) Implementation in Virtex for MPEG Video Applications", XAPP208(v1.1), Dec 29, 1999.

[32] Xilinx, "Video Decompression Using IDCT", XAPP611(v1.2), June 3, 2005.

[33] M. Santos, A. Silva, A. Navarro, "MPEG-4 VLD+IQ+IDCT Implementation on a Virtex-II", ISO/IEC JTC1/SC29/WG11, M13139, April 2006.

[34] M. Santos, A. Silva, A. Navarro, "H.264 deblocking filter implementation on a VIRTEX-II", ISO/IEC JTC1/SC29/WG11, M13620, July 2006.

[35] Peter List, Anthony Joch, Jani Lainema, et al., "Adaptive Deblocking Filter", IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp614-619, July 2003.

[36] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, et al., "Architecture design for deblocking filter in H.264/JVC/AVC", Proc. IEEE ICME 2003, vol. 1, pp.693-696, January 2006.

[37] Mustafa Parlak, Ilker Hamzaoglu, "An Efficient Hardware Architecture for H.264 Adaptive Deblocking Filter Algorithm", Adaptive Hardware and Systems (AHS'06), June 2006.

[38] See Yeow Yap and John V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation", IEEE Transactions on Circuits and Systems, vol.51, No.7, pp. 384-389, July 2004.

[39] Choudhury A. Rahman and Wael Badawy, "UMHexagonS Algorithm Based Motion Estimation Architecture for H.264/AVC", Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS'05)", July, 2005.

[40] M. Santos, A. Navarro, "Hardware implementation of full search H.264 motion estimation", ISO/IEC JTC1/SC29/WG11, M13620, October 2006.

ANEXO

No CD-ROM em anexo encontra-se o código VHDL e C/C++ desenvolvido para a elaboração dos módulos apresentados neste trabalho.

A seguinte figura mostra como está organizada a estrutura de ficheiros no CD-ROM com os códigos VHDL e C/C++ desenvolvidos.