Sumário

Neste trabalho foi projectado e implementado um sistema de codificação e descodificação de vídeo que segue a norma H.263, a ser utilizado num sistema de transmissão de vídeo através da rede eléctrica. Os módulos implementados foram baseados nos processadores digitais de sinal (DSPs) do tipo TMS320C50 da Texas Instruments, conseguindo-se obter taxas de codificação entre 6 e 14 imagens por segundo.

Foram ainda desenvolvidos dois co-processadores de DMA baseados em circuitos reconfiguráveis (FPGAs), que efectuam a transferência dos dados correspondentes aos *pixels* das imagens entre os DSPs e o codificador e descodificador dos formatos de vídeo PAL/NTSC.

Por fim, foram implementados circuitos de codificação e descodificação de fonte, baseados no algoritmo de Viterbi, com vista à protecção dos dados contra erros de transmissão pelo canal de comunicação utilizado.

Agradecimentos

O presente relatório representa o culminar de um período de muitos meses de grande empenho para realizar este trabalho final de curso, durante o qual pudemos contar com a ajuda e paciência de um elevado número de pessoas a quem gostaríamos, antes de mais, de mostrar o nosso sincero agradecimento pela sua preciosa colaboração.

Em primeiro lugar, gostaríamos de agradecer ao Grupo de Sistemas de Processamento de Sinal em geral e, em particular, aos nossos professores Prof. José Gerald e Prof. Leonel Sousa, responsáveis pelo projecto, pela sua disponibilidade em esclarecer as nossas dúvidas e por nos terem acompanhado na resolução dos problemas que enfrentámos no decorrer deste trabalho. Não poderíamos esquecer de agradecer, também, ao Prof. Moisés Piedade, pelo seu inesgotável interesse pelo nosso projecto, e ao engenheiro Kevin Ferreira, pelos seus preciosos esclarecimentos sobre a norma de codificação de vídeo utilizada.

Em segundo lugar, gostaríamos de agradecer ao Grupo de Algoritmos de Optimização e Simulação e, em particular, aos professores Arlindo Oliveira e Luís Silveira, que prontamente nos disponibilizaram os seus recursos computacionais, indispensáveis na fase de desenvolvimento dos circuitos de lógica programável (FPGAs). Salientamos também a preciosa ajuda prestada pelos engenheiros António Mota e Nuno Ferreira, que também integram este grupo.

Gostaríamos também de agradecer a colaboração do professor Beltran Almeida, que prontamente nos facultou os circuitos integrados utilizados para realizar as memórias FIFO utilizadas nos módulos de emissão e recepção, e a colaboração do professor Pedro Vítor pelos seus conselhos sobre algumas opções a tomar ao nível da etapa de transmissão dos dados.

Agradecemos também a ajuda prestada pelo professor Horácio Neto e pelos engenheiros Paulo Flores e José Pedro Abreu, na fase de aprendizagem da linguagem de síntese VHDL e de projecto dos circuitos de lógica programável (FPGAs).

Agradecemos ainda aos nossos colegas Pedro Leitão e Pedro Sequeira pela disponibilização das Placas de Aquisição e Visualização por eles desenvolvidas e que utilizamos no nosso trabalho.

Não poderíamos também deixar de agradecer aos nossos colegas e amigos Luís Coelho, Jorge Gonçalves e Carlos Coelho pela sua colaboração e encorajamento constantes ao longo destes últimos meses. Agradecemos, também, à nossa colega e amiga Nília Oliveira pela sua inesgotável paciência e prontidão em nos facultar, sempre que necessário, a sua viatura para as deslocações que necessitámos de efectuar ao longo deste período. De um modo geral, agradecemos a todos os nossos amigos que sempre se mostraram interessados no nosso trabalho, tendo-nos dado o encorajamento necessário nas horas mais difíceis.

Agradecemos, ainda, aos nossos colegas e companheiros de laboratório Carlos Morgado, Ezequiel Pires, Frederico Vieira e João Mestre pelo agradável ambiente de trabalho que nos proporcionaram.

Por fim, não nos poderíamos esquecer de expressar um agradecimento muito especial às nossas famílias que, nos últimos cinco anos, e em particular, nos últimos meses, se revelaram de uma enorme colaboração e paciência para connosco e de uma constante compreensão pelo facto de, em inúmeras circunstâncias, termos estado ausentes.

Índice

Apêndices

Lista de Figuras

Lista de Tabelas

Aplicações de *software* utilizadas

Durante a realização deste projecto, foi necessário recorrer a vários recursos computacionais. Em seguida, enumeram-se as aplicações de *software* que foram mais utilizadas e que, como tal, foram essenciais para o desenvolvimento de várias componentes deste projecto:

Desenvolvimento dos circuitos baseados em FPGAs:

- *VHDL Analyzer* [Versão 1997.08] - Synopsys, Inc. (Compilador de VHDL);
- *VHDL Debugger* [Versão 1997.08] - Synopsys, Inc. (Simulador de VHDL);
- *Design Analyzer* [Versão 1997.08] - Synopsys, Inc. (Síntese de VHDL);
- *XACT Step Design Manager* [Versão M1.3.7] - Xilinx, Inc. (Síntese final e configuração).

Desenvolvimento do código do programação dos DSPs:

- *TMS320C5x COFF Assembler* [Versão 6.60] - Texas Instruments, Inc. (Compilador de *assembly*);
- *TMS320C5x Debugger* [Versão 7.00]/ Simulator [Versão 1.20] - Texas Instruments, Inc. (Simulador);
- *TMS320C5x Debugger* [Versão 7.20]/ XDS510 Emulator [Rev. 3] - Texas Instruments, Inc. (Emulador).

Desenvolvimento do código do programação das PALs:

- *PALASM4* [Versão Market Release 1.5] - Advanced Micro Devices, Inc. (Compilador de PALASM).

Desenvolvimento do código do programação dos microcontroladores PIC:

- *MPLAB for Windows/16* [Versão 3.40.00] - Microchip, Inc. (Compilador de *assembly*, simulador e programador).

Desenho de esquemas eléctricos:

- *Orcad Capture for Windows* [Versão V7.01-S] - Orcad, Inc.

Programas auxiliares:

- *Borland C++* [Versão 3.1] - Borland International, Inc. (Compilador de C++);
- *DJGPP* [Versão 2.01] - DJ Delorie (Compilador C++);
- *Matlab* [Versão 4.2c.1] - The MathWorks, Inc.

1 Introdução

Neste capítulo apresenta-se uma breve introdução ao trabalho realizado, efectuando-se o seu enquadramento e a definição dos objectivos que se propõe atingir.

1.1 ENQUADRAMENTO E OBJECTIVOS DO TRABALHO

Nas últimas décadas, a sociedade tem vindo a ser confrontada com um crescente desenvolvimento de aplicações nos mais variados campos de comunicações, multimédia, telemetria e controlo remoto. Como consequência desta evolução tecnológica, tem-se vindo a verificar um aumento substancial, em termos de dimensão e complexidade, das estruturas de cablagens associadas aos edifícios mais recentes.

Neste contexto, o objectivo deste trabalho consiste em desenvolver um sistema alternativo de comunicação entre vários sistemas, procurando-se evitar o uso de cablagem adicional. A solução adoptada consiste em usar, como canal de comunicação, a rede doméstica de distribuição de energia eléctrica, de 230 V - 50 Hz. Como aplicação para este meio de comunicação, pretende-se desenvolver um sistema que codifica, transmite e descodifica vídeo de acordo com a norma de compressão de vídeo ITU-T H.263 [1] (cuja descrição se apresenta no Apêndice A) usando-se, para tal, um conjunto de três DSP's[1] do tipo TMS320C50 da Texas Instruments [2]: dois DSPs para a codificação e um DSP para a descodificação.

Este sistema poderá ser usado num vasto campo de aplicações incluindo, por exemplo, tele-vigilância e intercomunicadores de vídeo, dispensando-se a instalação de cabos adicionais para a transmissão do sinal entre o emissor e o receptor.

O trabalho realizado deu origem a um artigo apresentado na conferência *"The Second European DSP Education and Research Conference"* organizada pela Texas Instruments [26]. A apresentação teve lugar durante os dias 23 e 24 de Setembro de 1998 na *École Supérieure d' Ingénieurs en Electrotechnique et Electronique-ESIEE* em Noisy le Grand, Paris. Foram apresentados, juntamente com o artigo, alguns resultados experimentais já disponíveis na altura da apresentação. Todo este processo culminou na inclusão do artigo produzido na publicação "*Proceedings – Image Processing"* da Texas Instruments (ver Apêndice K).

1.2 OBJECTIVOS

O objectivo deste trabalho consiste no projecto e implementação de um sistema de codificação e descodificação de vídeo segundo a norma H.263, para ser utilizado com o módulo de transmissão suportado pela rede de distribuição de energia eléctrica.

Para além do módulo de processamento do sinal de vídeo, foi também projectado um sistema de codificação de fonte da trama gerada para a protecção dos dados transmitidos contra ruído inerente ao canal de transmissão.

1.3 ARQUITECTURA GERAL DO SISTEMA

No sistema desenvolvido podem ser identificados dois grandes blocos: um para a emissão do sinal e outro para a respectiva recepção (ver Figura 1.1).

[1] Do inglês *Digital Signal Processors.*

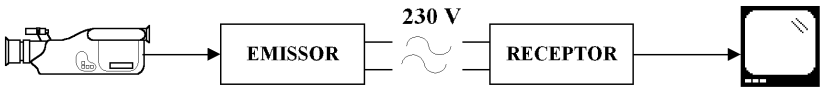


Figura 1.1 - Estrutura do sistema desenvolvido.

O sistema de emissão é constituído por dois módulos: um codificador de H.263 que codifica as imagens de vídeo adquiridas no formato QCIF[2] e um modulador que efectua a transmissão da trama H.263 gerada, modulada em GMSK[3] (ver Figura 1.2).

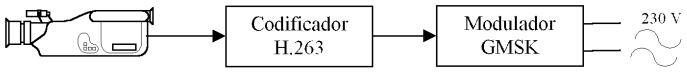


Figura 1.2 - Módulo de emissão.

O módulo de recepção apresenta uma estrutura recíproca da do módulo de emissão. Assim, é constituído por um desmodulador GMSK que desmodula o sinal recebido pela linha de transmissão de energia eléctrica e transfere a trama recebida para o módulo de descodificação H.263. Este último converte a imagem codificada num sinal de vídeo analógico, a ser enviado para um monitor de vídeo PAL[4]/NTSC[5] (ver Figura 1.3).

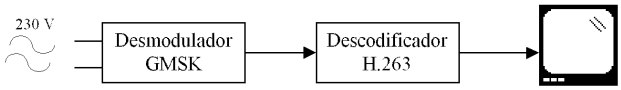


Figura 1.3 - Módulo de recepção.

Na Figura 1.4 apresenta-se um diagrama que ilustra a arquitectura geral do sistema desenvolvido, assim como o enquadramento dos vários módulos descritos no sistema global de comunicação.

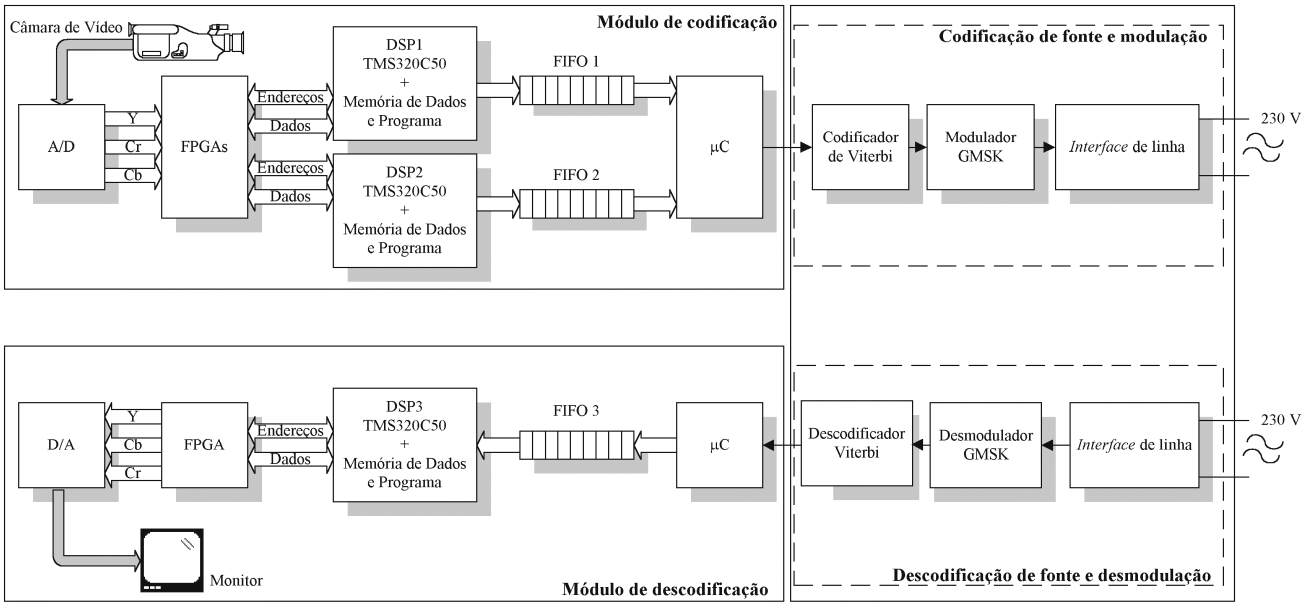


Figura 1.4 - Arquitectura geral do sistema.

[2] Do inglês *Quarter of Common Intermediate Format.*
[3] Do inglês *Gaussian Minimum Shift Keying.*
[4] Do inglês *Phase Alternate Line.*
[5] Do inglês *National Television System Committee.*

2 Sistema de Emissão

Neste capítulo apresenta-se o sistema de emissão desenvolvido, descrevendo-se, com detalhe, o seu funcionamento.

2.1 MÓDULO DE CODIFICAÇÃO

Como se referiu anteriormente, o módulo de codificação tem como principal função a codificação das imagens de vídeo adquiridas por uma câmara. Numa análise a uma granularidade mais fina, pode dividir-se este módulo em sub-módulos com as seguintes funções: *i) conversão analógica-digital (A/D) do sinal de vídeo*; *ii) formatação das amostras recolhidas*; *iii) codificação de vídeo*; *iv) conversão paralelo-série da trama gerada* (ver Figura 2.1).

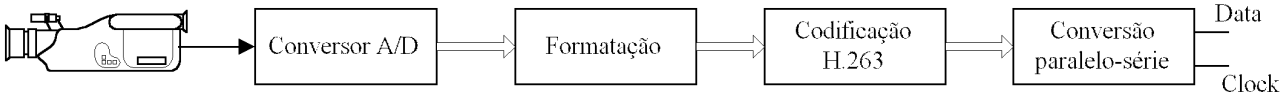


Figura 2.1 - Módulo de codificação.

A codificação das imagens segundo a norma H.263 (cuja descrição se apresenta no Apêndice A) [1] foi realizada com dois processadores digitais de sinal - DSP, do tipo TMS320C50 da Texas Instruments [2]. A escolha deste tipo de DSP foi baseada no facto de se pretender desenvolver um sistema simples e pouco dispendioso, requerendo uma quantidade mínima de *hardware* para a realização de um sistema de processamento de sinal. A escolha efectuada recaiu num DSP de vírgula fixa devido a não se ter a necessidade de efectuar operações de maior precisão que implicariam o uso de um processador de vírgula flutuante como, por exemplo, o TMS320C30 [3].

Neste sistema, foi utilizado processamento paralelo para a codificação de imagem, atribuindo-se sub-imagens a cada um dos DSPs. A decisão de utilizar dois DSPs numa arquitectura paralela prende-se com duas razões fundamentais:

- Um único DSP não dispõe de espaço de endereçamento de memória de dados suficiente para efectuar o processamento. De facto, uma imagem QCIF ocupa:

$$64 \cdot \left({}^{pixels}/_{bloco} \right) \times 6 \cdot \left({}^{blocos}/_{macrobloco} \right) \times 11 \cdot \left({}^{macroblocos}/_{GOB} \right) \times 9 \cdot \left({}^{GOBs}/_{imagem} \right) = 38\,016 \left({}^{pixels}/_{imagem} \right) \quad (1)$$

  Como em qualquer instante é necessário ter em memória a imagem corrente e a sua predição, seria necessário um espaço de endereçamento total de 2 x 38016 = 76032 palavras o que excede o espaço de endereçamento do DSP utilizado (64 kwords).

- Por outro lado, verifica-se que o tempo de processamento necessário para codificar uma imagem num único DSP seria demasiado elevado, pelo que houve a necessidade de introduzir processamento concorrente no sistema. Dado que a quantidade de dados envolvida é elevada, conclui-se que uma arquitectura em *pipeline* acarretaria demasiado tempo de comunicação/sincronização entre o Sistema de Aquisição $\rightarrow$ DSP1 $\rightarrow$ DSP2 $\rightarrow$ *Buffer* de Emissão. Esta opção obrigaria também à utilização de uma quantidade elevada de memória entre cada um dos estágios do *pipeline*.

Assim, optou-se pela codificação paralela de imagem com dois DSPs, atribuindo-se a cada um deles cerca de metade da imagem a codificar. A transferência de dados entre o sistema de aquisição e os DSPs é realizada por um mecanismo de DMA, de forma a torná-la o mais rápida possível. A

sincronização entre os processadores é feita durante a transferência de dados, através de uma ordem de transferência fixa: transferência para o DSP1 seguido da transferência para o DSP2.

Na Figura 2.2 apresenta-se a arquitectura do sistema de codificação H.263 baseado nos dois DSPs.

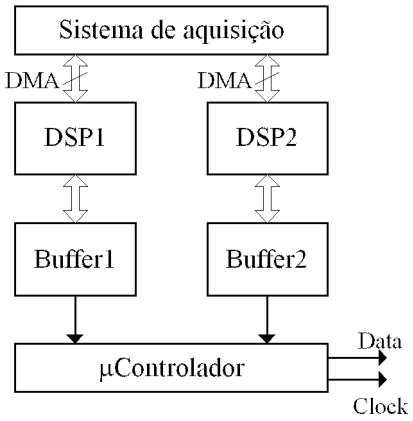


Figura 2.2 - Estrutura do sistema de codificação.

Nas secções seguintes, passar-se-á a descrever, detalhadamente, cada um dos módulos referidos na Figura 2.1.

2.1.1 Aquisição e Conversão A/D do Sinal de Vídeo

Nesta primeira etapa, efectua-se a conversão de uma sequência de imagens analógicas, contínuas no espaço e no tempo, em imagens digitais. Uma sequência de imagens digitais pode ser vista como uma série de imagens bidimensionais ordenadas no tempo. Em geral, estas imagens são adquiridas por intermédio de uma câmara de vídeo, implicando, devido às características inerentes aos sensores de vídeo disponíveis, uma discretização ao nível da amplitude e do espaço do sinal de vídeo. As amostras de uma imagem, que constituem os pontos elementares com níveis discretos, designam-se por *pixels*[6]. Os *pixels* ficam, em geral, equidistantes no espaço, organizados segundo grelhas bidimensionais rectangulares.

A representação de cada *pixel* de uma imagem policromática é, geralmente, representada segundo três componentes designadas por R[7], G[8] e B[9], que apresentam uma elevada correlação entre si. Assim, por forma a utilizar a largura de banda disponível de uma forma mais eficiente, estas componentes são usualmente transformadas noutras três componentes menos correlacionadas: uma de luminância (Y) e duas de crominância (C_B e C_R).

Para efectuar a aquisição do sinal de vídeo analógico e a sua conversão para o formato digital pretendido, foi utilizado um módulo desenvolvido para o efeito. Este módulo baseia-se no circuito integrado Bt812 fabricado pela *Brooktree Corporation* [4] que converte um sinal de vídeo analógico no formato PAL ou NTSC num sinal de vídeo digital RGB ou YC_RC_B.

O circuito acima mencionado disponibiliza uma série de sinais de sincronismo à sua saída, dos quais são utilizados, neste sistema, o sinal `ACTIVE` e o sinal `VALID`. O sinal `ACTIVE` assinala a presença de *pixels* YC_RC_B na saída do circuito, estando desactivado nos retornos horizontais e

[6] Do inglês *picture elements.*
[7] Do inglês *Red.*
[8] Do inglês *Green.*
[9] Do inglês *Blue.*

verticais. O sinal VALID apresenta uma frequência igual à frequência de *pixel*, estando activo durante os períodos em que o sinal ACTIVE está no nível lógico '1', conforme se observa na Figura 2.3.

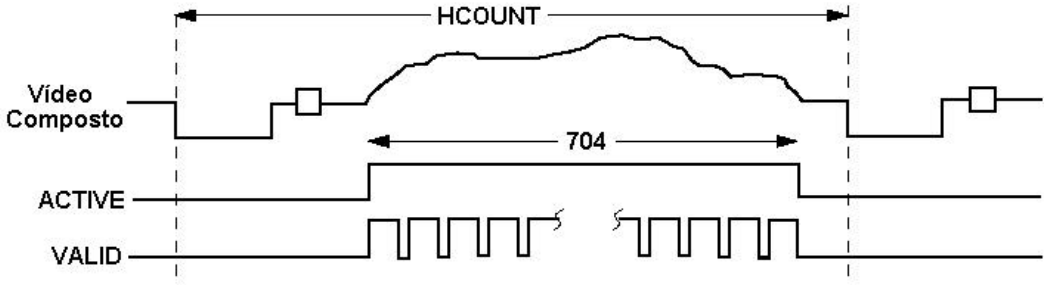


Figura 2.3 - Diagramas temporais dos sinais VALID e ACTIVE.

Por forma a simplificar o projecto da etapa seguinte do sistema no que diz respeito à conversão do formato *raster*, à saída do conversor A/D, para o formato de blocos, característico da norma H.263, foi tomada a opção de se programar o circuito de aquisição para fornecer um número de *pixels* por linha múltiplo de 176. O circuito integrado Bt812 impõe que a frequência de *pixel* esteja compreendida no intervalo entre 1/2 e 8/11 de metade da frequência de funcionamento do circuito, que neste caso é de 30 MHz. Assim, os valores de frequência de *pixel* podem variar entre 10,9 MHz e 15 MHz. Dado que o período de linha do formato PAL é de 64 μs, dos quais só 52 μs correspondem a informação de vídeo útil, o número de *pixels* amostrados por linha varia entre 567 (10,9 MHz x 52 μs) e 780 (15 MHz x 52 μs). Adoptou-se o valor correspondente a 704 *pixels* por linha (174 x 4 = 704).

A escolha entre os diferentes modos de funcionamento deste conversor é efectuada por intermédio da programação dos seus registos internos. Esta programação, foi efectuada usando o microcontrolador do tipo PIC16F84 da Microchip [6] com o programa cuja listagem se apresenta no Apêndice I.1. No mesmo apêndice, é apresentada uma tabela onde constam os diversos registos de programação do Bt812 e os valores com que foram programados para o funcionamento pretendido.

A frequência de relógio do circuito é indicada ao conversor através da programação do registo HCLOCK, que indica o número de ciclos de relógio para uma linha PAL:

$$\text{HCLOCK} = 64\,\mu\text{s} \times 15\,\text{MHz} = 960 \equiv 3\text{C0h} \qquad (2)$$

O tempo entre o sinal de sincronismo horizontal e o início da saída de vídeo activo é programado através do valor do registo HDELAY, como sendo o número de ciclos de relógio de *pixel* que correspondem a esse intervalo:

$$\text{HDELAY} = \frac{704 \times 12\,\mu\text{s}}{52\,\mu\text{s}} \cdot 0{,}5 = 81 \qquad (3)$$

O número de *pixels* activos por linha é programado no registo ACTIVE_PIXELS. Este foi programado com o valor 704 (2C0h).

O número de linhas de vídeo digital no formato *raster* que o conversor deve gerar é programado no registo ACTIVE_LINES. Este registo contém o número de linhas desejado por imagem, correspondendo a dois campos PAL. Dada a estrutura da imagem na norma H.263, esta deve ser constituída por 144 linhas. No entanto, como é necessário amostrar a informação de crominância a um ritmo diferente do ritmo de amostragem da informação de luminância, optou-se por programar o conversor para gerar 288 linhas de vídeo digital. O valor do registo ACTIVE_LINES foi programado com o valor 576. Efectua-se posteriormente a decimação explicitada na figura seguinte,

onde já é tomado em consideração o facto de o número de *pixels* por linha à saída do conversor ser 4 vezes superior ao correspondente na recomendação H.263.

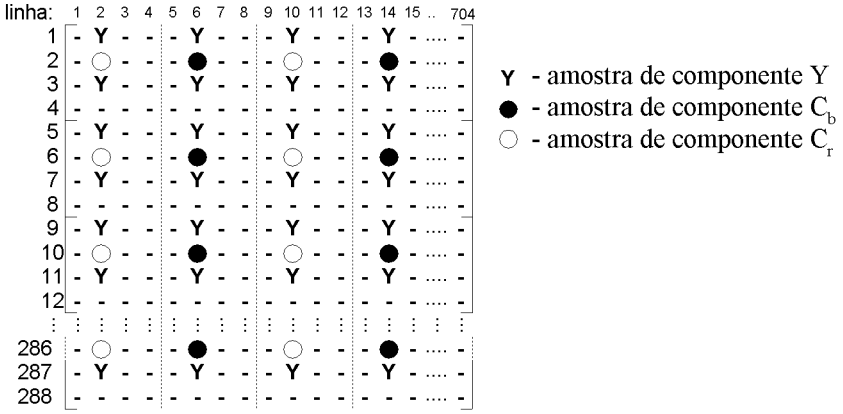


Figura 2.4 - Decimação da imagem gerada pelo Bt812 para formar uma imagem H.263.

A programação da frequência de *pixel* desejada é efectuada através do registo `SampleRateConversion` que é definido como:

$$\text{SampleRateConversionRatio} = \frac{\text{HCLOCK} - \text{DESIRED\_HCLOCK}}{\text{DESIRED\_HCLOCK}} \times 2^{16} \tag{4}$$

em que `DESIRED_HCLOCK` é definido como o número de ciclos de relógio à frequência de *pixel* desejados durante o período de uma linha (isto é, o número de *pixels* por linha). Este valor deve encontrar-se dentro do intervalo:

$$\frac{8}{11} \cdot \text{HCLOCK} < \text{DESIRED\_HCLOCK} < \text{HCLOCK} \tag{5}$$

Feitas estas considerações, apresenta-se o cálculo do valor a preencher no registo `SampleRateConversion`:

$$\text{DESIRED\_HCLOCK} = \frac{704 \times 64\,\mu s}{52\,\mu s} = 866{,}46 \tag{6}$$

$$\text{SampleRateConversionRatio} = \frac{960 - 866{,}46}{866{,}46} \times 2^{16} \approx 7075 \equiv 1\text{BA3h} \tag{7}$$

Por último, resta apenas referir que o valor apresentado no conjunto de 3 registos `P0`, `P1` e `P2` permitem definir o valor da frequência da subportadora de cor que, no sistema PAL [5], é de 4,43361875 MHz. A fórmula que permite calcular esse valor é apresentada em seguida, assim como o cálculo do valor a inserir nos registos referidos:

$$\texttt{P} = \frac{\text{Freq.Subportadora de cor}}{f_{CLOCK \times 1}} \times 2^{22} \tag{8}$$

$$= \frac{4{,}43361875}{15} \times 2^{22} \approx 1239730 \equiv 12\text{EAB2h} \equiv \underbrace{12}_{P2}\,\underbrace{EA}_{P1}\,\underbrace{B2}_{P0} \tag{9}$$

2.1.2 Formatação das Amostras Recolhidas

Como referido anteriormente, a norma H.263 define um formato de vídeo digital em que a imagem é dividida em blocos de 8 x 8 *pixels*. Como o formato de vídeo digital à saída do conversor A/D utilizado é do tipo *raster*, foi necessário efectuar a conversão entre esses dois formatos de forma a

colocar, em memória, os blocos H.263 organizados por macroblocos da forma ilustrada na Figura 2.5.

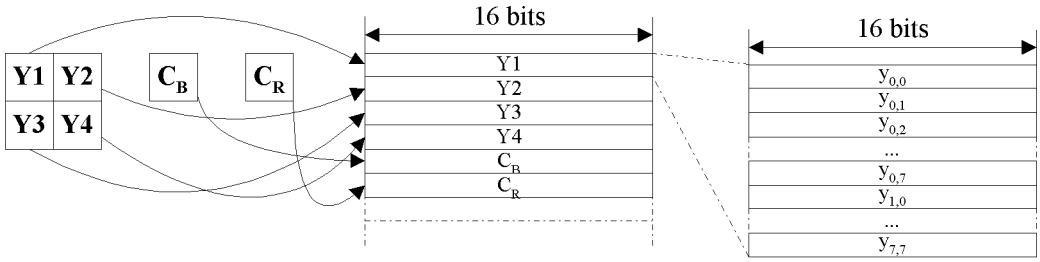


Figura 2.5 - Estrutura dos dados na memória.

As componentes $y_{i,j}$ correspondem aos *pixels* de luminância que se situam na linha *i* e na coluna *j* do bloco de luminância correspondente. A utilização desta estrutura permitirá, como será descrito adiante na secção que se refere à codificação H.263, utilizar um algoritmo mais eficiente para o cálculo da DCT[10] [7] que se baseia na multiplicação de matrizes organizadas segundo a ordem descrita na Figura 2.5.

Para efectuar a conversão entre os formatos de vídeo foi desenvolvido um co-processador que efectua, numa primeira fase, a transferência dos *pixels* provenientes do conversor A/D para uma memória estática do tipo IS61C1024-20 (128 kBytes) da ISSI [8], onde são armazenados segundo a estrutura matricial mencionada. Numa segunda fase, os dados nessa memória são transferidos para as memórias dos DSPs sendo, em simultâneo, efectuada a diferença entre a imagem existente nos DSPs e a imagem adquirida mais recentemente, poupando assim tempo de processamento. O co-processador desenvolvido utiliza dois circuitos do tipo *Field Programmable Gate Array* (FPGA) da Xilinx (XC4010e-2 [9]), encontrando-se o respectivo esquema no Apêndice E.1.

Como se pode verificar pela análise da figura no Apêndice E.1, a FPGA1 é responsável pela transferência dos *pixels* do conversor A/D para a memória estática, utilizando EPROMs[11] para efectuar a geração dos endereços que permitem a reorganização da imagem para o formato utilizado na recomendação H.263. Uma segunda FPGA (FPGA2) efectua a transferência por DMA entre a memória dos dois DSPs e a memória estática externa. Durante esta transferência, a FPGA2 encarrega-se de efectuar a subtracção entre os valores das amostras actuais e os valores presentes nas memórias dos DSPs.

2.1.2.1 Reorganização da imagem

Dada a complexidade dos circuitos digitais a utilizar para efectuar a reordenação da imagem, constatou-se que a sua implementação com circuitos integrados discretos iria dar origem à utilização de um grande número destes circuitos. A solução encontrada foi a de utilizar circuitos do tipo FPGAs, permitindo não só uma redução significativa na dimensão física do sistema, como também tornar o sistema mais flexível e adaptável a outras soluções, através da programação do seu funcionamento.

A descrição dos circuitos a implementar nas FPGAs foi feita utilizando a linguagem de descrição de *hardware* VHDL [10]. A tradução do código VHDL para os circuitos lógicos correspondentes e o mapeamento dos circuitos nas células da FPGA foram efectuados com os programas *Design Analyser* da *Synopsys* [11] e o *Design Manager - Xilinx Alliance* [12]. O facto de a linguagem

[10] Do inglês *Discrete Cosine Transform.*
[11] Do inglês *Erasable Programmable Read Only Memory.*

VHDL permitir uma certa abstracção em relação ao *hardware* constitui uma desvantagem no sentido em que se pode, facilmente, perder o controlo sobre o resultado da síntese dificultando, sobretudo, a fase de teste do funcionamento da própria FPGA. Para minimizar este problema, optou-se por implementar, numa primeira fase, pequenos blocos funcionais que foram sintetizados separadamente, tendo construído os restantes à base destes blocos funcionais. De entre os blocos funcionais implementados para as três FPGAs salientam-se: *flip-flops*, *multiplexers*, somadores e contadores. Para os somadores, adoptou-se uma estrutura do tipo *carry look ahead* [13], que permite diminuir o tempo de um soma, em relação ao somador do tipo *ripple-carry*.

Apresenta-se, em seguida, um esquema funcional da estrutura interna da FPGA1.

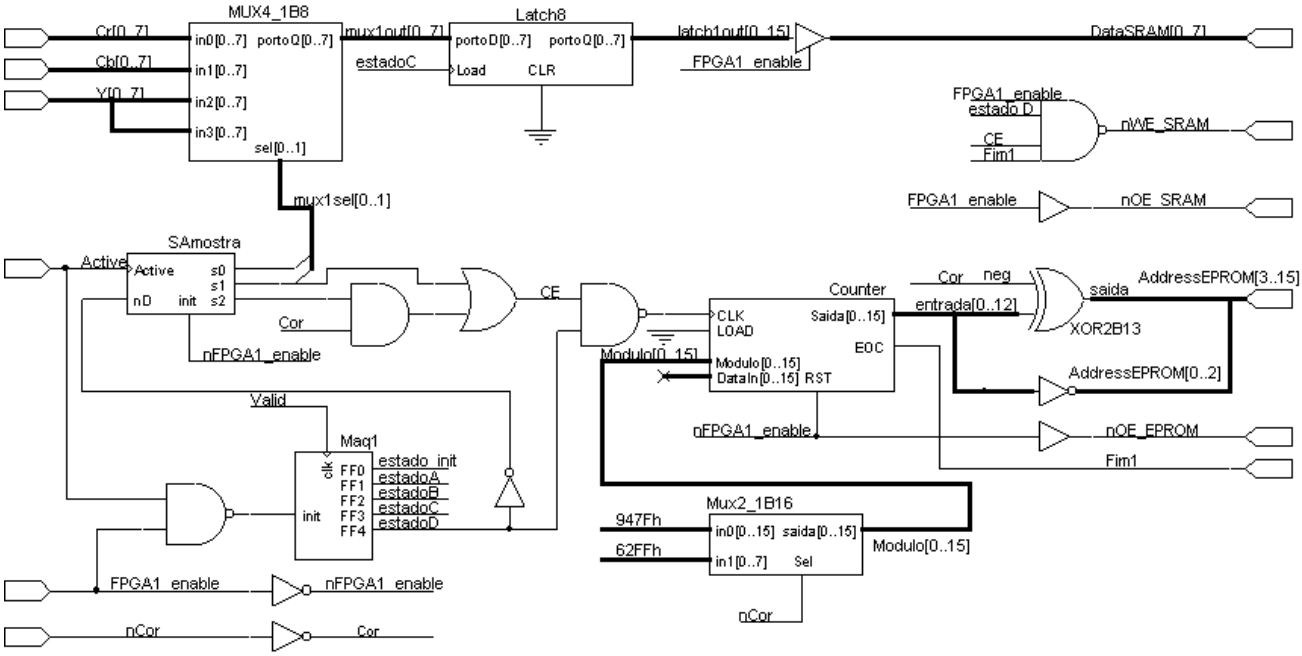


Figura 2.6 - Esquema funcional da FPGA1.

O multiplexer `MUX4_1B8`, em conjunto com o *latch* `Latch8`, efectua a multiplexagem das três componentes de vídeo. Esta multiplexagem é controlada pelas máquinas de estados `Samostra` e `Maq1`. A geração de endereços para a SRAM[12] e para a EPROM é efectuada pelo contador `Counter` cujo sinal de relógio é comandado, também, pelas duas máquinas. De acordo com o modo de funcionamento (monocromático ou policromático), o multiplexer `MUX2_1B16` controla qual o final de contagem do referido contador. Os sinais de controlo para a SRAM e para a EPROM são também gerados por esta FPGA.

Como é possível observar na figura apresentada no Apêndice E.1, ambas as FPGAs acedem à mesma memória estática. Esta característica impõe a existência de uma sincronização entre a FPGA1 e a FPGA2. Esta sincronização é obtida a partir dos sinais `FPGA1_enable` (entrada na FPGA1) e `Fim1` (saída da FPGA1). Na descrição do funcionamento da FPGA2 será explicado o método de sincronização em mais detalhe, podendo-se referir que o sinal `FPGA1_enable`, activo com o nível lógico '`1`', requer à FPGA1 a transferência de uma nova imagem para a SRAM, cabendo à FPGA2 detectar o início de uma nova imagem na saída do conversor A/D. A FPGA1 assinala a conclusão da sua tarefa através da activação da saída `Fim1`.

[12] Do inglês *Static Random Access Memory*.

Como referido, o co-processador deve efectuar uma decimação dos *pixels* recebidos do conversor A/D, armazenando apenas 1 *pixel* em cada 4 recebidos. Esta decimação é controlada pela máquina de estados `Maq1` (cuja descrição VHDL se apresenta no Apêndice E.2.3.2). Apresenta-se na figura seguinte o esquema que representa a constituição da máquina de estados `Maq1`.

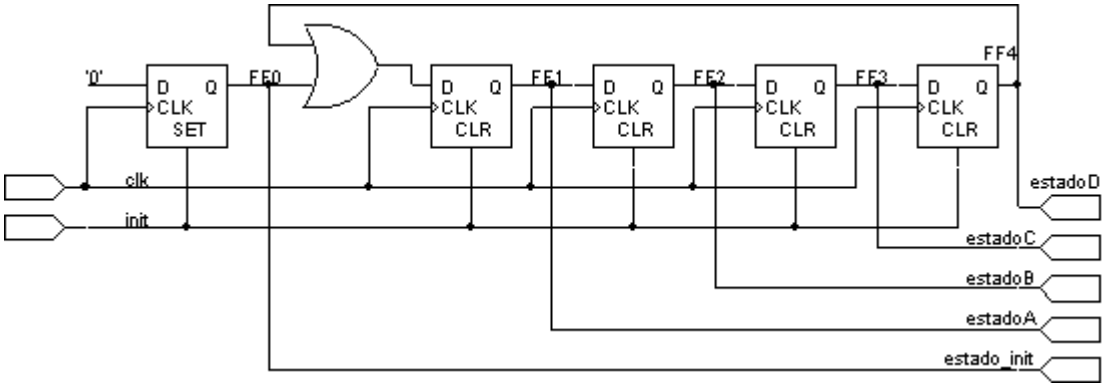


Figura 2.7 - Máquina de estados Maq1.

O relógio que comanda esta máquina de estados tem como frequência a frequência de *pixel*. Assim, apenas num em cada 4 ciclos de relógio é que é escrito, na memória estática, o valor correspondente a um dado *pixel*. Essa escrita acontece na transição do `estadoC` para o `estadoD` (ver Figura 2.6 e Figura 2.7). O *pixel* escrito nessa ocasião é o *pixel* que foi colocado à saída do *latch* `LATCH8T` no início do `estadoC`. Na transição descendente do `estadoD`, o contador de 16 bits (cuja descrição VHDL se apresenta no Apêndice E.2.3.5) é então avançado de um unidade. Dado ser este contador que gera os endereços para a EPROM que, por sua vez, contém o endereço de escrita na memória estática, o caminho crítico neste circuito é formado pelo caminho que se inicia com a transição do `estadoD` para o `estadoA`, passando pelo estabelecimento das saídas do contador e tendo, como última etapa, o acesso à EPROM. As EPROMs utilizadas foram do tipo AM27C64 (8 kBytes) [14] com tempos de acesso de 120 ns. Foi esta a razão que levou a adoptar a solução de avançar o contador em 3 ciclos do relógio de *pixel*, antes de efectuar a escrita conseguindo-se, assim, uma janela de tempo de $\frac{52\,\mu s}{704} \times 3 = 220\,ns$. Apresenta-se, em seguida, um diagrama temporal que pretende ilustrar o funcionamento descrito.

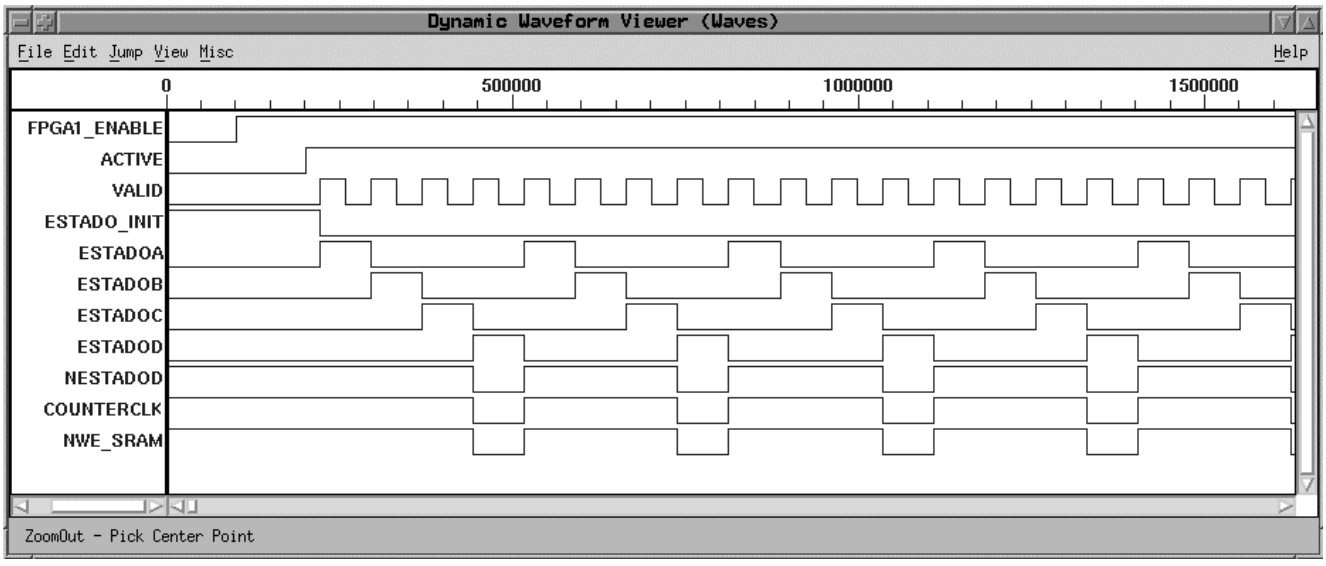


Figura 2.8 - Diagrama temporal da `Maq1`.

O bloco combinatório na saída do contador de 16 bits, constituído por 13 portas XOR, permite o funcionamento deste sistema tanto no modo policromático como no modo monocromático. De facto, existem duas tabelas guardadas na EPROM: a tabela correspondente aos endereços do modo monocromático (armazenados desde o endereço `0000h` até ao endereço `0C60h`), e a tabela correspondente aos endereços do modo policromático (armazenados desde o endereço `0D70h` até ao endereço `1FFFh`). A selecção entre uma tabela e outra é feita bastando negar (através das portas XOR) o barramento de endereços da EPROM, ligado à saída do contador de 16 bits referido.

O conjunto formado pela máquina de estados `SelAmostra` (cuja descrição VHDL se apresenta no Apêndice E.2.3.3) e o *multiplexer* `MUX4_1B8` (ver Figura 2.6 e descrição VHDL no Apêndice E.2.3.4) permite efectuar a selecção de qual das componentes de vídeo é escrita na memória estática num dado instante, para a situação representada na Figura 2.4. Na Figura 2.9, representa-se a máquina de estados `SelAmostra`.

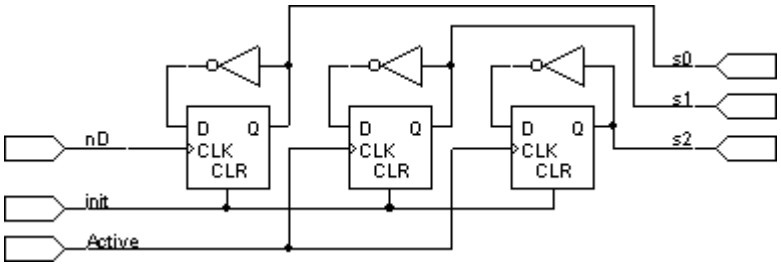


Figura 2.9 - Máquina de estados `SelAmostra`.

A comutação entre as componentes de luminância e as de crominância efectua-se ao nível da linha, pelo que o sinal de relógio que comanda o funcionamento desta máquina de estados é o sinal `ACTIVE`, descrito na secção 2.1.1. Esta máquina gera, de cada vez que existe uma transição de linha, um sinal de comando para o *multiplexer* de forma a que, caso as componentes a escrever sejam as de luminância, os sinais de controlo `S0` e `S1` alternem entre `S1S0=10` e `S1S0=11`. Caso contrário, alternam entre `S1S0=00` e `S1S0=01`. Como para cada conjunto de 4 linhas só se armazena um linha de crominância, o sinal de relógio do contador de 16 bits e o sinal de comando de escrita na SRAM (`WE`) são desactivados pelo sinal `S2` de 4 em 4 linhas.

A saída `Fim1`, que assinala o final de transferência efectuada pela FPGA1, é obtida a partir da comparação do endereço gerado pelo contador de 16 bits com os valores das constantes K_{YY} (176 x 144 x 1,5) ou K_{YC} (176 x 144), dependendo de se estar no modo monocromático ou policromático. Apresenta-se no Apêndice E.2.3.1 a descrição VHDL deste comparador, bem como a interligação dos blocos anteriormente mencionados. A descrição dos portos de entrada e saída da FPGA1 é apresentada no Apêndice E.2.1.

2.1.2.2 Transferência DMA e cálculo da diferença entre a imagem actual e a sua predição

A FPGA2 é responsável pela partição da imagem contida na SRAM e na correspondente atribuição às memórias dos dois DSPs. No entanto, tal como se encontra descrito na recomendação H.263 [1] (ver Apêndice A), o modo de funcionamento INTER requer que seja efectuada a subtracção entre a imagem recentemente adquirida e o resultado da descodificação da imagem anterior, presente na memória do DSP. Assim, antes de efectuar a escrita do novo valor na memória do DSP, a FPGA2 efectua a subtracção do valor presente na memória do DSP à amostra adquirida. Apresenta-se, na Figura 2.10, um esquema que pretende ilustrar o funcionamento da FPGA2.

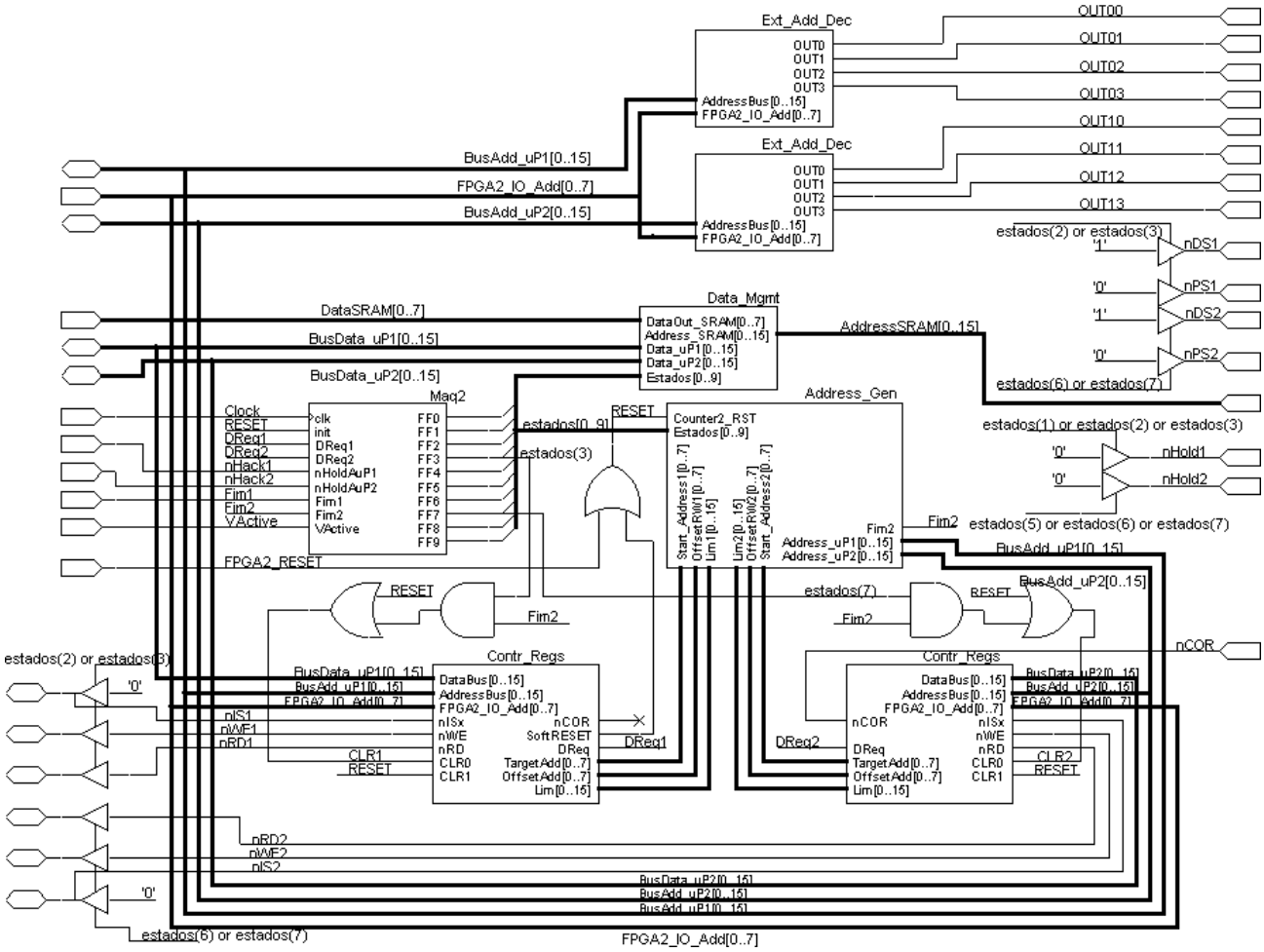


Figura 2.10 - Esquema funcional da FPGA2.

Os módulos `Address_Gen`, `Data_Mgmt` e a máquina de estados `Maq2` implementam um controlador de DMA. A programação desta FPGA por parte de ambos os DSPs é efectuada através da programação de diversos registos implementados nos módulos `Contr_Regs`. Foi, por último, efectuada a descodificação de alguns endereços auxiliares nos módulos `Ext_Add_Dec`.

O conjunto de 2 bancos de registos (cuja descrição VHDL se encontra nos Apêndices E.3.4.1 e E.3.4.2), correspondentes a cada um dos DSPs, permitem que cada DSP efectue a sua própria configuração do co-processador. Cada registo implementado segue uma estrutura idêntica à descrita na Figura 2.11, em que os sinais `nRD` e `nWE` são, respectivamente, os sinais de controlo de leitura e escrita do DSP e `nIS` é activado num acesso do DSP ao seu espaço de I/O.

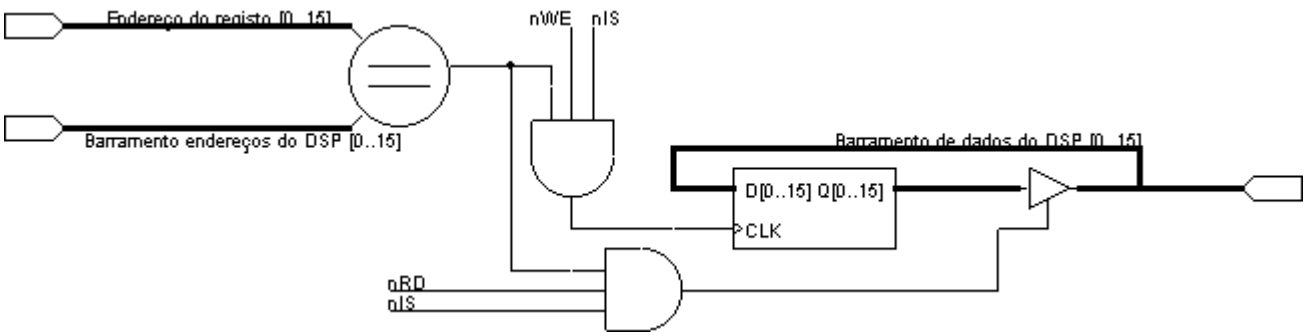


Figura 2.11 - Registo de controlo da FPGA2.

Na Tabela 2.1 são apresentados os vários registos referidos, onde o endereço de cada um dos registos corresponde aos 8 bits menos significativos do barramento de endereços, já que o byte mais significativo é obtido a partir da entrada `FPGA2_IO_ADD`.

Registo	Endereço	DSP1	DSP2
`Limite_1`	`50h`	R/W	
`Limite_2`	`50h`		R/W
`Target/Offset_1`	`51h`	R/W	
`Target/Offset_2`	`51h`		R/W
`nCor`	`52h`	R/W	R
`Reset`	`53h`	W	W

Tabela 2.1 - Endereços dos registos de configuração do co-processador de aquisição.

O registo `nCor` permite ao DSP identificado como DSP1 proceder à escolha entre o modo de funcionamento policromático (‘0’) ou monocromático (‘`1`’). A definição de qual o modo a utilizar é dada pelo bit menos significativo deste registo.

O registo de `Reset` permite, a qualquer um dos DSPs, efectuar a inicialização da FPGA2 bastando-lhe, para tal, efectuar um acesso de escrita para o endereço correspondente a este registo.

Os registos `Target/Offset` foram implementados para cada um dos dois DSPs e permitem, de uma forma simples, definir as zonas de memória dos respectivos DSPs que tomam parte na transferência de dados. Em conjunto com o valor escrito no registo `Limite`, o mapa de memória dos DSPs fica definido da seguinte forma:

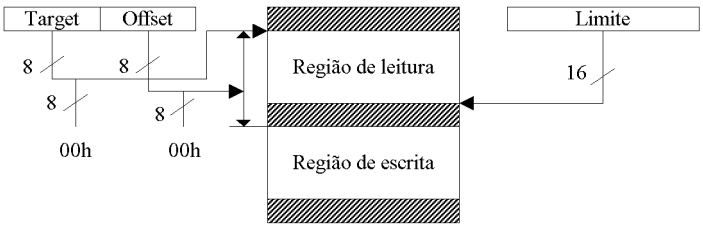


Figura 2.12 - Mapa de memória dos DSPs.

Os registos `Limite` são, ainda, utilizados para dar início às transferências DMA, o que acontece logo após a escrita neste registo por parte do DSP.

O controlo global das operações efectuadas tanto pela FPGA2 como pela parte restante do co-processador (nomeadamente, a FPGA1) é feito pela máquina de estados `Maq2`, cujo descrição em VHDL se encontra no Apêndice E.3.4.3 e cujo diagrama de estados é apresentado na Figura 2.13.

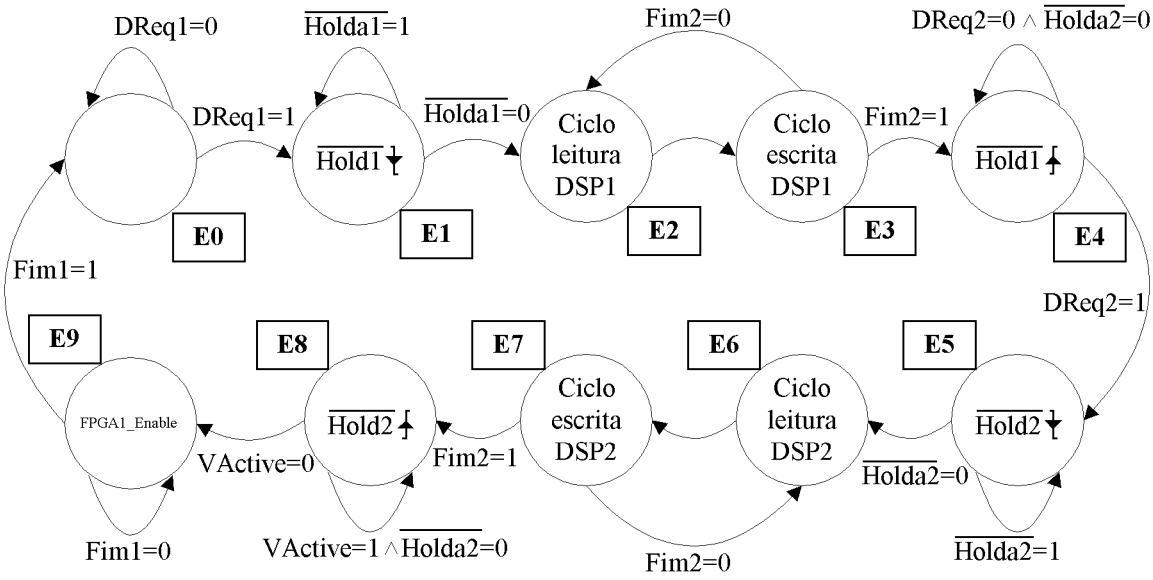


Figura 2.13 - Máquina de estados `Maq2` (diagrama de estados).

No estado inicial (`E8`), a máquina encontra-se a aguardar a activação da linha $\overline{\text{VACTIVE}}$ por parte do conversor A/D, assinalando o início da aquisição de uma nova imagem. Quando tal se verifica, transita-se para o estado `E9` onde activa a linha `FPGA1_Enable`, permitindo a transferência dos *pixels* do conversor para a SRAM através da FPGA1. Assim que esta terminar, transita para o estado `E0` onde aguarda o pedido, por parte do DSP1, da transferência DMA. Consumado este pedido, transita para o estado `E1` onde activa o sinal $\overline{\text{HOLD}}$ deste DSP e aguarda pela sua resposta afirmativa através da activação da linha $\overline{\text{HOLDA}}$ iniciando, após a confirmação, o ciclo de transferência (estados `E2` e `E3`). No estado `E2` é efectuada a leitura da SRAM, enquanto que no estado `E3` é efectuada a escrita para a memória do DSP. Após terminar a transferência (`Fim2='1'`), evolui-se para o estado `E4`, desactivando-se a linha de $\overline{\text{HOLD}}$ do DSP1 e aguardando pela desactivação de $\overline{\text{HOLDA}}$ por parte deste DSP. Mantém-se este estado até que exista um pedido de transferência DMA do DSP2, sob a forma da activação de `DREQ2`, resultante do preenchimento do registo `Limite` com um valor diferente de zero por este DSP. Após o pedido, o processo descrito para o DSP1 repete-se para o DSP2 durante os estados `E5`, `E6`, `E7` e `E8`. Neste último estado, volta a aguardar pela activação de $\overline{\text{VACTIVE}}$. Na figura seguinte, apresenta-se um diagrama temporal para os principais sinais associados a esta máquina de estados.

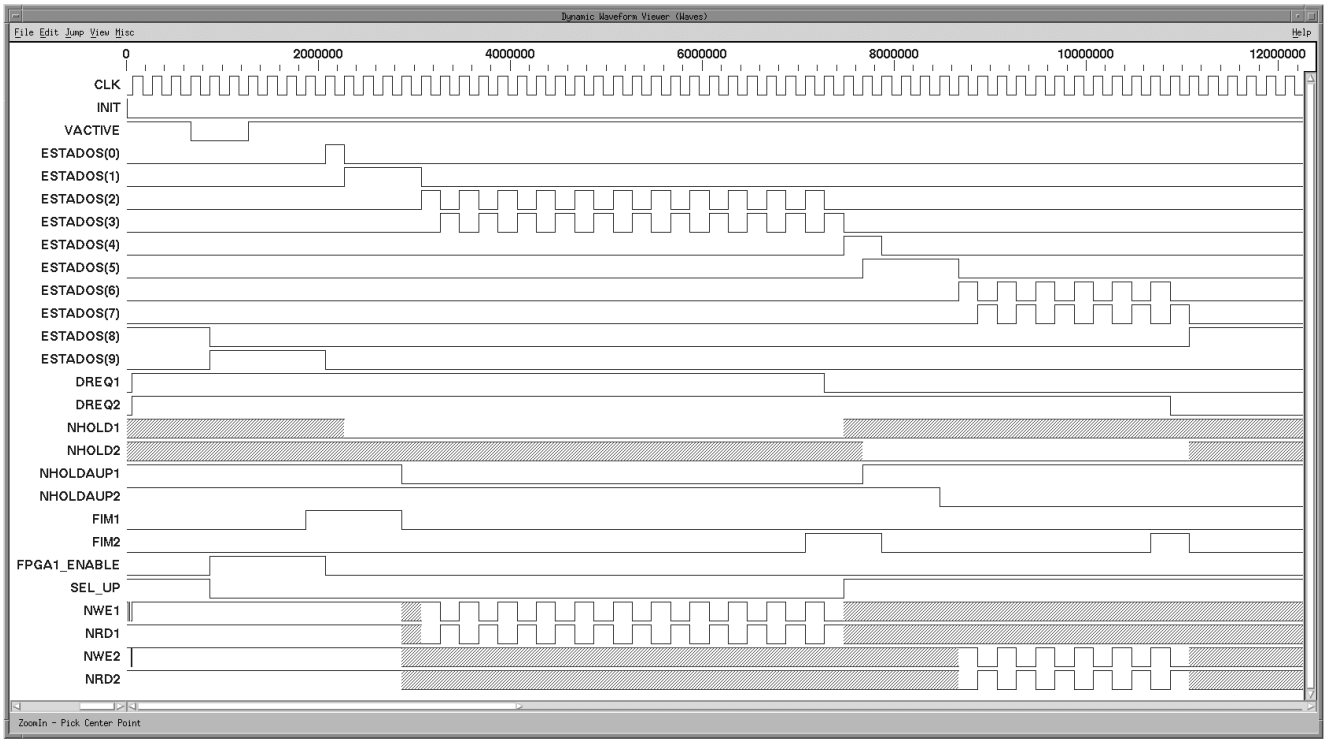


Figura 2.14 - Máquina de estados `Maq2` (diagrama temporal).

Dado esta máquina ser responsável pelo controlo de grande parte dos restantes módulos implementados nesta FPGA, apresenta-se no Apêndice E.3.3 a tabela de estados onde constam os sinais de controlo e os seus valores lógicos para cada estado da máquina.

A geração dos endereços de memória no controlador de DMA é efectuada no módulo `Address_Gen` (descrito no Apêndice E.3.4.4), cujo esquema se apresenta na Figura 2.15.

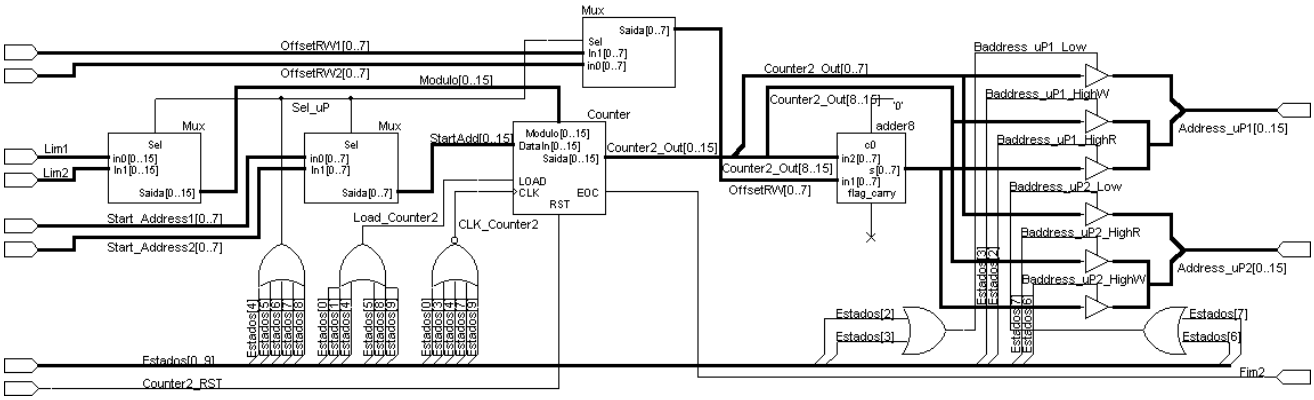


Figura 2.15 - Circuito de geração de endereços para a memória do DSP.

Este circuito é constituído, basicamente, por um contador (descrito em VHDL no Apêndice E.3.4.7) cujo início e fim de contagem são dependentes da variável `Sel_uP`, que indica qual o DSP para que se efectua a transferência. Contém ainda um somador para que, na fase de escrita nas memórias dos DSPs, ao endereço de leitura seja somado o deslocamento referido no registo `Target/Offset` (mais exactamente, na parte correspondente ao *offset*).

A geração dos endereços para o acesso à SRAM do co-processador é efectuada no módulo `Data_Mgmt` (descrito no Apêndice E.3.4.5) cujo esquema se apresenta na Figura 2.16 e que é, também, responsável por efectuar a subtracção já referida anteriormente.

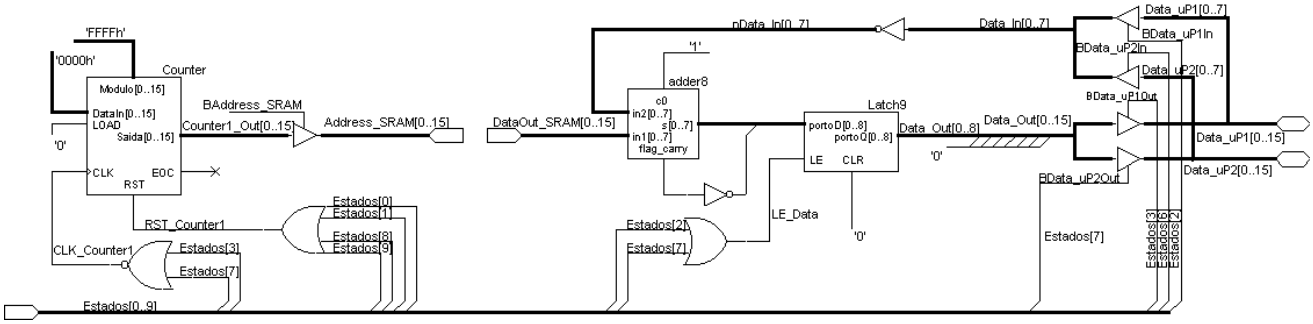


Figura 2.16 - Leitura da SRAM e subtracção das amostras com sua a predição, guardada na memória do DSP.

Este módulo subtrai o valor de cada *pixel* presente na SRAM (utilizando um somador de 8 bits cuja descrição VHDL se encontra no Apêndice E.3.4.8) com o valor correspondente na memória do DSP, estando este último negado e tendo colocado o bit de *Carry in* do somador a '1'. Desta forma, o resultado à saída do somador corresponde à subtracção do valor na memória do DSP ao valor na SRAM do co-processador.

Conforme será descrito na secção 2.1.4 referente ao circuito onde se efectua a geração da trama, é necessário que ambos os DSPs tenham a possibilidade de endereçar dispositivos externos. Para tal, foram implementados, como parte do co-processador e na FPGA2, dois descodificadores de endereços (cuja descrição é apresentada no Apêndice E.3.4.6) que permitem, a cada DSP, aceder a 4 dispositivos através da activação das saídas OUT correspondentes, cujos endereços são apresentados na Tabela 2.2.

Endereços	**Saída Correspondente**	
	DSP1	**DSP2**
54h	OUT00	OUT00
55h	OUT01	OUT01
56h	OUT02	OUT02
57h	OUT03	OUT03

Tabela 2.2 - Descodificação dos endereços dos registos auxiliares do co-processador de aquisição.

Tal como no caso anterior, os endereços apresentados correspondem aos 8 bits menos significativos, sendo os bits mais significativos definidos pelo valor da entrada `FPGA2_IO_Addr`. A descrição VHDL desta FPGA encontra-se no Apêndice E.3.4.1. A descrição dos seus portos de entrada e saída é apresentada no Apêndice E.3.1.

2.1.3 Codificação H.263

Na implementação do codificador desenvolvido foi omitido o módulo de estimação de movimento para a codificação das imagens do tipo INTER. Este módulo, embora previsto e descrito na recomendação H.263 [1], tem um carácter opcional. A decisão de não o incluir prende-se com o facto de tal requerer capacidades de processamento que tornariam o tempo total de codificação demasiado elevado, caso fosse realizado com os processadores utilizados. Para além disso, a implementação da estimação de movimento implicaria a utilização de um espaço de armazenamento em memória não disponível no sistema desenvolvido. O diagrama de blocos do codificador toma, então, a forma apresentada na Figura 2.17.

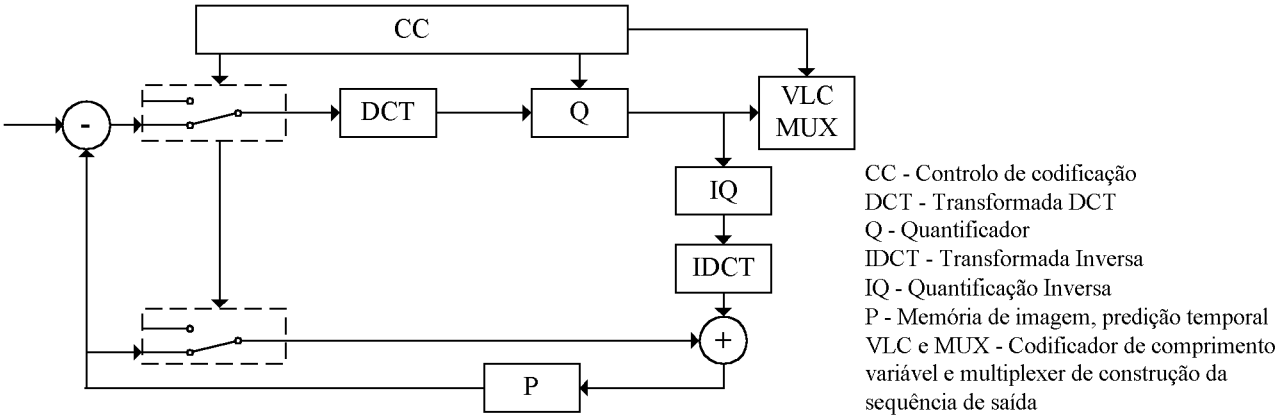


Figura 2.17 - Diagrama de blocos do codificador implementado.

A programação do TMS320C50, para a codificação H.263, foi realizada em *Assembly* [2], de forma a utilizar eficientemente os recursos do processador.

Em seguida, passar-se-á a descrever os vários módulos do programa para o codificador H.263. Houve o cuidado de desenvolver o programa de uma forma modular e estruturada, seguindo o fluxo de dados no codificador, tal como apresentado na Figura 2.17. Os módulos desenvolvidos têm a estrutura apresentada na Figura 2.18:

i) no módulo *parametrização do codificador*, realizam-se os procedimentos para a transferência de uma nova imagem e desencadeia-se o processo de codificação;

ii) no módulo *codificação directa de imagem*, calcula-se a DCT e efectua-se a quantificação dos coeficientes para uma imagem completa, preenche-se os campos da trama de bits correspondente à imagem e inicia-se o processamento adicional para cada um dos GOBs;

iii) o módulo de *interface* recebe cada um dos GOBs da imagem codificada, preenche os campos da trama de bits correspondente e invoca o módulo seguinte de processamento ao nível dos macroblocos;

iv) o módulo *VLC e codificação inversa* preenche o campo na trama de bits correspondente a um macrobloco, aplica um código de comprimento variável aos coeficientes quantificados da DCT e realiza o procedimento de codificação inversa para construção da imagem de predição.

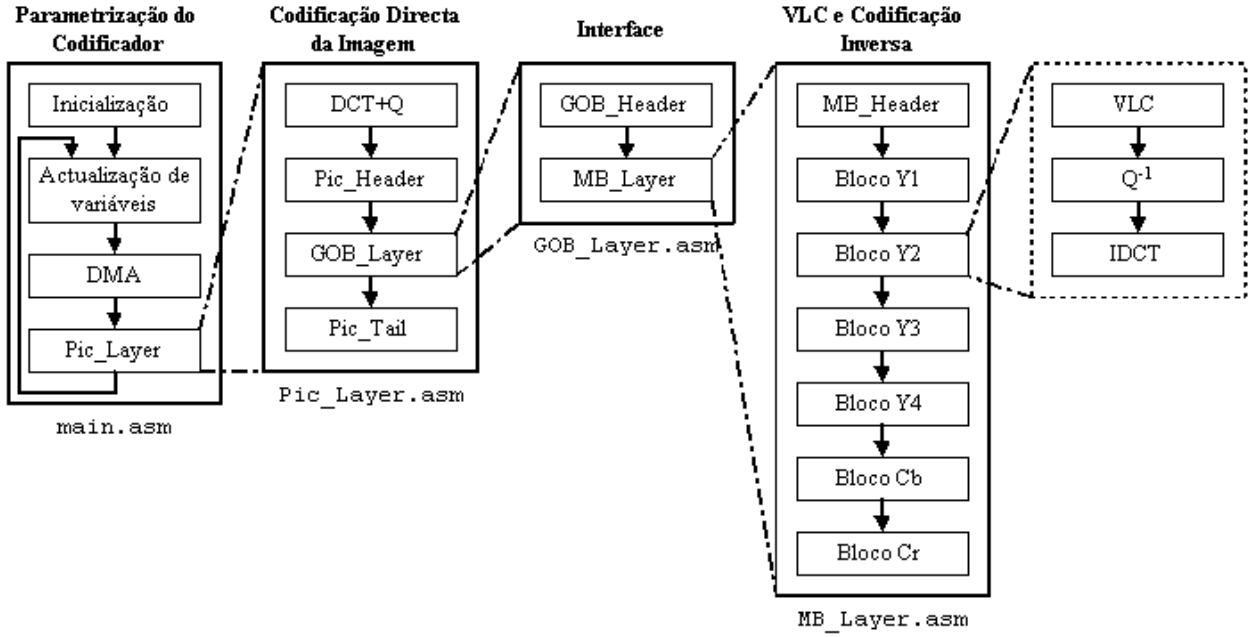


Figura 2.18 - Estrutura hierárquica dos vários módulos do programa do codificador H.263.

2.1.3.1 Módulo *parametrização do codificador*

O topo desta hierarquia é ocupado pelo módulo *parametrização do codificador*, que corresponde ao ficheiro "`main.asm`", cuja listagem se apresenta no Apêndice C.2. Este módulo é executado após a inicialização do DSP. Nele, cada DSP começa por invocar as várias rotinas que inicializam alguns dos blocos do codificador, tais como blocos das transformadas DCT e IDCT[13] ("`tranini`") e do codificador de comprimento variável ("`vlc_ini`"). De seguida, é realizada a actualização do passo de quantificação a utilizar, dependendo do nível actual do *buffer* de emissão. Determina-se também se a imagem a codificar deverá ser do tipo INTRA ou do tipo INTER. Para finalizar, é efectuado o preenchimento dos registos da FPGA2, que determinarão a forma como se irá processar a transferência dos dados para a memória do DSP através de um ciclo de DMA.

Durante esta transferência, o co-processador implementado pelas FPGAs fará a leitura da predição da imagem actual (resultante da descodificação da imagem anteriormente transmitida) a partir do endereço de memória `2C00h`, subtraindo-a à imagem adquirida. O resultado desta subtracção será escrito na memória de cada DSP a partir do endereço `8200h`, conforme se verifica na Figura 2.19.

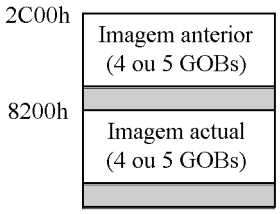


Figura 2.19 - Mapa de memória de cada DSP.

Desta forma, o algoritmo de codificação irá funcionar, por defeito, no modo INTER visto aproveitar a capacidade de predição do codificador. No entanto, para codificar macroblocos, ou mesmo a totalidade da imagem, no modo INTRA, o DSP apenas terá de preencher a memória respeitante à imagem de predição com o valor zero, de forma a que a imagem diferença corresponda à imagem adquirida.

A utilização da memória do DSP ao longo do processo de codificação é a apresentada na Figura 2.20.

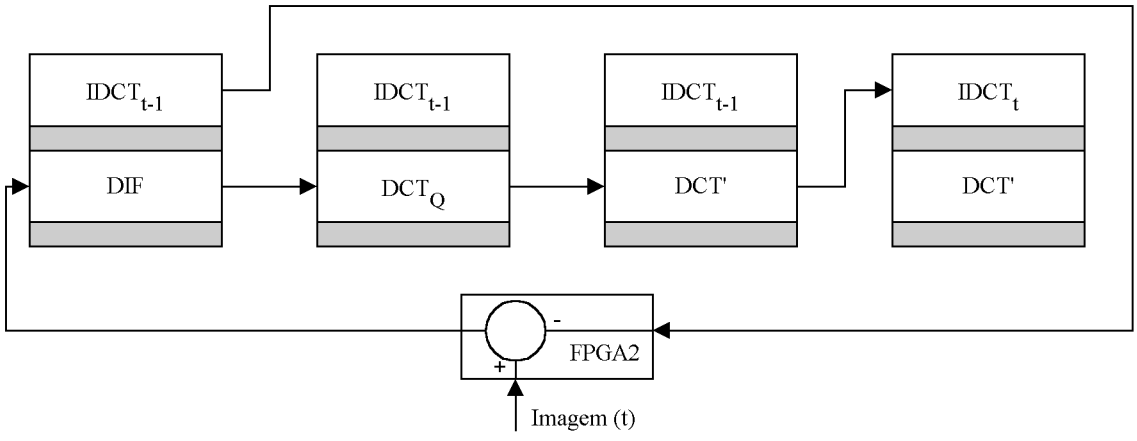


Figura 2.20 - Utilização da memória de cada DSP ao longo da codificação.

[13] Do inglês *Inverse Discrete Cosine Transform.*

De seguida, o processador desencadeia a fase de codificação e passa a executar o módulo *codificação directa da imagem*.

2.1.3.2 Módulo *codificação directa da imagem*

Este módulo, cujo código se apresenta no Apêndice C.3, começa por calcular a transformada de coseno para todos os macroblocos da imagem. A definição desta transformada é dada pela seguinte expressão:

$$F(u,v) = \frac{C(u).C(v)}{4} \cdot \sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y).\cos\left[\frac{p\ (2.x+1)}{16}\cdot u\right].\cos\left[\frac{p\ (2.y+1)}{16}\cdot v\right] \qquad (10)$$

com $u, v, x, y = 0, 1, 2, \ldots 7$

e onde x, y – coordenadas espaciais do *pixel* no bloco da imagem;
u, v – coordenadas no domínio da transformada;

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & ,k = 0 \\ 1 & ,k \neq 0 \end{cases}.$$

Esta equação pode ser reescrita da seguinte forma, evidenciando um produto de matrizes:

$$F(u,v) = \sum_{x=0}^{7}\left[\frac{1}{2}\cdot C(u)\cdot\cos\frac{(2.x+1)\cdot u\cdot\pi}{16}\cdot\sum_{y=0}^{7} f(x,y)\cdot\left(\frac{1}{2}\cdot C(v)\cdot\cos\frac{(2.y+1)\cdot v\cdot\pi}{16}\right)\right] \qquad (11)$$

Assim, considerando **A** a matriz 8 x 8 dos *pixels* do bloco em questão e **F** a matriz dos coeficientes obtidos após a aplicação da transformada, pode-se representar a expressão (11) como um produto de matrizes:

$$\mathbf{F} = \mathbf{C}\ \mathrm{x}\ \mathbf{A}\ \mathrm{x}\ \mathbf{C}^{\mathrm{T}} \qquad (12)$$

em que **C** é a matriz de coeficientes, obtida a partir das expressões seguintes:

$$C = \begin{bmatrix} \alpha & \alpha & \alpha & \alpha & \alpha & \alpha & \alpha & \alpha \\ \sigma & \varepsilon & \mu & \tau & -\tau & -\mu & -\varepsilon & -\sigma \\ \beta & \delta & -\delta & -\beta & -\beta & -\delta & \delta & \beta \\ \varepsilon & -\tau & -\sigma & -\mu & \mu & \sigma & \tau & -\varepsilon \\ \alpha & -\alpha & -\alpha & \alpha & \alpha & -\alpha & -\alpha & \alpha \\ \mu & -\sigma & \tau & \varepsilon & -\varepsilon & -\tau & \sigma & -\mu \\ \delta & -\beta & \beta & -\delta & -\delta & \beta & -\beta & \delta \\ \tau & -\mu & \varepsilon & -\sigma & \sigma & -\varepsilon & \mu & -\tau \end{bmatrix} \qquad (13)$$

em que : $C_{i,j} = \frac{1}{2}\cdot C(i)\cdot\cos\frac{(2.j+1).i.\pi}{16}$ (14)

ou seja,

$$\alpha = \frac{1}{2.\sqrt{2}}, \quad \beta = \frac{1}{2}\cdot\cos\left(\frac{\pi}{8}\right), \quad \delta = \frac{1}{2}\cdot sen\left(\frac{\pi}{8}\right), \quad \sigma = \frac{1}{2}\cdot\cos\left(\frac{\pi}{16}\right)$$
$$\varepsilon = \frac{1}{2}\cdot\cos\left(\frac{3.\pi}{16}\right), \quad \mu = \frac{1}{2}\cdot sen\left(\frac{3.\pi}{16}\right), \quad \tau = \frac{1}{2}\cdot sen\left(\frac{\pi}{16}\right) \qquad (15)$$

Por forma a realizar o cálculo descrito na equação (12), realizou-se o código correspondente à seguinte expressão:

$$\mathbf{Y} = (\mathbf{M_D} \times \mathbf{M_P}^T)^T \tag{16}$$

em que $\mathbf{M_D}$ é uma matriz existente em memória de dados, que corresponde à matriz **A**, e $\mathbf{M_P}$ é uma matriz existente em memória de programa, que corresponde à matriz **C**. Este código representa o produto de matrizes com uma execução muito rápida num DSP, visto que é possível a aplicação directa da instrução '`MAC`' que efectua, num único ciclo, a multiplicação de um elemento presente em memória de dados com um outro presente na memória de programa e somar o resultado ao valor presente no acumulador.

Manipulando a equação (16) é possível obter:

$$\mathbf{F} = \left[\left(\mathbf{A}.\mathbf{C}^T\right)^T.\mathbf{C}^T\right]^T = \mathbf{C}.\left[\left(\mathbf{A}.\mathbf{C}^T\right)^T\right]^T = \mathbf{C}.\mathbf{A}.\mathbf{C}^T \tag{17}$$

Com base na equação (17), basta executar duas vezes consecutivas o código correspondente à expressão (16), em que, num primeiro passo, $\mathbf{M_D}$=**A** e $\mathbf{M_P}$=**C** e, num segundo passo, $\mathbf{M_D}$ é o resultado do primeiro passo e $\mathbf{M_P}$=**C**. Apresenta-se, no Apêndice C.7, o código correspondente ao cálculo da transformada DCT.

Faz-se notar que existem alguns algoritmos, referidos na literatura [7], que são mais eficientes do que o usado neste projecto no que se refere ao número de operações aritméticas efectuadas. Contudo, devido à natureza não sequencial dos acessos aos dados e da execução do programa, estes algoritmos não são adequados para o cálculo da DCT em DSPs. Por exemplo, implicam o uso intensivo de instruções de salto (*branch*) que, dada a estrutura do *pipeline* do DSP utilizado, requerem um total de 4 ciclos. Desta forma, o processador passaria a maior parte do tempo a efectuar indexações à memória e a executar instruções de controlo. Concluiu-se assim que, devido à sua estrutura e características de programação, o algoritmo baseado na multiplicação de matrizes é o mais adequado para implementações baseadas em DSPs.

Com o objectivo de acelerar o processamento correspondente à codificação da imagem, implementou-se o bloco de quantificação em simultâneo com a DCT. De facto, como se optou por utilizar passos de quantificação correspondentes a potências de 2, o bloco quantificador resume-se a uma operação de deslocamento do resultado da DCT para a direita. A vantagem em implementar estes dois blocos em simultâneo é evidente, evitando-se duas operações adicionais por cada *pixel* da imagem: uma de leitura da memória (instrução `lacl`) e outra de escrita para a memória (instrução `sacl`). Como cada imagem contem 38016 *pixels*, isso implicava um tempo adicional de processamento de 2 x 38016 x 50 ns = 3,8016 ms.

Foi também com o propósito de tornar a codificação mais rápida que se optou por se preencher, em simultâneo com o cálculo da DCT, uma tabela denominada de CBPC, contendo uma descrição de cada macrobloco transformado. Cada entrada desta tabela (correspondente a um dado macrobloco da imagem), tem a estrutura apresentada na Figura 2.21.

Bits:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	**0**	**Y1**	**Y2**	**Y3**	**Y4**	**Cb**	**Cr**	**0**	**0**	**Y1**	**Y2**	**Y3**	**Y4**	**Cb**	**Cr**
Significado:	'1'= Bloco tem coeficiente DC diferente de zero '0'= Bloco tem coeficiente DC nulo								'1'= Bloco tem pelo menos um coeficiente não-DC diferente de zero '0'= Bloco tem todos os coeficientes não-DC nulos							

Figura 2.21 - Estrutura de cada entrada da tabela CBPC.

Assim, um bit com o nível lógico '1' no byte mais significativo, significa que o bloco da posição correspondente tem o coeficiente DC com um valor não nulo. Se este bit tiver o nível lógico '0', significa que este coeficiente apresenta um valor nulo. No caso dos bits correspondentes ao byte menos significativo, um bit com o nível lógico '1' significa que o bloco da posição correspondente tem pelo menos um coeficiente não-DC diferente de zero. No caso do nível lógico desse bit ter o valor '0', significa que todos os coeficientes não-DC desse bloco são nulos.

Esta tabela permite, para além de obter a informação necessária para determinar os códigos correspondentes aos campos MCBPC e CBPY a enviar no cabeçalho de cada macrobloco (ver Apêndice A), tornar a execução de alguns módulos subsequentes mais rápida, como é o caso do módulo de determinação dos códigos VLC dos coeficientes de cada bloco e o módulo de cálculo da transformada inversa IDCT, tal como será explicado nas secções seguintes.

Terminada a fase de cálculo da transformada DCT, o programa procede ao preenchimento dos diversos campos correspondentes ao cabeçalho da *picture* a codificar, conforme descrito na recomendação H.263 (ver Apêndice A). Os bits correspondentes a estes campos são escritos numa zona de memória denominada de "`buf_vlc`". Neste *buffer*, o programa regista o valor de cada campo (ou grupo de campos) através de 2 palavras de 16 bits: na primeira palavra, insere o código VLC propriamente dito, alinhado à esquerda; na palavra seguinte, indica o número de bits válidos do código presente na palavra anterior. Desta forma, torna-se possível ao programa escrever os diversos campos com comprimentos arbitrários entre 1 e 16 bits.

A concatenação dos bits correspondentes a estes campos será então feita por uma rotina denominada de "`PutBits`", invocada por todos os módulos do programa. Com esta rotina, simplifica-se o preenchimento dos diversos campos da trama H.263

Terminado o preenchimento dos vários campos correspondentes ao cabeçalho da *picture*, o programa procede à execução dos módulos correspondentes a cada GOB, invocando o módulo *VLC e codificação inversa* através do módulo de *interface*.

Por fim, o programa executa o preenchimento dos campos correspondentes ao *tail*, onde se inclui a sequência de fim de *picture* - `EOS`[14]. Nesta fase são também inseridos bits de *stuffing* por forma a garantir o alinhamento ao byte requerido pela norma (ver Apêndice A).

2.1.3.3 Módulo *interface*

Este módulo, cujo código se apresenta no Apêndice C.4, começa por proceder ao preenchimento do cabeçalho do nível de GOB, cuja estrutura se encontra descrita no Apêndice A, juntamente com a descrição da norma H.263. Um dos campos pertencentes a este cabeçalho refere-se ao passo de quantificação utilizado nos coeficientes não DC dos blocos (campo `GQUANT`). Para além disso, este

[14] Do inglês *End Of Sequence*.

módulo invoca, para cada um dos 11 macroblocos que constituem o GOB, o módulo de *VLC e codificação inversa*. De forma a evitar o enchimento do *buffer* dos códigos VLC (‘`buf_VLC`’), procedeu-se à execução da rotina `PutBits` entre cada chamada à função `MB_Layer`.

2.1.3.4 Módulo *VLC e codificação inversa*

Este módulo, cujo código se apresenta no Apêndice C.5, implementa o conjunto de operações referentes à codificação de um macrobloco. Para isso, começa por efectuar o preenchimento do cabeçalho, onde se incluem os campos MCBPC e CBPY. A informação correspondente a estes campos pode ser facilmente deduzida a partir da tabela CBPC, referida na secção 2.1.3.2, preenchida aquando do cálculo da DCT. Como o codificador realizado não implementa o bloco de compensação de movimento, o campo correspondente do cabeçalho do nível de macrobloco foi preenchido com os códigos correspondentes aos vectores (horizontal e vertical) de deslocamento nulo.

Neste módulo realiza-se a codificação inversa e a codificação com VLC correspondente a cada um dos blocos que constituem o macrobloco: Y1, Y2, Y3, Y4, C_B e C_R. Para cada um destes blocos, começa-se por chamar a função `VLC`, cujo código se apresenta no Apêndice C.9. Esta função realiza a leitura dos coeficientes quantizados segundo uma sequência em *zig-zag*, conforme descrito na norma e ilustrado na Figura A.5. Sempre que um dos coeficientes lidos tiver um valor não nulo, procede-se à determinação do código variável correspondente e à sua escrita no *buffer* ‘`buf_VLC`’. Caso o valor do coeficiente em causa não se encontre tabelado na norma, procede-se à determinação do código VLC correspondente, através da sequência de *escape*, também prevista na norma. Por fim, determina-se ainda se o coeficiente lido corresponde ao último coeficiente não nulo desse macrobloco. Nesse caso, procede à determinação do código VLC correspondente, quer este se encontre tabelado na norma, quer este seja obtido através de uma sequência de *escape*.

Finalizada a codificação dos coeficientes, o programa procede à sua desquantificação, de acordo com o valor do passo de quantificação utilizado. Dado existirem dois quantificadores possíveis (tal como se descreveu no Apêndice A), o programa irá executar operações distintas, dependendo de se tratar de uma imagem INTRA ou INTER. Para além disso, como apenas se utilizaram passos de quantificação correspondentes a potências de 2, este bloco resume-se, na realidade, a um deslocamento dos bits do coeficiente para a esquerda.

Seguidamente, o programa procede ao cálculo da transformada de coseno inversa – IDCT, cujo código se apresenta no Apêndice C.8.

O cálculo desta transformada é baseado na expressão seguinte:

$$f(x,y) = \frac{C(u).C(v)}{4} \cdot \sum_{x=0}^{7}\sum_{y=0}^{7} F(u,v).\cos\left[\frac{p\ (2.x+1)}{16}\cdot u\right].\cos\left[\frac{p\ (2.y+1)}{16}\cdot v\right] \qquad (18)$$

com $u, v, x, y = 0, 1, 2, \ldots 7$

e onde x, y – coordenadas espaciais no domínio do *pixel*;

u, v – coordenadas no domínio da transformada;

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & ,k = 0 \\ 1 & ,k \neq 0 \end{cases}.$$

Manipulando a equação (12), obtém-se:

$$\mathbf{F} = \mathbf{C} \cdot \mathbf{A} \cdot \mathbf{C}^{\mathbf{T}} \Leftrightarrow \mathbf{A} = \mathbf{C}^{-1} \cdot \mathbf{F} \cdot \left(\mathbf{C}^{\mathbf{T}}\right)^{-1} \tag{19}$$

Contudo, considerando que a matriz de coeficientes **C** é uma matriz ortonormada, isto é, $\mathbf{C}^{-1} = \mathbf{C}^{\mathbf{T}}$, a expressão anterior fica:

$$\mathbf{A} = \mathbf{C}^{T} \cdot \mathbf{F} \cdot \mathbf{C} \tag{20}$$

Fazendo a mudança de variável $\mathbf{D} = \mathbf{C}^{\mathbf{T}}$, obtém-se:

$$\mathbf{A} = \mathbf{D} \cdot \mathbf{F} \cdot \mathbf{D}^{\mathbf{T}} \tag{21}$$

Como se pode constatar, a expressão obtida é em tudo idêntica à expressão (12) para o cálculo da DCT, à parte da matriz de coeficientes ser agora a matriz transposta da matriz usada para o cálculo da DCT. Isto significa que o algoritmo é idêntico ao usado anteriormente para o cálculo da DCT. Desta forma, justifica-se a semelhança entre os ficheiros "`idct.asm`", apresentado no Apêndice C.8 e o ficheiro "`dct.asm`", descrito anteriormente.

O facto de se ter feito a qualificação de macroblocos através do preenchimento da tabela CBPC aquando do cálculo da DCT pode, em geral, tornar o processamento ao nível do macrobloco mais rápido. Para isso, foi utilizado o algoritmo apresentado na Figura 2.22.

1. No caso de todos os coeficientes de um determinado bloco tomarem o valor nulo, não é necessário efectuar qualquer tipo de processamento, pelo que o programa pode passar para o bloco seguinte;
2. Se apenas o coeficiente DC for não nulo, pode-se verificar que:

$$f(x,y) = \frac{C(u).C(v)}{4} \cdot \sum_{x=0}^{7}\sum_{y=0}^{7} F(u,v).\cos\frac{(2.x+1)p\ .u}{16}.\cos\frac{(2.y+1)p\ .v}{16}\Bigg|_{u,v=0} \tag{22}$$

$$= \frac{C(0).C(0)}{4} \cdot F(0,0) = \frac{1}{4} \cdot \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} \cdot F(0,0) \tag{23}$$

$$= \frac{1}{8} \cdot F(0,0) \tag{24}$$

   Assim, é apenas necessário preencher o valor $\frac{F(0,0)}{8}$ em todos os *pixels* de um determinado bloco. Para além disso, após se proceder à codificação do coeficiente DC não se verifica a necessidade de procurar outros coeficientes nas restantes 63 posições, pelo que o programa pode continuar a processar o bloco seguinte.
3. Se o bloco tiver coeficientes não DC não nulos, então terá de ser aplicado o algoritmo mais lento, baseado na multiplicação de matrizes descrito anteriormente.

Figura 2.22 - Algoritmo para o cálculo da IDCT e codificação de coeficientes VLC.

2.1.3.5 Funções auxiliares do programa

No Apêndice C, apresenta-se um conjunto de ficheiros que, embora necessários à execução da codificação, não contêm código correspondente a qualquer tipo de processamento. É o caso, por exemplo, do ficheiro "`Mem_org.h`" onde se estruturou e organizou a memória de dados utilizada pelo programa.

2.1.4 Controlo de "buffer" e geração de trama

O módulo de controlo de *buffer* e geração de trama tem, como o próprio nome indica, as seguintes funções (ver Figura 2.23):

- Ler as palavras escritas pelos DSPs, que constituem a trama de bits resultante do codificador e armazená-las nas memórias do tipo FIFO[15];
- Serializar as palavras a emitir pelo sistema de transmissão, por forma a gerar a trama de bits no formato H.263.

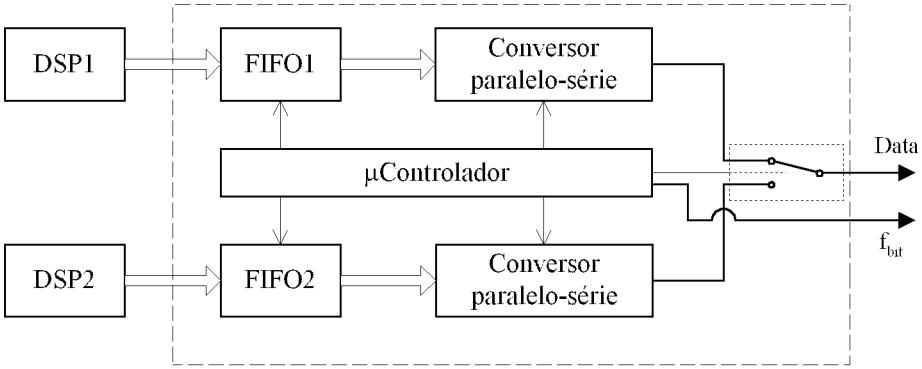


Figura 2.23 - Estrutura geral do controlador de *buffer* de emissão e geração de trama.

O controlo global destas funções é efectuado com base num microcontrolador do tipo PIC16C74A [6].

As palavras são escritas por cada um dos DSPs em 2 memórias do tipo FIFO: `FIFO1` e `FIFO2`, realizadas através de integrados do tipo TMS4C1050 da Texas Instruments [15]. Como cada um destes circuitos integrados apresenta uma capacidade de 256k x 4 bits, foram necessárias 4 unidades para implementar cada um dos FIFOs, com uma capacidade total de 256k x 16 bits. Na Tabela 2.3 são referidos os principais sinais de controlo deste circuitos:

Sinal	Descrição
`WE`	'Write Enable' – Permite activar ou desactivar a entrada, mantendo intacto o ponteiro de escrita enquanto mantiver o nível lógico '`0`';
`SWCK`	'Serial Write Clock' – Efectua a escrita de dados no flanco ascendente quando o sinal `WE` apresenta o nível lógico '`1`';
`RSTW`	'Reset Write' – Inicializa o ponteiro de escrita da memória;
`RE`	'Read Enable' – Permite activar ou desactivar a saída, (colocando-a no estado de alta-impedância) e manter o ponteiro de leitura intacto enquanto mantiver o nível lógico '`0`';
`SRCK`	'Serial Read Clock' – Efectua a leitura de dados no flanco ascendente quando o sinal `RE` apresenta o nível lógico '`1`';
`RSTR`	'Reset Read' – Inicializa o ponteiro de leitura da memória.

Tabela 2.3 - Sinais de controlo das memórias do tipo FIFO utilizadas.

[15] Do inglês *First In First Out.*

Para garantir o correcto funcionamento destes circuitos, constituído por células de memória dinâmica, tem que se atender às seguintes restrições:

- inicializar os ponteiros de escrita e leitura no início da operação;
- manter o ponteiro de escrita avançado do ponteiro de leitura de pelo menos 600 posições, para se conseguir ler os dados anteriormente escritos;
- efectuar os ciclos de escrita e de leitura com um período não superior a 1 ms (sinais SWCK e SRCK), caso em que se tornaria necessário realizar o procedimento de inicialização, colocando ambos os ponteiro de escrita e leitura no endereço zero;
- manter os períodos em que as entradas WE e RE se encontram no nível lógico '0' com um intervalo de tempo não superior a 1 ms, caso em que seria, também, necessário realizar o procedimento de inicialização.

Como se pode constatar a partir do que foi descrito (nomeadamente a partir das duas últimas restrições), torna-se evidente que as características das memórias utilizadas não são as mais adequadas para esta aplicação. De facto, faz-se notar que a opção de se terem utilizado prende-se exclusivamente com razões de ordem económica, pois os circuitos integrados alternativos mostraram-se bastante mais dispendiosos.

Ligaram-se os barramentos de dados de cada um dos DSPs às entradas DIN das memórias FIFO, conforme se ilustra no esquema global deste circuito, apresentado no Apêndice G. Os sinais de WE e SWCK de controlo de escrita dos FIFOs foram gerados a partir dos sinais $\overline{WE}$ dos DSPs e dos sinais OUT00 e OUT01 provenientes da FPGA2, correspondentes à descodificação do endereço 0054h de cada um dos DSPs.

Por forma a garantir a correcta sequencialidade da trama H.263, foi necessário garantir que as 2 sub-tramas geradas por cada um dos DSPs são correctamente utilizadas para a criação da trama final. Para tal, utilizaram-se dois circuitos integrados FIFO adicionais, por forma a armazenarem um bit de controlo gerado por cada DSP, através da linha correspondente à *flag* externa XF, que se denominou por bit 17, B17. Assim, o bit B17 de cada DSP deve ser escrito com o valor '0' sempre que se escreve uma palavra para o FIFO correspondente a esse DSP. A excepção a esta regra ocorre sempre que o DSP escrever a última palavra da sua sub-trama, situação em que deverá colocar o bit 17 com o nível lógico '1'. Este bit assinala ao módulo que procede à união da trama a altura em que deve comutar entre um FIFO e outro. Este módulo foi implementado com base na máquina de estados, cujo diagrama se apresenta na Figura 2.24.

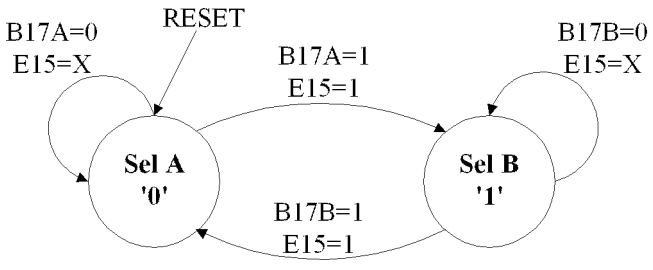


Figura 2.24 - Máquina de estados de formação da trama H.263.

Esta máquina de estados foi implementada por intermédio de uma PAL[16] (pal1) do tipo PALCE22V10 da AMD [15], cujo código de programação se apresenta no Apêndice J.1.

[16] Do inglês *Programmable Array Logic*.

Como se pode observar, existem apenas 2 estados, correspondentes à leitura de um dos FIFOs ou do outro. A comutação entre os estados ocorre quando se verifique que o nível lógico da entrada `B17` em causa se encontrar com o valor '`1`', e a entrada `E15`, cujo significado se explicará adiante, se encontrar activa (valor lógico '`1`').

Por forma a efectuar a serialização da trama, as saídas das memórias FIFO são ligadas a um registo de deslocamento de 16 bits, implementado por intermédio de dois circuitos integrados do tipo 74LS166, conforme se ilustra no esquema da Figura 2.25.

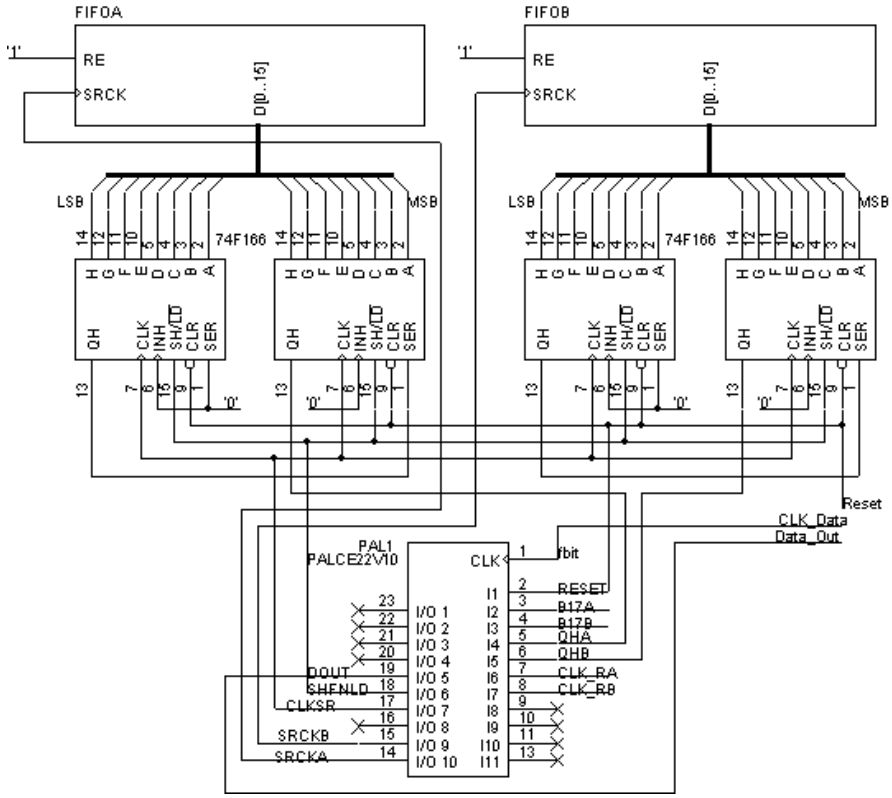


Figura 2.25 - Esquema do módulo de serialização da trama.

Os sinais de controlo destes registos são gerados por intermédio da máquina de estados referida e por um contador de 4 bits, ambos implementados na PAL. O contador de 4 bits tem como função gerar o sinal `E15` (referido anteriormente), com uma frequência 16 vezes inferior à frequência de bit e que efectua a geração dos sinais de controlo de leitura dos FIFOs (`RE` e `SRCK`). Assim, em cada período do sinal de relógio de bit, é realizado um deslocamento de um bit do registo activo. Ao fim de 15 deslocamentos, o sinal `E15` é activado e a linha de leitura do registo de deslocamento em causa permite o carregamento da próxima palavra de 16 bits a serializar. Desta forma, obtém-se, como sinais de saída, os sinais `fbit` e `Data_Out`, correspondentes ao relógio de bit e à linha de dados com a trama H.263. Estes sinais irão, então, ser passados ao módulo de codificação de fonte.

Por forma a que o DSP possa, em qualquer instante, avaliar o nível de enchimento do seu *buffer* de emissão, implementou-se um circuito que mantém o número de palavras guardadas em memória, por intermédio de contadores do tipo 74HCT4040 e de um microcontrolador PIC16C74A.

Contudo, uma vez que a capacidade de cada *buffer* (256 k x 16 bits) implica o uso de 18 bits para contar o número de palavras em memória, houve a necessidade de efectuar a divisão deste valor por um factor superior a 4. Assim, e considerando a característica das memórias FIFO utilizadas que exige que o ponteiro de escrita se encontre adiantado do ponteiro de leitura de pelo menos 600

posições, optou-se por fazer este factor de divisão igual a 1024. Apresenta-se, na Figura 2.26, o sistema desenvolvido para controlo de *buffer*.

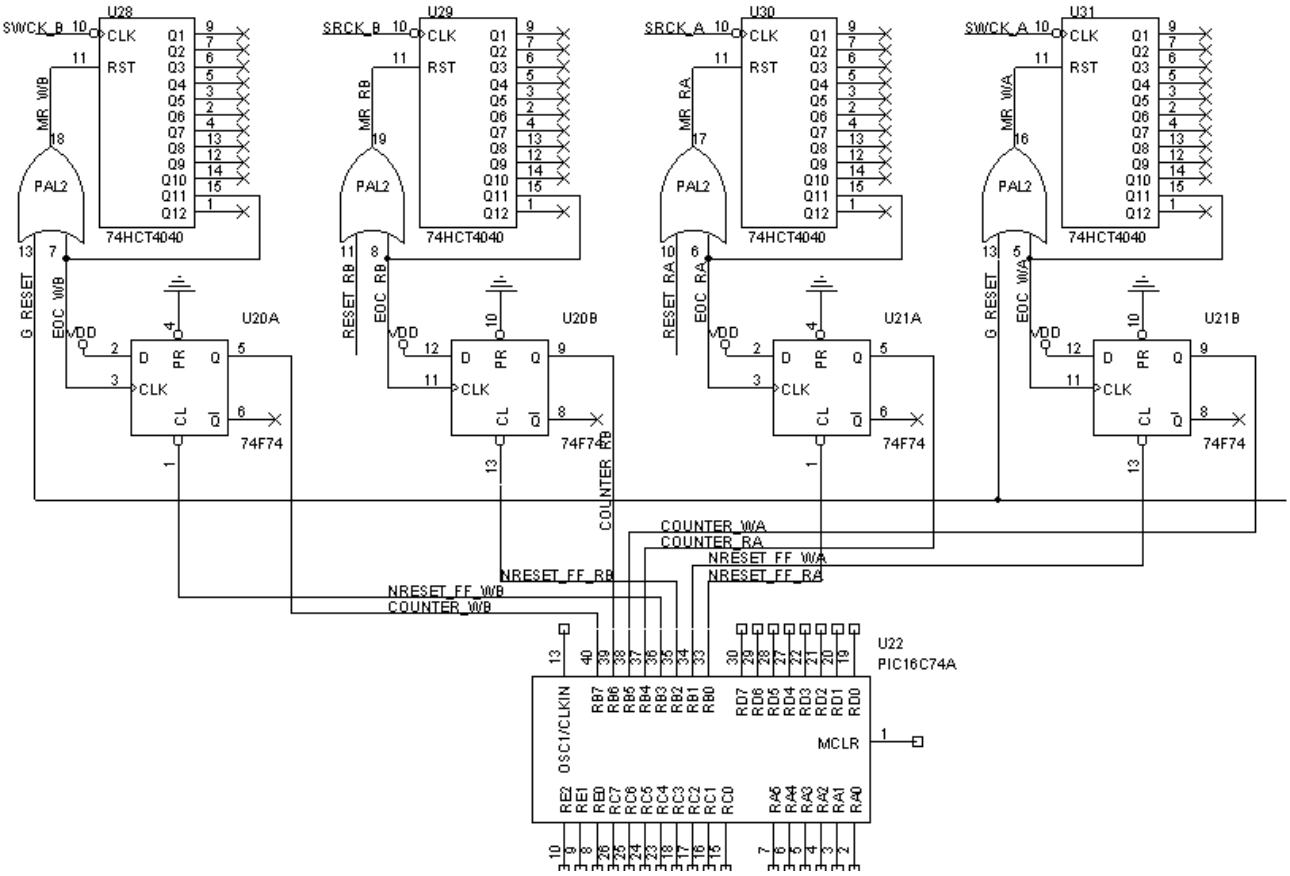


Figura 2.26 - Diagrama dos contadores do nível de ocupação dos *buffers* de emissão.

Assim, sempre que ocorre a escrita de uma palavra num dos *buffers*, é incrementado o contador de 10 bits correspondente, até atingir o valor de fim de contagem (1024 escritas). Nessa situação, o contador deverá ser inicializado a zero (através da activação da entrada de Reset) e é activada uma das entradas de interrupção do microcontrolador. De igual modo, utilizou-se um circuito idêntico para contar o número de leituras de cada um dos *buffers*, sinalizando-se ao microcontrolador a leitura de 1024 palavras de cada *buffer*.

Assim, sempre que um destes contadores atinge o fim de contagem, é desencadeada uma interrupção. Contudo, como não existe nenhum registo de controlo do microcontrolador que indique qual das entradas de interrupção foi activada, é necessário garantir que o valor na entrada é mantido até que a rotina de tratamento de interrupção possa verificar qual das quatro situações deu origem à interrupção. Para garantir um valor estável, optou-se por utilizar um *flip-flop* tipo D entre cada um dos contadores e o microcontrolador, activado pelo impulso desencadeado no fim da contagem, e que efectua a reinicialização do contador.

A rotina de tratamento das interrupções mantém dois contadores internos que, consoante a entrada de sinalização activada, são incrementados ou decrementados. Os valores correntes destes contadores são mantidos em registos de 16 bits, acessíveis a cada um dos DSPs. Por fim, esta rotina faz a reposição a zero do *flip-flop* correspondente, por forma a permitir a sua activação numa próxima situação de fim de contagem do contador de 10 bits respectivo.

Desta forma, sempre que o programa de um dos DSPs efectuar a leitura do nível de enchimento do *buffer* respectivo, far-se-á a activação da saída do registo de 16 bits correspondente através da

activação de um dos sinais '`OE_A`' ou '`OE_B`'. Estes sinais são gerados através dos sinais de $\overline{RD}$ dos DSPs e do sinal '`OUT01`' e '`OUT11`' proveniente da FPGA2 (correspondente à descodificação do endereço '`0055h`'), através das seguintes expressões:

$$OE\_A = \overline{OUT01 \cdot \overline{RD_A}} \quad ; \quad OE\_B = \overline{OUT11 \cdot \overline{RD_B}} \tag{25}$$

Para finalizar, refere-se ainda que é da responsabilidade do microcontrolador proceder à inicialização de todo o sistema descrito. Assim, após se activar a entrada '`MCLR`' deste microcontrolador, este deverá proceder à inicialização dos vários componentes com os seus valores ou estados iniciais, assim como à programação dos vários registos do circuito integrado Q1900, responsável pelo bloco de Codificação de Fonte, e à inicialização da FPGA2, responsável pelo bloco de formatação das amostras recolhidas.

2.2 CODIFICAÇÃO DE FONTE E MODULAÇÃO

Este bloco é constituído por três etapas distintas. Na primeira, é feita a codificação de fonte dos dados recebidos do bloco de codificação, por forma a aumentar o nível de protecção dos dados digitais contra erros de transmissão. Na etapa seguinte, o sinal digital é aplicado a um modulador GMSK com uma portadora centrada em 90 kHz [17]. Por fim, o sinal modulado é aplicado à etapa de *interface* de linha que tem, como função, a amplificação deste sinal e a sua introdução na linha de transmissão eléctrica.

O bloco de codificação de fonte é responsável pela codificação dos símbolos a enviar pelo sistema de transmissão. O seu objectivo é a introdução de uma componente de redundância estruturada que, em caso de ocorrência de um número de bits errados no bloco de recepção, permite, dentro de certos limites, recuperar a informação original.

O algoritmo de codificação utilizado foi o algoritmo de Viterbi. As razões que levaram a escolher este algoritmo prendem-se com a sua crescente divulgação no mercado das telecomunicações devido, nomeadamente, às suas características de eficácia e eficiência, tanto em termos de largura de banda como em termos da potência transmitida. Além disso, optou-se por utilizar este algoritmo com uma taxa de codificação de 3/4, significando que são enviados 4 símbolos codificados por cada 3 bits úteis que se pretendem transmitir. Na realidade, no caso de se desejarem níveis de imunidade ao ruído superiores, poder-se-iam ter utilizado outras taxas de codificação (tal como 1/2, 1/3, …) que, ao introduzirem níveis de redundância mais elevados, permitem fazer a correcção de um número de bits superior. Contudo, tal viria a degradar o ritmo efectivo de transmissão da informação, pelo que houve que encontrar uma solução de compromisso que melhor se adequa à implementação requerida. O compromisso encontrado baseou-se, como se referiu, na taxa de codificação de 3/4.

O bloco que realiza a codificação de fonte foi realizado através de um circuito integrado do tipo Q1900 da Qualcomm [18] e cuja descrição detalhada se encontra registada no Apêndice B. O diagrama deste bloco encontra-se representado na Figura 2.27.

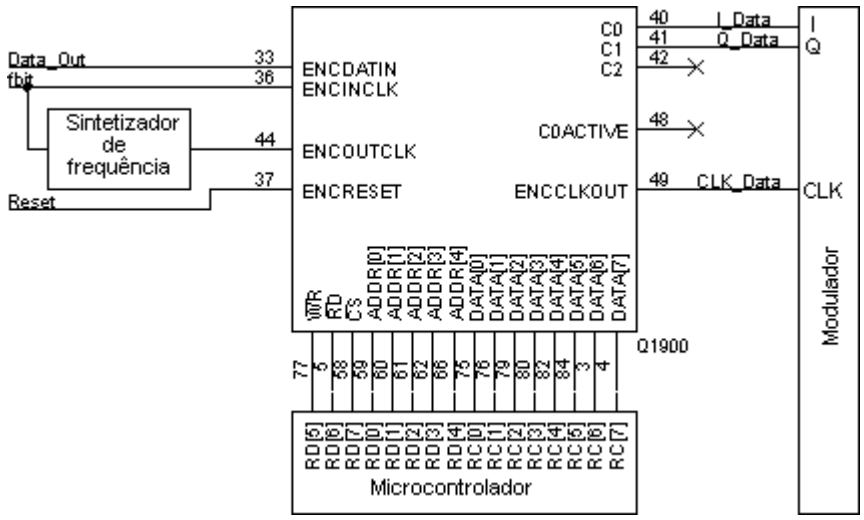


Figura 2.27 - Diagrama do bloco do codificador de fonte.

Este circuito integrado recebe, como sinais de entrada, os sinais `Data_Out` e `fbit`, provenientes do bloco de serialização de trama, respectivamente nas entradas `ENCDATIN` e `ENCINCLK`. O sinal `fbit` é também aplicado na entrada de um sintetizador de frequência, por forma a obter um sinal com uma frequência igual a 2/3 da frequência de `fbit`.

Assim, em cada 3 impulsos do relógio `fbit`, este circuito receberá 3 bits de informação na entrada `ENCDATIN`, e devolverá nas suas saídas `C0` e `C1` 2 pares de símbolos codificados no flanco ascendente do relógio `Clk_Data`, presente na saída de `ENCCLKOUT`. Como se pode verificar, por cada 3 ciclos do relógio `fbit`, deverão existir dois ciclos do relógio `ENCOUTCLK`. Daí a necessidade de gerar um sinal com uma frequência igual a 2/3 do sinal `fbit`.

A geração deste sinal foi feita por intermédio de um sintetizador de frequência do tipo 74HCT4046, conforme se ilustra na Figura 2.28.

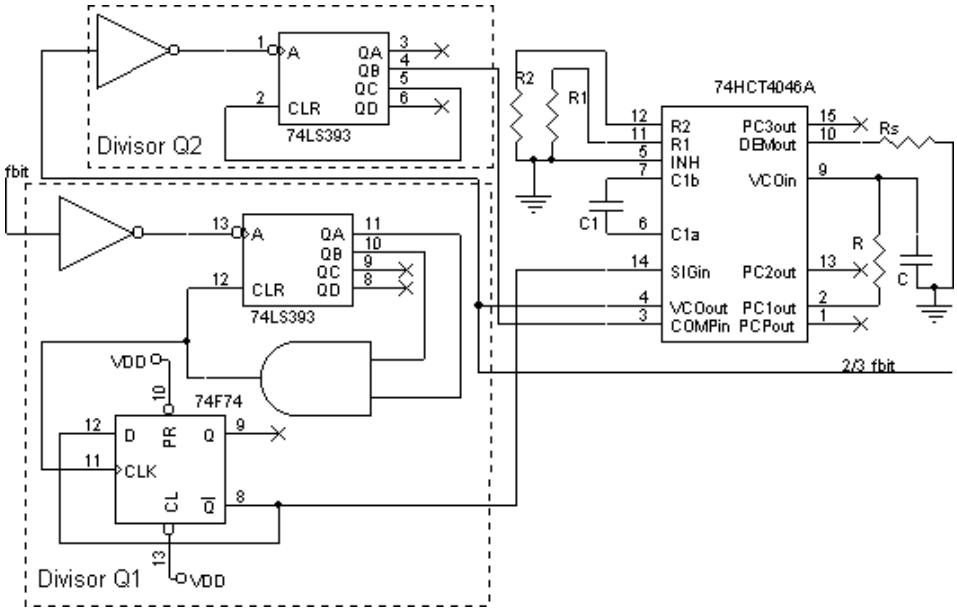


Figura 2.28 - Diagrama do sintetizador de frequência utilizado.

Assim, na situação de sincronismo, a frequência do sinal à saída do sintetizador será dada por:

$$f_{ENCOUTCLK} = \frac{Q_2}{Q_1} \cdot f_{ENCINCLK} \quad (26)$$

À primeira vista, os valores de Q_2 e Q_1 necessários para obter uma relação entre as frequências destes dois sinais de 2/3 seria de Q_2=2 e Q_1=3. Contudo, como o sintetizador utilizado apresenta um funcionamento ideal quando os sinais à entrada do detector de fase apresentam um factor de ciclo de 50%, e visto que esta condição não é garantida com o uso de um simples divisor de frequência por 3 na entrada de referência, optou-se pela introdução de um *flip-flop* na saída do divisor Q_1, por forma a garantir um factor de ciclo de 50%. No entanto, visto que este *flip-flop* vem introduzir uma divisão de frequência por dois, conclui-se que o factor Q_1 tem de assumir um valor par. Daí que, por forma a atingir a relação desejada, os valores de Q_1 e Q_2 deverão ser de Q_1=4 e Q_2=6.

O sinal na saída `ENCCLKOUT` terá uma frequência idêntica à do sinal à saída deste sintetizador de frequência.

A programação dos registos internos do circuito integrado Q1900 é feita por intermédio de um microcontrolador do tipo PIC16C74A, através dos seus barramentos de dados e endereços, e das suas linhas de controlo ($\overline{WR}$, $\overline{RD}$ e $\overline{CS}$).

Como se referiu anteriormente, no Apêndice B apresenta-se uma descrição mais detalhada do funcionamento deste circuito integrado, tanto no modo de funcionamento como codificador como no modo de funcionamento como descodificador.

3 Sistema de Recepção

3.1 DESMODULAÇÃO E DESCODIFICAÇÃO DE FONTE

O bloco de desmodulação tem uma constituição recíproca do bloco de modulação do emissor sendo, também, constituído por três etapas. Na primeira, denominada de *interface* de linha, é feita a extracção do sinal modulado da rede de distribuição de energia, englobando processos de filtragem e amplificação. A etapa seguinte é constituída por um desmodulador GMSK, onde é feita a desmodulação do sinal modulado por uma portadora de 100 kHz. Na última etapa, o sinal desmodulado é aplicado a um descodificador de fonte obtendo-se, à sua saída, a trama H.263.

O bloco de descodificação de fonte é responsável pela descodificação dos símbolos recebidos do desmodulador, codificados segundo o algoritmo de Viterbi pelo bloco de codificação de fonte do Sistema de Emissão.

Tal como o codificador de Viterbi, o descodificador adoptado utiliza uma taxa de codificação de 3/4. O diagrama deste bloco apresenta-se representado na Figura 3.1.

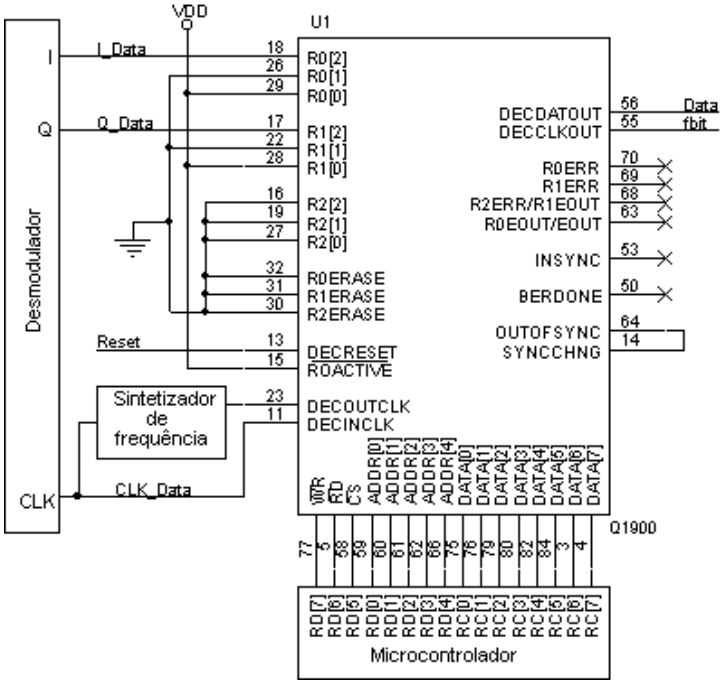


Figura 3.1 - Diagrama do bloco de descodificação de fonte.

Este bloco é também constituído por um circuito integrado do tipo Q1900 da Qualcomm, que recebe, como entradas, os sinais `I` e `Q` do desmodulador GMSK, assim como o relógio de símbolo que é aplicado à entrada '`DECINCLK`'. Este sinal de relógio é também inserido na entrada de um sintetizador de frequência, por forma a obter um sinal com uma frequência igual a 3/2 da frequência do sinal `CLK_Data` a introduzir na entrada '`DECOUTCLK`' do circuito integrado Q1900.

Assim, em cada 2 impulsos do relógio `Clk_Data`, este circuito receberá dois pares de valores `I` e `Q` e devolverá, na sua saída, 3 bits descodificados no flanco ascendente do relógio `Fbit`, presente na saída `DECCLKOUT`. Como se pode verificar, por cada 2 ciclos de relógio `Clk_Data`, deverão existir 3 ciclos do relógio `Fbit`. Daí a necessidade de gerar um sinal com uma frequência igual a 3/2 do sinal `Clk_Data`.

A geração deste sinal foi feita por intermédio de um sintetizador de frequência do tipo 74HCT4046, conforme se ilustra na Figura 3.2. Este sintetizador apresenta uma estrutura muito semelhante da utilizada no bloco de codificação, apresentando apenas valores diferentes de coeficientes de divisão Q_1 e Q_2.

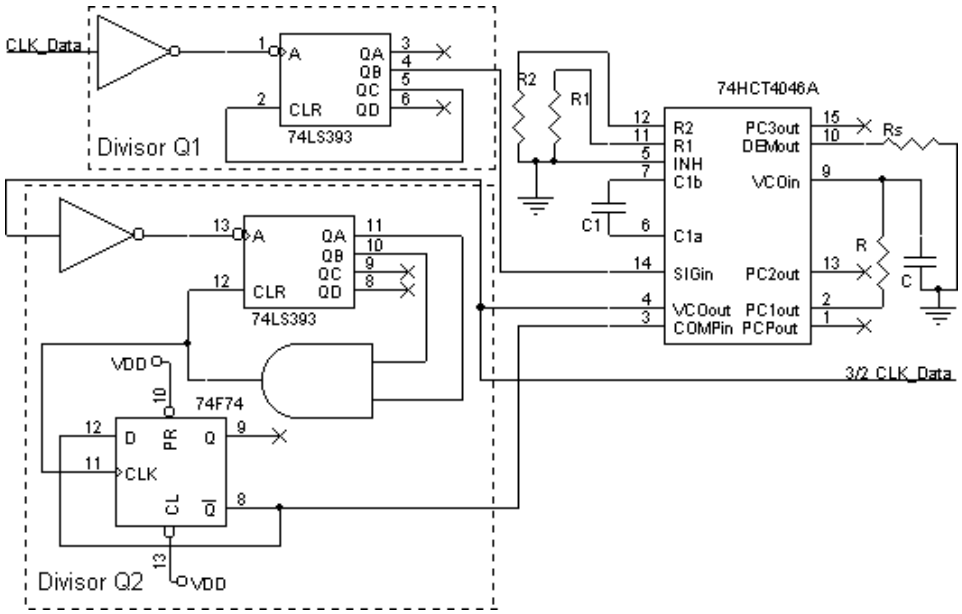


Figura 3.2 - Diagrama do sintetizador de frequência utilizado.

O sinal na saída DECCLKOUT terá uma frequência idêntica à do sinal à saída deste sintetizador de frequência.

Tal como no bloco de codificação, a programação dos registos internos do circuito integrado Q1900 é feita por intermédio de um microcontrolador do tipo PIC16C74A, através dos seus barramentos de dados e endereços e das suas linhas de controlo ($\overline{WR}$, $\overline{RD}$ e $\overline{CS}$).

No Apêndice B apresenta-se uma descrição mais detalhada do funcionamento deste circuito integrado, tanto no modo de configuração como codificador como no modo de configuração como descodificador.

3.2 MÓDULO DE DESCODIFICAÇÃO

O módulo de descodificação apresenta uma configuração recíproca da do módulo de codificação do sistema emissor, anteriormente descrita. Este módulo é, também, constituído por várias sub-módulos. Assim, o primeiro sub-módulo recebe a trama H.263 do módulo de desmodulação e procede à sua conversão série-paralelo. De seguida, os dados são entregues ao sub-módulo de descodificação H.263 que, por sua vez, entrega o sinal de vídeo digital ao sub-módulo de formatação. Este último sub-módulo constitui uma *interface* com o conversor digital-analógico (D/A) que apresenta, por fim, na sua saída, o sinal de vídeo analógico no formato PAL/NTSC (ver Figura 3.3).

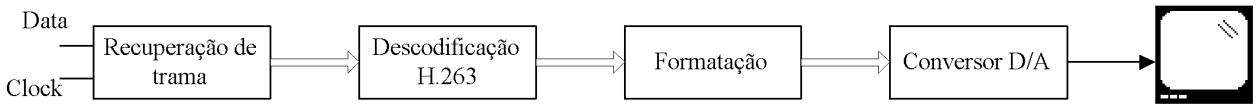


Figura 3.3 - Módulo de descodificação.

Tal como no caso do codificador, a transferência de dados entre o DSP e o sistema de visualização é realizada por um mecanismo de DMA, por forma a torná-la o mais rápida possível. A arquitectura do sistema implementado encontra-se representada na Figura 3.4.

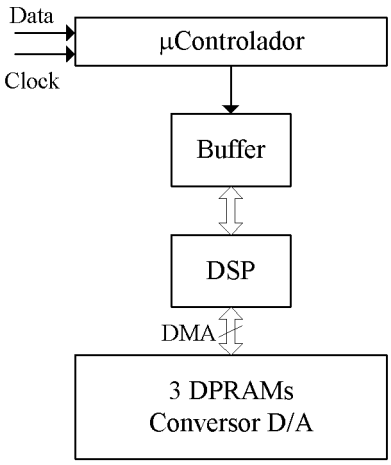


Figura 3.4 - Arquitectura do sistema de descodificação.

Nas secções seguintes, descreve-se, detalhadamente, cada um dos sub-módulos referidos na Figura 3.3.

3.2.1 Controlo de "buffer" e recuperação de trama

Tal como se referiu na secção anterior, o descodificador de fonte tem como saída dois sinais: Sinal de Dados (`Data`) e Relógio de Dados (`Fbit`). Estes sinais são colocados à entrada de um conversor série-paralelo, que realiza as seguintes funções (ver Figura 3.5):

- Alinhamento ao byte da trama recebida;
- Conversão série-paralelo da trama recebida;
- Armazenamento das palavras recebidas na memória FIFO do descodificador H.263.

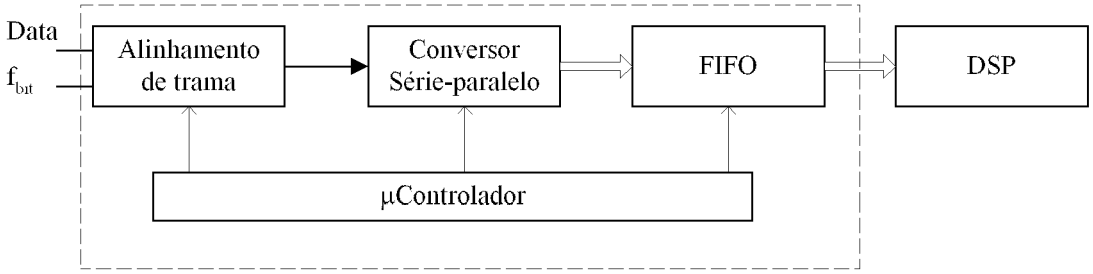


Figura 3.5 - Estrutura geral do conversor Série-Paralelo.

O controlo global de todas estas funções é efectuado por intermédio de um microcontrolador do tipo PIC16C74A. Apresenta-se, no Apêndice H, o esquema geral do circuito implementado.

A função de alinhamento ao nível do byte da trama recebida tem como objectivo facilitar a descodificação H.263 pelo DSP. Na realidade, este alinhamento encontra-se já associado à própria norma, visto que a mesma prevê a inclusão de bits de "*stuffing*", a fim de permitir o alinhamento ao byte de alguns dos cabeçalhos emitidos. Assim, por forma a se atingir este objectivo, implementou-se uma máquina de estados de sincronização, cuja principal missão é encontrar, na

trama de bits recebida, a sequência de início de uma nova imagem - PSC[17], constituída pela seguinte sequência de bits:

PSC (22 bits): '`0000 0000 0000 0000 1000 00`'

Na Figura 3.6, apresenta-se o diagrama da máquina de estados implementada.

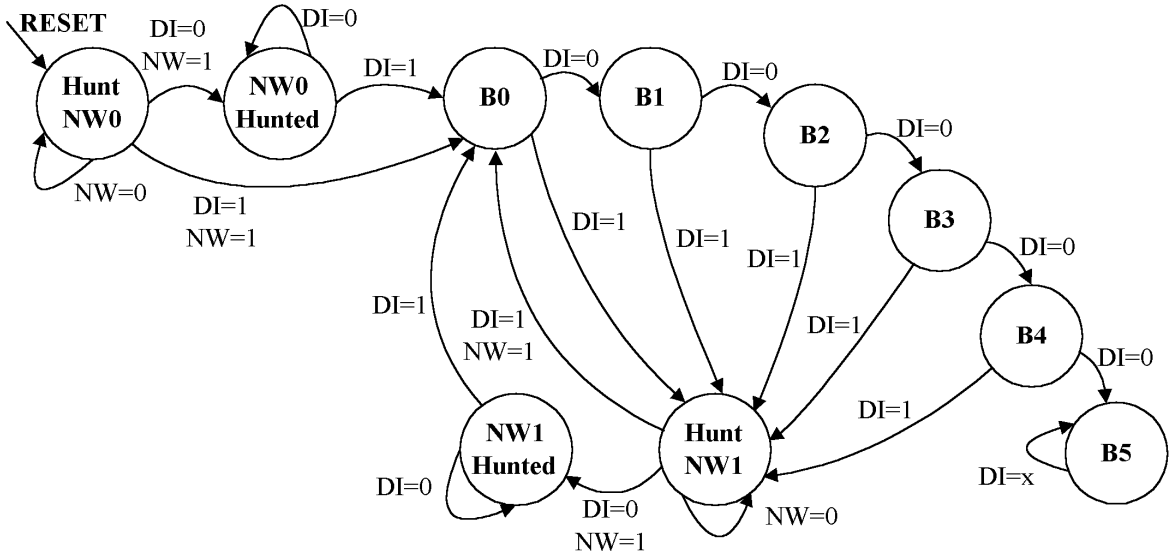


Figura 3.6 - Diagrama de estados da máquina de alinhamento de trama.

Os sinais de entrada desta máquina são:

`Din` - Sinal correspondente aos bits de dados de entrada (`Data`);

`NW` - Sinal indicador da detecção de uma palavra de 16 bits igual a '`0000h`', conforme se explicará adiante;

`Reset` - Inicialização da máquina de estados.

Conforme se observa no diagrama da Figura 3.6, a máquina ficará, após a inicialização, num ciclo de procura da sequência PSC, a que correspondem os estados: `Hunt NW0`, `NW0 Hunted`, `Hunt NW1`, `NW1 Hunted`, `B0`, `B1`, `B2`, `B3` e `B4`. A partir do momento em que a sequência PSC é detectada, a máquina permanecerá no estado `B5`. De notar que, no projecto deste mecanismo de sincronização, se previu situações de ocorrência de múltiplos bits a '`0`' antes da sequência PSC através dos estados `NW0 Hunted` e `NW1 Hunted`. O facto de se terem duplicado os estados `Hunt NW0` e `NW0 Hunted` relaciona-se com o mecanismo de escrita na memória FIFO, por forma a evitar escritas múltiplas da palavra '`0000h`' no ciclo de procura inicial. De facto, esta escrita na memória da palavra '`0000h`' correspondente à PSC apenas deve ocorrer no estado `Hunt NW0`. A máquina de estados foi implementada numa PAL (`pal1`) do tipo PALCE22V10, cujo código de programação é apresentado no Apêndice J.4.

Como se referiu acima, a segunda função deste bloco consiste na conversão série-paralelo da trama recebida. Tal foi conseguido por intermédio de um registo de deslocamento de 16 bits (correspondente ao comprimento de palavra do DSP utilizado), e cujo diagrama é apresentado na Figura 3.7.

[17] Do inglês Picture Start Code.

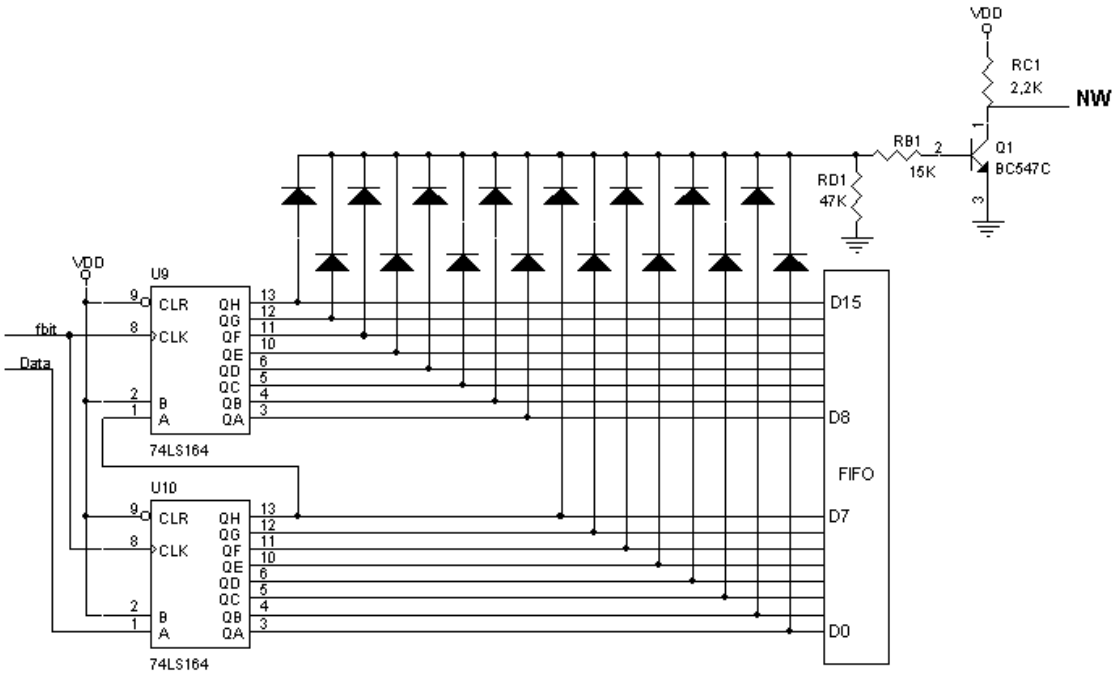


Figura 3.7 - Diagrama do conversor série-paralelo.

Este registo é constituído por dois circuitos integrados do tipo 74LS164, sendo cada um deles um registo de deslocamento de 8 bits.

Tal como se pode observar na Figura 3.7, introduziu-se um conjunto de díodos na saída deste registo. Este circuito implementa a função lógica NOR de todos os bits de dados na sua saída: quando o sinal à saída do registo de deslocamento apresentar todos os bits com o nível lógico zero, nenhum dos díodos estará em condução, o transístor encontrar-se-á cortado, fazendo com que `NW`='1'. Basta que um dos bits de dados se encontre no nível lógico '`1`' para que o díodo correspondente entre em condução, dando assim origem a uma corrente de base no transístor suficiente para o levar à saturação, fazendo assim com que `NW`='0'. Assim, o nível lógico de `NW` indicará a presença, ou não, de uma cadeia de 16 bits a zero, necessária ao funcionamento de máquina de estados anteriormente descrita.

A decisão de se implementar esta função lógica por intermédio deste circuito prende-se com a inexistência de circuitos integrados com portas NOR com um número tão elevado de entradas, o que obrigaria ao uso de um conjunto de portas lógicas ligadas em cadeia, implicando uma incomportável ocupação de espaço no circuito implementado. Para além disso, como a frequência de bit é razoavelmente baixa (da ordem da dezena de kbit/s), verificou-se que existe tempo suficiente para retirar o transístor da saturação, mantendo um correcto funcionamento do circuito.

A partir do instante em que a máquina de estados se sincroniza com a PSC dá-se início ao ciclo de escrita na memória FIFO. Dado o comprimento da palavra de 16 bits, esta escrita decorrerá a um ritmo 16 vezes inferior à frequência de bit. Este mecanismo foi realizado por intermédio de um contador de 4 bits, implementado também na `pal1`.

Tal como no bloco emissor, a memória FIFO utilizada foi realizada com 4 circuitos integrados do tipo TMS4C1050. Desta forma, o *buffer* de recepção apresentará uma capacidade total de 256k posições de 16 bits cada.

Tal como se referiu aquando da descrição do sistema de emissão, as memórias utilizadas apresentam algumas restrições no modo de funcionamento, nomeadamente no que se refere às temporizações dos sinais de controlo que são utilizados para o refrescamento interno da memória.

O FIFO de recepção encontra-se ligado ao DSP, através de *buffers* com saídas em três estados (do tipo SN74F245 [21]). Desta forma, é possível manter a saída destes circuitos em alta impedância durante o tempo em que o DSP não se encontra a ler dados do FIFO. O controlo da abertura do *buffer* foi implementado através do sinal '`OE_BUF245`', obtido através da seguinte expressão:

$$OE\_245 = \overline{OUT0} \tag{27}$$

em que `OUT0` é um sinal de saída da `FPGA3`, correspondente à descodificação do endereço de I/O `0054h`. O facto de não se ter utilizado o sinal de $\overline{RD}$ do DSP para o controlo da saída do *buffer* prende-se com a necessidade de efectuar a leitura o mais rapidamente possível. De facto, de acordo com a opção tomada, não é necessário esperar que o sinal $\overline{RD}$ tome o valor lógico '`0`' para que a saída do *buffer* saia do estado de alta impedância, podendo fazê-lo logo que o DSP gera este endereço. Esta opção poderia, no entanto, trazer algumas complicações caso o DSP efectuasse também escritas para este endereço, situação em que tanto o *buffer* como o barramento de dados do DSP tentariam impor os seus níveis lógicos. Como tal nunca se verifica durante a execução do programa realizado, conclui-se que a opção tomada não apresenta qualquer tipo de problema.

Por forma a que o DSP possa, em qualquer instante, avaliar o nível de enchimento do *buffer* de recepção, implementou-se, tal como no sistema emissor, um contador de palavras guardadas em memória, por intermédio de contadores do tipo 74HCT4040 e de um microcontrolador do tipo PIC16C74A.

Uma vez mais, dado que a capacidade do *buffer* (256 kwords) implica o uso de 18 bits para contar o número de palavras em memória, houve a necessidade de dividir este valor por 1024, facilitando assim também a gestão dos ponteiros de escrita e de leitura da memória FIFO (ver secção 2.1.4). Na Figura 3.8 apresenta-se o circuito de controlo de *buffer* implementado.

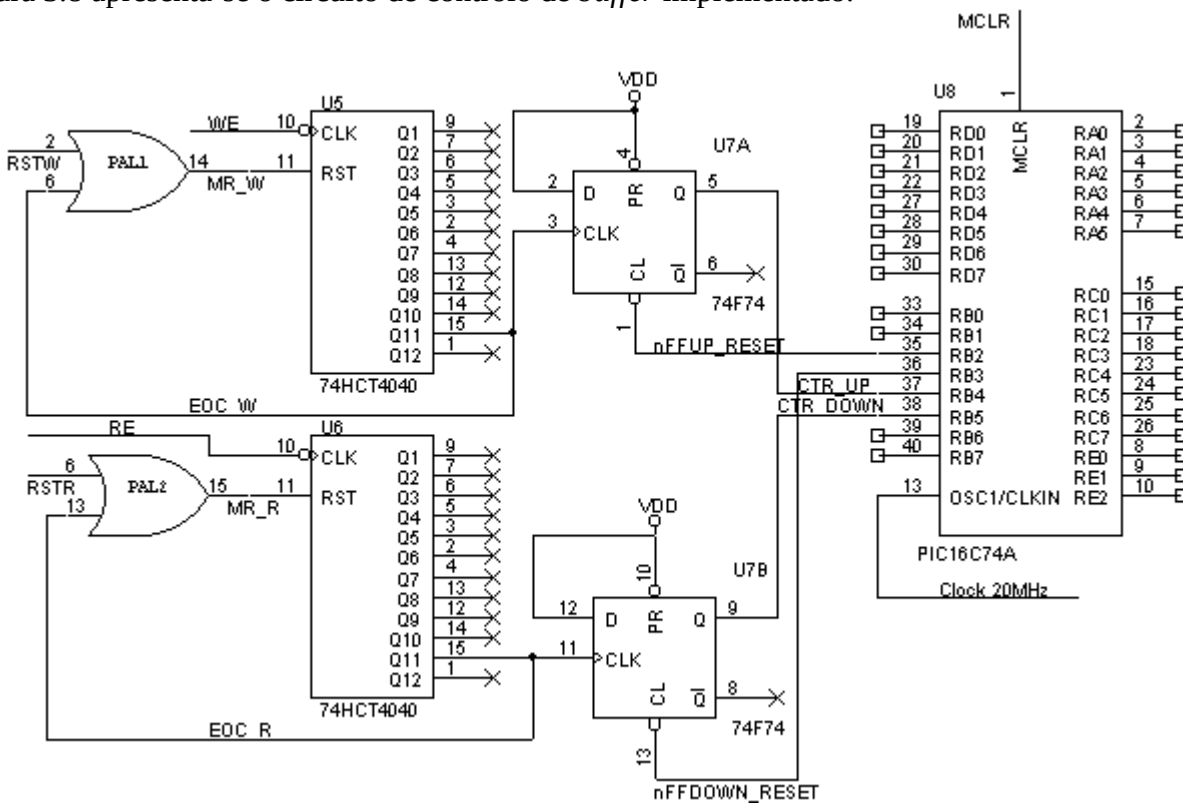


Figura 3.8 - Diagrama do contador do nível de ocupação do *buffer* de recepção.

Como se pode constatar, este circuito apresenta uma estrutura muito semelhante à do circuito utilizado no módulo de emissão, baseado na activação das entradas de interrupção do microcontrolador.

Tal como no sistema de emissor, o microcontrolador mantém, também, o valor actual do contador interno num registo de 16 bits, acessível ao DSP. Desta forma, sempre que o programa do DSP efectuar a leitura do nível de enchimento do *buffer*, far-se-á a activação da saída deste registo de 16 bits através da activação do sinal '`OE_REG`'. Este sinal é gerado através do sinal de $\overline{RD}$ do DSP e do sinal '`OUT1`' proveniente da FPGA3 (correspondente à descodificação do endereço '`0055h`'), através da seguinte expressão:

$$OE\_REG = \overline{OUT1 \cdot \overline{RD}} \tag{28}$$

Para finalizar, refere-se ainda que é da responsabilidade deste microcontrolador proceder à inicialização de todo o sistema descrito. Assim, após se activar a entrada '`MCLR`' do microcontrolador, este deverá proceder à inicialização dos vários componentes com os seus valores ou estados iniciais, assim como à programação dos vários registos do circuito integrado Q1900, responsável pelo bloco de descodificação de fonte, e à inicialização da FPGA3, responsável pelo bloco de restauro do sinal de vídeo em modo *raster*.

3.2.2 Descodificação H.263

Ao contrário da codificação de vídeo, a descodificação H.263 foi realizada utilizando apenas um processador digital de sinal. Tal deve-se ao facto de o algoritmo a implementar apresentar características de processamento menos exigentes do que o algoritmo de codificação, tal como se pode constatar a partir da descrição da norma no Apêndice A. Na Figura 3.9 apresenta-se o diagrama de blocos do descodificador.

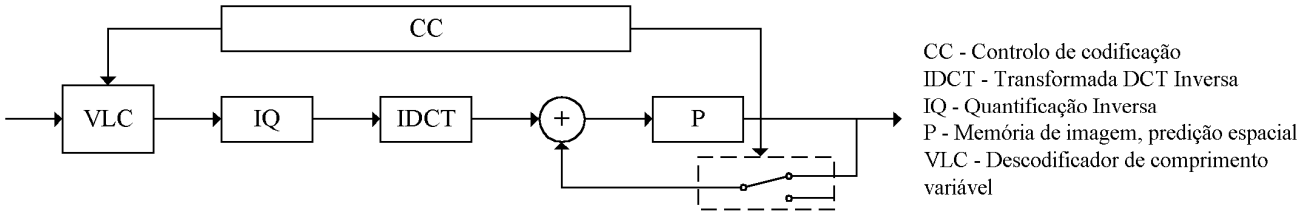


Figura 3.9 - Diagrama de blocos do descodificador.

Em seguida, passar-se-á a descrever os vários módulos do programa realizado. Durante a sua execução, houve o cuidado em organizar os ficheiros segundo uma estrutura hierárquica, por forma a manter uma organização dos vários blocos executados. Esta estrutura encontra-se ilustrada na Figura 3.10.

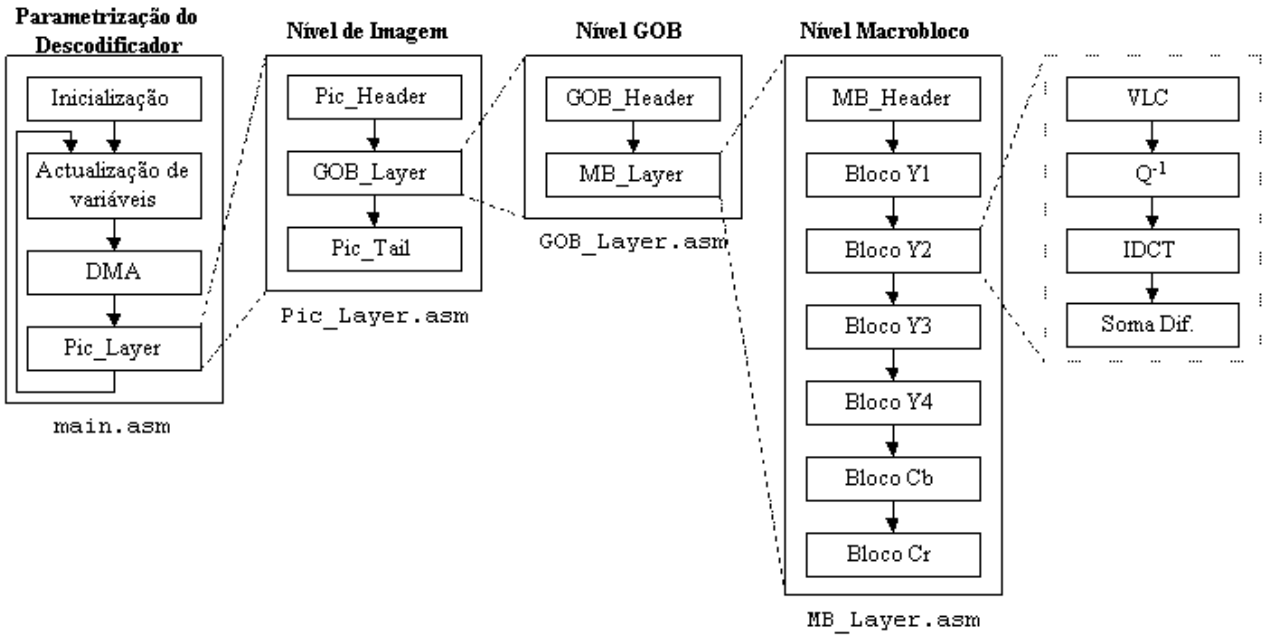


Figura 3.10 - Estrutura hierárquica dos vários módulos do programa do descodificador H.263.

3.2.2.1 Módulo *parametrização do descodificador*

Tal como no módulo de codificação, o módulo de parametrização do descodificador, que corresponde ao ficheiro "`main.asm`" (cuja listagem se apresenta no Apêndice D.2), ocupa o topo da hierarquia referida. Nele, o DSP começa por invocar as várias rotinas que inicializam alguns blocos do descodificador, tais como o bloco descodificador dos códigos de comprimento variável ("`vlc_ini`") e o bloco da transformada inversa IDCT ("`tranini`"). De seguida, é realizado o preenchimento dos registos da FPGA3, que determinarão a forma como se irá processar a transferência dos dados da memória do DSP para o módulo de visualização, através de um ciclo de DMA.

Durante esta transferência, o co-processador implementado pela FPGA3 fará a leitura da imagem resultante da descodificação, a partir do endereço `2C00h`, escrevendo os *pixels* respectivos em cada uma das memórias de duplo porto (DPRAMs[18]) do módulo de visualização, tal como se descreve nas secções seguintes. Após terminada a descodificação de uma imagem, o processador começa a descodificação da imagem seguinte.

Tal como foi feito na secção dedicada ao codificador, passar-se-á a descrever cada um dos módulos que constituem a hierarquia anteriormente referida.

3.2.2.2 Módulo *nível de imagem*

Este módulo, cuja listagem se apresenta no Apêndice D.3, começa por proceder à leitura dos vários campos que constituem o cabeçalho da *picture* a descodificar, conforme descrito na norma H.263 referida no Apêndice A. A leitura destes bits é realizada por intermédio de duas funções distintas: `GetBits`, que permite a leitura de um número arbitrário de bits dentro do intervalo [1,15] e `GetWord`, que permite a leitura de uma palavra de 16 bits. Ambas as funções retornam o número de bits requeridos no acumulador e numa posição de memória, denominada de "`Temp_VLC`", alinhados à direita.

[18] Do inglês *Dual Port RAM*.

Nesta fase, o programa identifica as características da imagem a descodificar, tais como o valor da variável `Pic_InP`, que caracteriza o modo em que a imagem foi codificada (INTRA ou INTER) e o passo de quantificação utilizado.

Terminada a leitura destes campos, correspondentes ao cabeçalho do nível de *picture*, o programa invoca o módulo *nível GOB* 9 vezes, correspondente ao número de GOBs existentes numa imagem.

Por fim, o programa efectua a leitura dos campos correspondentes ao "*tail*" da *picture*, onde se inclui a sequência de fim de *picture* – `EOS`.

3.2.2.3 Módulo *nível GOB*

Este módulo, cujo código se apresenta no Apêndice D.4, começa por efectuar a leitura dos campos do cabeçalho do nível GOB, cuja estrutura se encontra descrita no Apêndice A.2. De entre os campos lidos, inclui-se o valor do passo de quantificação a utilizar ao longo desse GOB (campo `GQUANT`).

Por fim, o programa invoca o módulo *nível macrobloco* 11 vezes, correspondente a cada um dos macroblocos que constituem o GOB.

3.2.2.4 Módulo *nível macrobloco*

Este módulo, cujo código se apresenta no Apêndice D.5, implementa o conjunto de operações necessárias à descodificação de cada um dos macroblocos. Assim, começa por efectuar a leitura dos campos do cabeçalho deste nível, onde se incluem os campos MCBPC e CBPY. A informação correspondente a estes campos irá ser utilizada para o preenchimento de uma variável, denominada de `CBPCx` que, à semelhança do que acontece no módulo de codificação, irá ser utilizada para reduzir o tempo de cálculo da transformada inversa – IDCT e de descodificação dos coeficientes VLC.

De seguida, o programa inicia o processamento de cada um dos blocos correspondentes a esse macrobloco: Y1, Y2, Y3, Y4, C_B e C_R. Para cada um destes blocos, começa por fazer a chamada à função VLC, cujo código se apresenta no Apêndice D.8. Esta função realiza a leitura do código correspondente a cada um dos coeficientes e a sua escrita no bloco segundo a sequência em *zig-zag* referida na norma (ver Figura A.5). Para tal, foram utilizadas as tabelas de códigos de comprimento variável definidas na recomendação.

Finalizada a descodificação dos coeficientes, o programa procede à sua desquantificação, de acordo com o valor do passo de quantificação lido anteriormente. Seguidamente, o programa procede ao cálculo da transformada de coseno inversa – IDCT, cujo código se encontra apresentado no Apêndice D.7, utilizando o mesmo algoritmo utilizado pelo módulo de codificação do sistema emissor.

Por fim, e terminado o cálculo da IDCT, o programa executará os seguintes procedimentos, dependendo de se tratar de uma imagem INTRA ou INTER:

- No caso de a imagem ser do tipo INTRA, o programa efectua uma transferência dos *pixels* correspondentes ao macrobloco descodificado para o espaço de memória reservado à imagem corrente (a partir do endereço `2C00h`);
- No caso de a imagem ser do tipo INTER, o programa efectua a soma do valor dos coeficientes decodificados com os *pixels* correspondentes à imagem anterior, presentes em

memória a partir do endereço `2C00h`. Os resultados dessa soma serão guardados, novamente, na memória correspondente à imagem corrente.

Como se referiu anteriormente, e à semelhança do módulo de codificação, a informação presente na variável `CBPCx` correspondente ao macrobloco a descodificar pode, em geral, tornar o processamento do cálculo da IDCT e da descodificação dos códigos VLC mais rápido, conforme se descreveu na secção 2.1.3.4

3.2.2.5 Funções auxiliares do programa

No Apêndice A, apresenta-se um conjunto de funções auxiliares que, embora necessários à execução da descodificação, não contêm código correspondente a qualquer tipo de processamento. É o caso, por exemplo, do ficheiro "`Mem_org.h`" onde se estruturou e organizou a memória de dados utilizada pelo programa.

3.2.3 Restauro do sinal de vídeo em modo "Raster"

Tal como no módulo de codificação do sistema de emissão, também aqui foi necessário realizar uma *interface* entre o DSP e o conversor D/A de vídeo, por forma a transferir os *pixels* armazenados na memória do DSP (resultante da descodificação H.263) para o conversor. Esta *interface* converte a representação da imagem estruturada em blocos, característica da norma H.263, para um formato de *raster* (uma sequência de *pixels* ordenados por linha) utilizado pelo conversor.

Para atingir este objectivo, foi desenvolvido um circuito cujo esquema é apresentado no Apêndice F.1. Como se pode observar neste esquema, foram utilizadas 3 DPRAMs de 32 kBytes do tipo IDT7007L [19]. Cada uma destas memórias armazena as amostras recolhidas da memória do DSP, sendo que uma contém as amostras correspondentes à luminância Y (176 x 144 = 25344 amostras) e as outras duas contêm, respectivamente, amostras de crominância C_B e C_R (88 x 72 = 6336 amostras). Para os dois últimos casos, não seria necessário utilizar memórias de 32 kBytes (8 kBytes seriam suficientes), prendendo-se a sua utilização a questões de disponibilidade de aquisição. A razão pela qual se usaram memórias de duplo porto tem a ver com o facto de se ter de fornecer continuamente ao conversor D/A as amostras de vídeo, enquanto que é necessário, simultaneamente, efectuar a actualização da imagem quando o DSP conclui a sua descodificação.

A formatação dos dados requer, novamente, o uso de EPROMs para armazenar os endereços que convertem o formato dos blocos de H.263 para a sequência recebida pelo conversor D/A. Para o efeito foram usadas 3 EPROMs de 8 kBytes, dado serem necessários 12 bits para endereçar os *pixels* Y e 10 bits para endereçar os *pixels* C_B e os *pixels* C_R. Os endereços para a DPRAM que contem os *pixels* Y estão divididos pela `EPROM Y` (que contem os 8 bits menos significativos desses endereços) e pela `EPROM YCBCR` (que contem os restantes 4 bits mais significativos dos endereços, no seu *nibble* menos significativo). De forma semelhante, os endereços para as DPRAM que guardam os *pixels* CB e CR estão contidos na `EPROM CBCR` (6 bits mais significativos) e na `EPROM YCBCR` (4 bits menos significativos, no *nibble* mais significativo).

Para a leitura das amostras síncrona com o relógio de *pixel* fornecido ao conversor D/A, foi utilizado um registo na saída de cada uma das DPRAMs. Como se pode constatar no esquema, é possível entregar ao conversor, a qualquer momento, um sinal monocromático bastando, para tal, colocar a linha `COR` no nível lógico zero. Dessa forma, as amostras de C_B e C_R tomarão o valor de 128 (correspondente à não existência de cor, independentemente das amostras que tiverem presentes à estrada dos registos).

Foi também implementado um oscilador para gerar o relógio necessário tanto ao funcionamento da FPGA (descrita mais à frente), como à conversão D/A das amostras de vídeo. Este oscilador, para uma frequência de 20 MHz, foi implementado com base num cristal de 20 MHz.

Por último, foi usada uma FPGA (`FPGA3`) que implementa as seguintes funções:

- Transferência através de DMA dos dados existentes na memória do DSP para as 3 DPRAMs (efectuando a sua separação segundo as componentes Y, C_B e C_R);
- Controlo da transferência dos *pixels* armazenados nas 3 DPRAMs para o registo associado ao conversor D/A;
- Implementação dos divisores necessários ao funcionamento do sintetizador de frequência;
- Descodificação de 4 endereços para permitir o acesso do DSP ao circuito do *buffer* de recepção descrito na secção 3.2.1.

Apresenta-se, em seguida, uma descrição mais pormenorizada dos circuitos que realizam cada uma destas funções, que foram descritos utilizando a linguagem VHDL.

3.2.3.1 Transferência por DMA

Após o DSP descodificar uma imagem, é necessário retirar imediatamente os *pixels* correspondentes da memória do processador, de forma a que o DSP possa prosseguir com a descodificação da imagem seguinte. A solução encontrada para tornar esta transferência o mais rápida possível, foi a utilização de um esquema DMA, tal como no caso do módulo de codificação (ver secção 2.1).

Com o objectivo de tornar o sistema flexível foram, novamente, implementados um conjunto de registos de 16 bits que permitem ao DSP configurar o co-processador de visualização. Na Tabela 3.1 apresentam-se os registos e a sua localização em relação ao endereço base `FPGA3_IO_ADD`, que constitui um dos barramentos de entrada da `FPGA3`.

Registo	**Offset**
`Limite`	`0050h`
`Origem`	`0051h`
`Cor`	`0052h`
`Reset`	`0053h`

Tabela 3.1 - Endereços dos registos de configuração do co-processador de visualização.

O registo `Origem` deve ser programado pelo DSP com o valor respeitante à posição de memória a partir da qual se encontram os *pixels* correspondentes à imagem descodificada (os 6 bits menos significativos deste registo são ignorados).

O registo `COR` define se se está ou não na presença de uma imagem policromática, sendo que essa definição é conseguida através do valor do bit menos significativo (bit a '`1`' significa que a imagem é policromática).

O registo *Reset* permite, ao processador, efectuar a inicialização da `FPGA3`, implicando a inicialização do circuito responsável pela transferência por DMA e do circuito de transferência dos *pixels* das DPRAMs para o conversor D/A. Para efectuar a inicialização, o processador apenas tem que escrever para o endereço correspondente ao registo de `Reset`. Na presente implementação, o valor que é escrito neste registo é ignorado.

O registo `Limite` é o registo que identifica o endereço de memória que corresponde ao último elemento da imagem descodificada. No entanto, este registo é também responsável pelo desencadeamento da transferência por DMA. Esta transferência é iniciada logo que o processador escreva um valor diferente de zero neste registo. Por forma a detectar a ocorrência de algum erro na transferência dos dados, este registo é colocado a '`0`' no final de uma transferência, permitindo ao processador tomar as acções necessárias caso, ao ler o referido registo, este apresente um valor diferente de '`0`'. Estas acções podem passar, por exemplo, pela inicialização da `FPGA3` e pelo desencadear de uma nova transferência.

Para controlar a transferência DMA, foi implementada uma máquina de estados cujo diagrama de estados é apresentado na Figura 3.11.

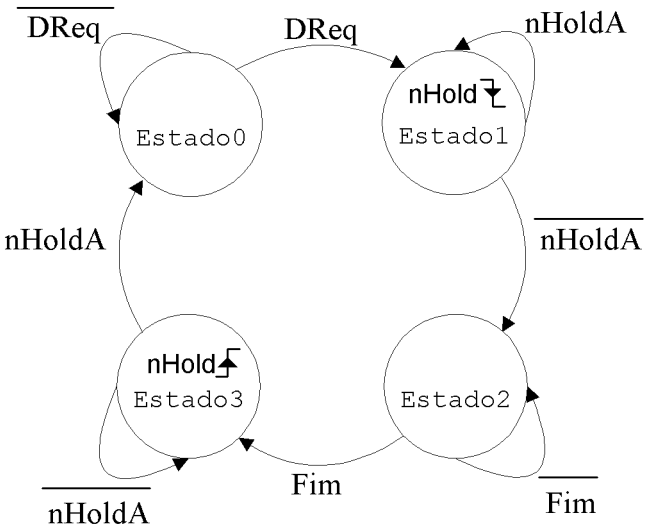


Figura 3.11 - Diagrama de estados do controlador de DMA.

Tal como se pode observar, esta máquina é constituída por 4 estados. Inicialmente, o sistema encontra-se no `Estado0` à espera do desencadeamento, por parte do processador, da transferência DMA. O sinal `DReq` é, na realidade, o resultado do 'ou' lógico dos bits que constituem o registo `Limite`. Após a escrita de um valor diferente de zero no referido registo, a máquina de estados transita para o `Estado1`, onde é activado o sinal de $\overline{\text{HOLD}}$ do DSP (`nHold`) assinalando ao processador a intenção de efectuar uma transferência. Aguarda, então, no `Estado1` que o processador active o sinal $\overline{\text{HOLDA}}$ (`nHoldA`). Após essa activação, a máquina transita para o `Estado2` onde se efectua a transferência por DMA. O término desta transferência é assinalado pela variável `Fim`, que é activada quando se atinge o endereço correspondente ao valor do registo `Limite`, que indica a última posição de memória a ser transferida. Após a conclusão da transferência, a máquina transita para o `Estado3`, onde é desactivado o sinal $\overline{\text{HOLD}}$ e se espera pela desactivação, por parte do DSP, do sinal $\overline{\text{HOLDA}}$.

Para além da lógica associada à máquina de estados, foi necessário acrescentar um circuito adicional, que protege o sistema contra acontecimentos inesperados que possam provocar a colocação das saídas de todos os *flip-flops* da máquina de estados a '`0`' lógico, provocando a sua paragem. Desta forma, sempre que for detectada essa anomalia, é efectuada a inicialização do sistema responsável pela transferência DMA. Apresenta-se, na Figura 3.12, o esquema da máquina de estados onde se inclui a detecção da anomalia referida.

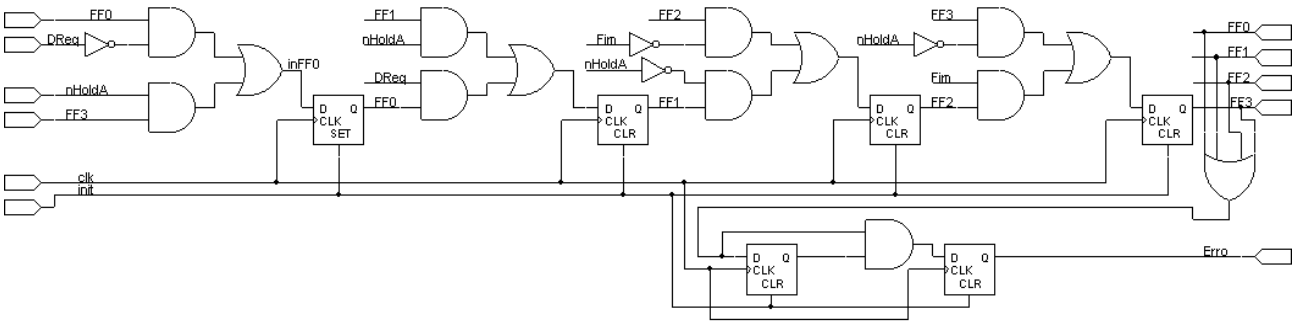


Figura 3.12 - Circuito lógico da máquina de estados do controlador de DMA.

A descrição VHDL desta máquina é apresentada no Apêndice F.2.3.7. Na Figura 3.13, apresenta-se o circuito que, durante o `Estado2` da máquina de estados acima referida, efectua a passagem dos *pixels* da memória do DSP para as DPRAMs.

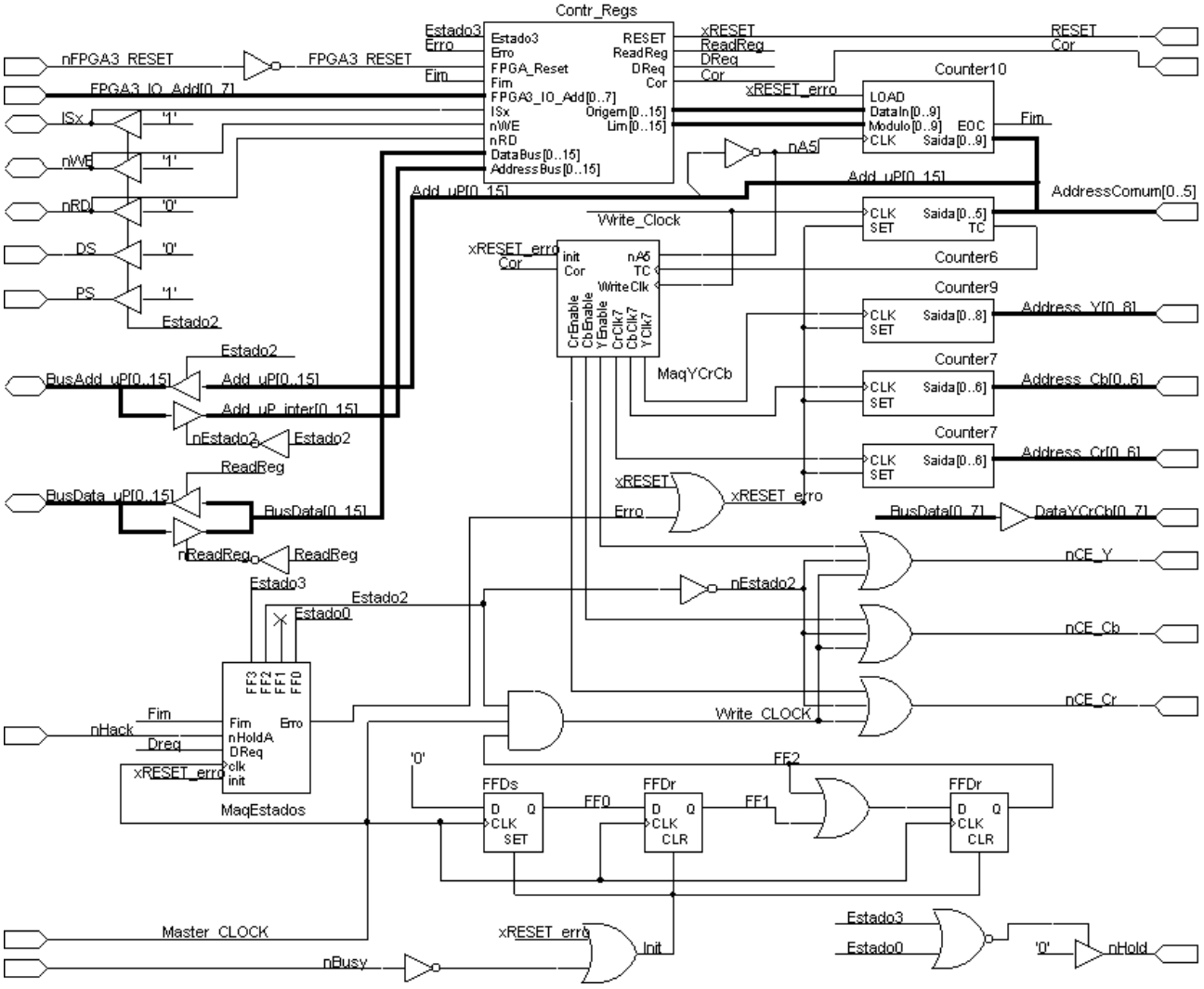


Figura 3.13 - Circuito de transferência por DMA.

Este circuito efectua a leitura da memória do processador desde o endereço especificado no registo `Origem` até ao endereço especificado no registo `Limite`. O sinal `Fim` referido na máquina de estados corresponde ao sinal de fim de contagem do contador de 10 bits. Este contador de 10 bits (cuja descrição VHDL é apresentada no Apêndice F.2.3.12), é inicializado com o valor presente no registo `Origem` no início do `Estado2` e gera os 10 bits mais significativos do barramento de

endereços de DSP. Os 6 bits menos significativos são gerados pelo contador de 6 bits (descrição VHDL em Apêndice F.2.3.15). Um ciclo completo deste contador de 6 bits corresponde, na realidade, à leitura de um bloco H.263 de 64 *pixels* (6 bits ≡ 64 posições).

Caso se tivesse optado por transferir directamente a imagem da memória do DSP para uma memória externa, a máquina de estados e estes dois contadores seriam suficientes para o efeito. No entanto, dado se ter optado por efectuar, logo durante a transferência DMA, a separação dos blocos Y, C_B e C_R, foi necessário incluir a restante lógica que procede à selecção da DPRAM em que se devem escrever os dados lidos e à geração dos endereços para as DPRAMs.

O elemento central que efectua a selecção entre as 3 DPRAMs é a máquina de estados identificada como `MaqYCrCb`. Esta máquina (cuja descrição VHDL é apresentada no Apêndice F.2.3.8), tem seis estados distintos correspondendo cada um deles a um dos blocos de um macrobloco, isto é: existem quatro estados correspondentes aos 4 blocos Y, um estado correspondente ao bloco C_B e um último correspondente ao bloco C_R. Através destes seis estados, foi possível gerar os sinais de controlo de escrita para as DPRAMs (escrita controlada por $\overline{CE}$) e três sinais de relógio para a actualização do endereço de escrita das memórias correspondentes a cada uma das componentes. Na Figura 3.14, apresenta-se o esquema lógico da máquina de estados.

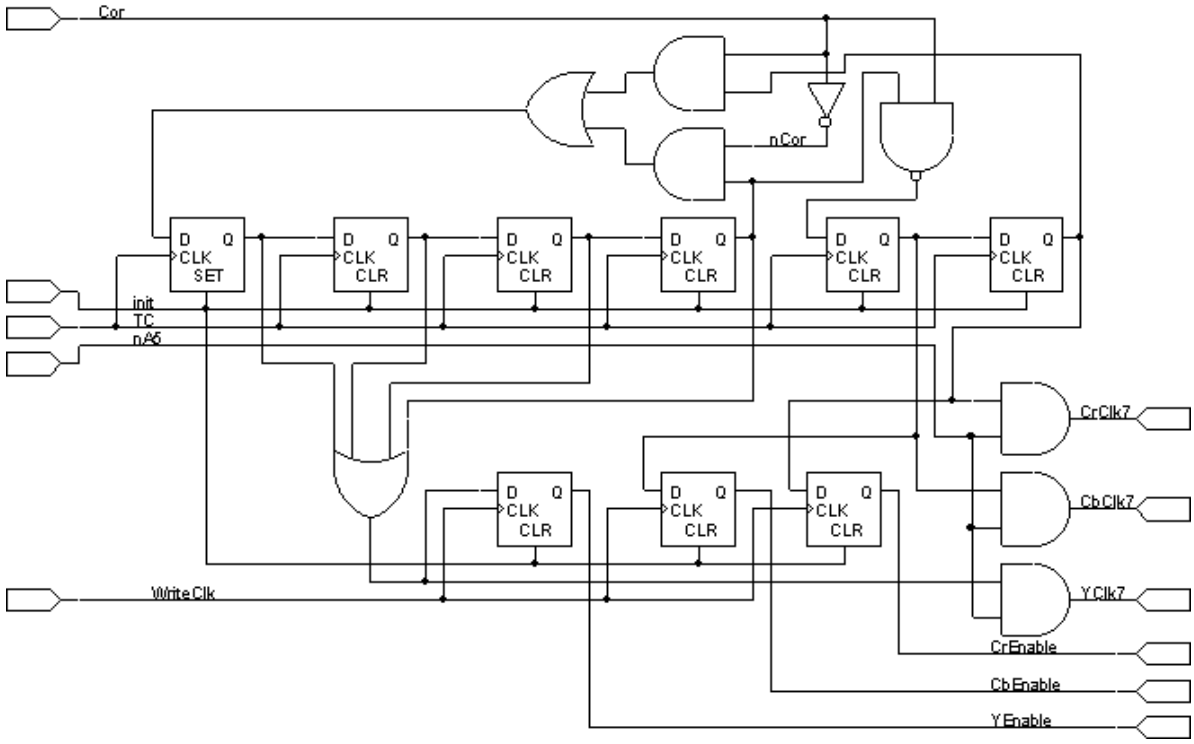


Figura 3.14 - Esquema lógico da máquina de estados `MaqYCrCb`.

Como se pode observar, na Figura 3.14, o funcionamento da máquina de estados depende do valor programado no registo `Cor`. No modo monocromático existem apenas 4 estados correspondentes aos 4 blocos Y.

Os endereços de escrita nas DPRAMs são, então, gerados por 3 contadores independentes (um por DPRAM). Este contadores geram os bits mais significativos do endereço, que são concatenados com os bits de saída do contador de 6 bits já descrito. A descrição VHDL destes contadores é apresentada no Apêndice F.2.3.16 e no Apêndice F.2.3.18.

A escrita nas DPRAMs exige, no entanto, que se tenha em atenção o facto de não ser possível escrever numa posição de memória que está a ser acedida pelo outro porto. De facto, é este o único

caso de contenção que pode ocorrer durante o funcionamento deste sistema, já que um porto é apenas usado para escrita (porto L) enquanto que o outro é apenas usado para leitura (porto R). Para que esta situação não causasse a perda de um ou mais *pixels* da imagem a ser escrita, foi implementado um mecanismo que efectua a suspensão temporária da escrita nas DPRAMs até que a linha $\overline{BUSY}$ (que assinala contenção nas DPRAMs) seja desactivada pela(s) memória(s).

O sistema de transferência completo encontra-se descrito no Apêndice F.2.3.2 e consiste na interligação dos vários blocos descritos, que englobam todas as funcionalidades necessárias para efectuar a transferência por DMA. No Apêndice F.2.1.1 é apresentada a descrição dos portos de entrada e saída deste sistema.

3.2.3.2 Transferência da imagem para o codificador de sinal de vídeo PAL

Para codificar a informação de vídeo num sinal que segue a norma PAL [5], é necessário efectuar a conversão D/A e proceder ao condicionamento do sinal de vídeo. As 3 componentes dos *pixels* da imagem armazenadas nas DPRAMs, são transferidas para o conversor D/A por uma ordem diferente daquela em que estão armazenadas na DPRAM e na memória do DSP. Para realizar a conversão de formato realizou-se o circuito apresentado na Figura 3.15 para converter a representação de imagem para o formato aceite pelo conversor D/A.

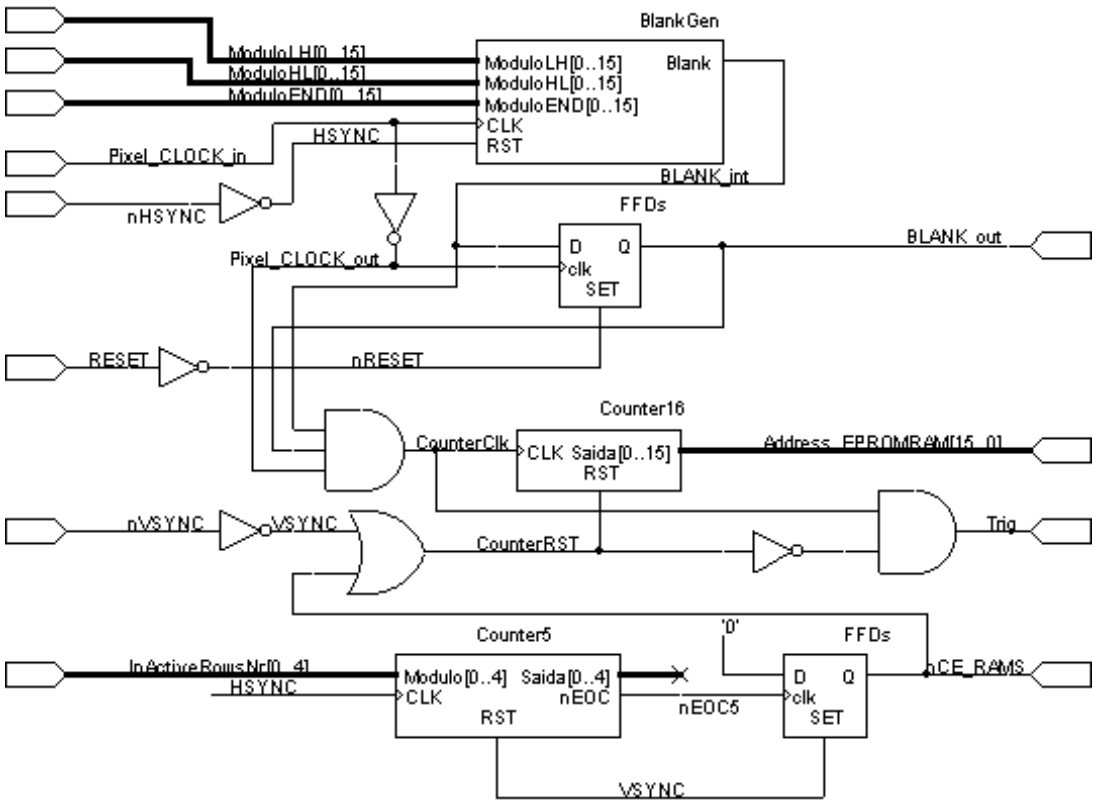


Figura 3.15 - Circuito de transferência da imagem para o conversor D/A.

O bloco referido como '`BlankGen`' gera o sinal '`Blank`', síncrono com o sinal de sincronismo horizontal `HSync` fornecido pelo codificador PAL. O sinal '`Blank`' indica ao conversor quais as amostras correspondentes a *pixels* que constituem a imagem. Respeitando a norma PAL, o sinal '`Blank`' é periódico com período de 64 µs e um factor de ciclo de 52/64. A descrição VHDL do módulo `BlankGen` encontra-se no Apêndice F.2.3.6.

O contador de 16 bits presente na Figura 3.15 (e cuja descrição VHDL se encontra no Apêndice F.2.3.13), é responsável pela geração dos endereços para as EPROMs. Dado que cada linha de um bloco H.263 corresponde a 8 *pixels* que também se encontram em linha na imagem, os últimos 3 bits deste contador podem ser usados para aceder directamente às memórias. Como tal, os endereços armazenados nas EPROMs correspondem ao endereço inicial de cada linha de um bloco H.263, perfazendo um total de $\frac{176\times144}{64}\times8=3168$ endereços diferentes para a componente Y e 792 endereços diferentes para as componentes C_B e C_R. No entanto, como é necessário que cada linha de *pixels* seja repetida, o número total de endereços gerado por uma EPROM será de 6336.

O contador de 5 bits presente no circuito (e cuja descrição VHDL se encontra no Apêndice F.2.3.14) encarrega-se de efectuar a contagem do número de linhas correspondente ao retorno vertical. Este circuito contador mantém-se inactivo enquanto não for desactivado o sinal $\overline{\text{VSYNC}}$ (nível lógico '1'), depois de assumir o nível '0' durante o tempo correspondente a 1 *pixel*. Este contador só permite, então, que o restante circuito comece a funcionar após se ter efectuado o retorno vertical, cuja duração é indicada pelo conversor através da activação, por 22 vezes, do sinal de sincronismo horizontal $\overline{\text{HSYNC}}$.

Como se refere na secção 3.2.4, a frequência de *pixel* do conversor tem de ter um valor quádruplo da frequência de *pixel* usada neste bloco. Esta característica poderia trazer problemas ao correcto funcionamento do circuito, pois o período de *pixel* do conversor não seria suficiente para realizar, em série, a geração do endereço para as EPROMs, aceder às EPROMs (tempo de acesso de 120 ns) e aceder às DPRAMs (tempo de acesso de 20 ns). De facto, o período de *pixel* presente no conversor é de cerca de 73,5 ns (f = 13,6 MHz), pelo que a realização em série das operações atrás referidas implicaria perder, no mínimo, 1 a 2 *pixels* por cada 8 numa imagem. A solução encontrada retirou partido do facto de a frequência de *pixel* no circuito ser 4 vezes inferior à do conversor, colocando-se registos à saída das DPRAMs que permitem aceder ao próximo *pixel* durante o tempo em que o conversor lê o *pixel* presente à saída dos registos. Os registos mencionados são comandados pelo sinal `TRIG`.

O circuito de transferência dos *pixels* para o conversor D/A encontra-se descrito em VHDL no Apêndice F.2.3.3. No Apêndice F.2.1.2 é apresentada a descrição dos portos de entradas e saída deste circuito.

3.2.3.3 Divisores do Sintetizador de Frequência de 13,6 MHz

Como já foi referido anteriormente, e se verá com mais detalhe na secção 3.2.4, é necessário fornecer um relógio de *pixel* para o conversor com uma frequência de 13,6 MHz. Para além deste relógio, é também necessário obter um outro relógio com uma frequência 4 vezes inferior, para ser utilizado na fase de transferência da imagem das DPRAMs para o conversor.

Para que se pudesse gerar um sinal com uma frequência de 13,6 MHz a partir do sinal proveniente do cristal de 20 MHz disponível, implementou-se um sintetizador de frequência baseado no circuito CMOS 74HCT4046A e em vários divisores que foram implementados na FPGA3. A frequência de 13,6 MHz pode ser obtida a partir da frequência de 20MHz através da operação:

$$13{,}6\ \text{MHz}=\frac{20\ \text{MHz}}{100}\times68 \qquad (29)$$

É suficiente, portanto, a implementação de um divisor por 100 e de um por 68. No entanto, dado ser necessário obter a frequência de 3,4 MHz ($\frac{13{,}6\ \text{MHz}}{4}$), a estrutura final do sintetizador implementado ficou a seguinte:

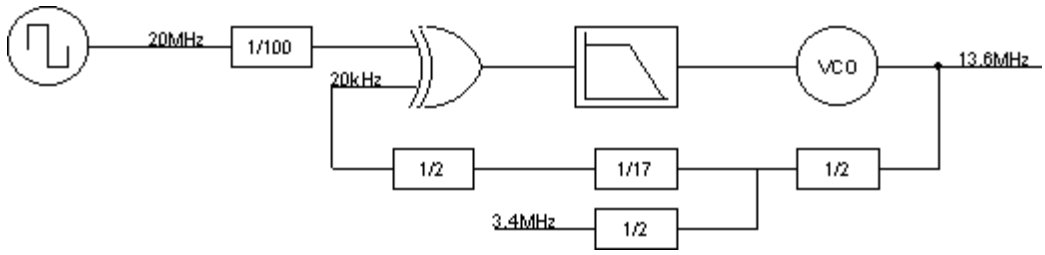


Figura 3.16 - Sintetizador de frequência.

A razão pela qual não se utiliza somente um único divisor por 4 e se optou por usar 2 divisores por 2 (dois na malha de realimentação e outro para gerar o relógio de 3,4 MHz) tem a ver com a necessidade de obter um sinal com um factor de ciclo de 50% em ambas entradas do detector de fase.

A descrição dos divisores em VHDL encontra-se descrita nos Apêndices F.2.3.11, F.2.3.10 e F.2.3.4. A descrição dos portos de entrada e saída do módulo que engloba todos os divisores é apresentada no Apêndice F.2.1.3.

3.2.3.4 Descodificação de endereços do DSP

Como se referiu na secção 3.2.1, é necessário que o DSP possa endereçar outros dispositivos externos para além dos registos que comandam o funcionamento da FPGA3. Para tal, foi implementado um descodificador de endereços adicional que permite o endereçamento de até 4 dispositivos externos. Os endereços atribuídos situam-se entre `0054h` e `0057h`. Desta forma, quando a FPGA3 detecta um acesso de I/O para estes endereços, é activada a saída correspondente de acordo com a Tabela 3.2:

Endereço	Saída activada
`0054h`	`OUT0`
`0055h`	`OUT1`
`0056h`	`OUT2`
`0057h`	`OUT3`

Tabela 3.2 - Descodificação dos endereços dos registos auxiliares do co-processador de visualização.

A descrição VHDL deste pequeno módulo encontra-se no Apêndice F.2.3.5, estando inserida na descrição do módulo encarregue de efectuar a transferência DMA, presente no Apêndice F.2.3.2, como se referiu anteriormente.

A integração de todas estas funcionalidades da FPGA3 encontra-se descrita em VHDL no ficheiro presente no Apêndice F.2.3.1.

3.2.4 Codificador de sinal de vídeo PAL

Para efectuar a conversão do sinal de vídeo digital no formato YC_RC_B para o sinal de vídeo composto (PAL/NTSC), foi utilizada uma placa previamente desenvolvida para esse efeito, projectada com base no circuito integrado Bt858 da Brooktree [20].

A versatilidade deste circuito integrado permite-lhe efectuar a conversão de inúmeros formatos digitais de imagem (RGB 24/16 bits, YC_RC_B 24/16 bits, …) tanto para vídeo no formato PAL como no formato NTSC. Por forma a permitir a integração desta placa de visualização no presente projecto, foi escolhido o formato de entrada "24 bit YC_RC_B", em que é necessário fornecer, à placa, 8 bits de cada uma das 3 componentes por cada ciclo do relógio de *pixel* do conversor. O formato de saída utilizado foi o vídeo composto PAL por razões que se prendem com os aparelhos disponíveis para a visualização da imagem.

A placa de codificação PAL permite escolher entre quatro modos diferentes de funcionamento, que diferem, basicamente, no número e tipo de sinais de sincronismo que é necessário fornecer ao circuito integrado. De entre os 4 modos disponíveis, optou-se por usar o modo em que é o codificador de PAL que gera os sinais de sincronismo horizontal $\overline{\text{HSYNC}}$ e vertical $\overline{\text{VSYNC}}$, tendo o circuito projectado que gerar o sinal '`Blank`' e fornecer o sinal de relógio de *pixel*. O sinal de sincronismo horizontal ($\overline{\text{HSYNC}}$) é activo a '`0`', durante o período de tempo correspondente a um *pixel*, após o contador horizontal do circuito atingir o valor `HCOUNT`. Quanto ao sinal de sincronismo vertical, esse é activado também durante o período de tempo correspondente a um *pixel*, mas após o contador vertical atingir o valor 625, correspondente ao número de linhas do formato de vídeo PAL.

Tendo definido os formatos de vídeo de entrada e saída, assim como o modo de funcionamento da placa de codificação PAL, foi apenas necessário efectuar os cálculos indicados pelo fabricante, por forma a programar os registos internos do circuito Bt858 e a definir a frequência de *pixel* correcta a fornecer ao conversor. Dentro dos registos de controlo que é necessário programar no conversor, destacam-se os registos `HCOUNT`, `P1` e `P2` que, em conjunto, definem o número de *pixels* por linha da imagem. Em seguida, apresenta-se as equações que dão conta da relação do valor destes registos com a frequência de *pixel* utilizada:

$$\text{HCOUNT} = (\text{int})\ (\text{FCLK\_DESIRED}/15625) \quad ; \quad 2<\texttt{HCOUNT}<4094 \text{ e } \texttt{HCOUNT} \text{ par} \tag{30}$$

$$\text{P1} = (\text{int})\ \frac{1135\times 2048}{4\times \text{HCOUNT}} \quad ; \quad 0<\texttt{P1}<1024 \tag{31}$$

$$\text{P2} = 1135\times 2048 - \text{P1}\times 4\times \text{HCOUNT} + 13 \quad ; \quad 0<\texttt{P2}<4096 \tag{32}$$

Como o formato QCIF comporta 176 *pixels* por linha, a frequência de *pixel* necessária corresponderia a:

$$\text{FCLK\_DESIRED} = \frac{176}{52\,\mu\text{s}} \approx 3{,}38\,\text{MHz} \tag{33}$$

em que 52 μs representa a duração do vídeo útil numa linha PAL. Com este valor para a frequência de *pixel*, obter-se-iam os seguintes valores para os 3 registos atrás mencionados:

$$\text{HCOUNT} = 216 \quad ; \quad \text{P1} = 2690 \quad ; \quad \text{P2} = 333 \tag{34}$$

Como se verifica, o valor obtido para o registo `P1` não se encontra dentro dos limites atrás referidos (registo de 10 bits). A solução adoptada foi de quadruplicar a frequência de *pixel*. Daí que resulta, então,

$$FCLK\_DESIRED \approx 13{,}54\ MHz \tag{35}$$

Esta frequência de relógio teve, no entanto, de ser gerada a partir do sinal de relógio de referência de 20 MHz disponível, através do sintetizador de frequência apresentado na secção 3.2.3.3. Como tal, a frequência de *pixel* final adoptada foi a seguinte:

$$FCLK\_DESIRED = \frac{20\,MHz}{100} \times 4 \times 17 = 13{,}6\,MHz \tag{36}$$

Com esta frequência de relógio, os 3 registos tomam os seguintes valores:

$$HCOUNT = 870 \quad ; \quad P1 = 667 \quad ; \quad P2 = 3333 \tag{37}$$

O circuito Bt858 possui um total de 14 registos. No Apêndice I.2 apresenta-se o valor com que cada um destes registos foi programado para o correcto funcionamento do codificador PAL.

Como se pode verificar na tabela referida, o valor do registo `HCOUNT` foi sujeito a ajustes experimentais que implicaram uma mudança também, dos registos `P1` e `P2`. Estes ajustes experimentais tiveram, como único objectivo, a melhoria subjectiva da qualidade da imagem de saída.

A programação do conversor ficou a cargo de um microcontrolador do tipo PIC16F84 da Microchip existente na placa de visualização apresentando-se, no Apêndice I.2, o programa desenvolvido para o efeito.

4 Resultados Obtidos

Nesta secção, apresentam-se alguns resultados experimentais obtidos ao longo deste trabalho. Separaram-se estes resultados em dois conjuntos: os que dizem respeito aos co-processadores implementados nas três FPGAs e os associados aos módulos de codificação e descodificação do sinal de vídeo.

4.1 CO-PROCESSADORES

No módulo de emissão, foi necessário utilizar duas FPGAs, devido, fundamentalmente, ao insuficiente número de pinos disponíveis em cada FPGA. De facto, tendo-se utilizado o encapsulamento PQFP[19] de 160 pinos, as FPGAs XC4010e disponibilizam apenas cerca de 130 desses pinos para entradas/saídas de uso geral. Os restantes pinos são reservados para alimentação e para configuração do dispositivo, não podendo ser utilizados após o circuito ter sido configurado. Como tal, foi necessário dividir o co-processador de aquisição implementado no emissor em duas FPGAs, já que o número de entradas/saídas necessário ascendia a cerca de 190. A partição dos circuitos pelas duas FPGAs teve, como principal critério, tornar os dois sub-circuitos o mais independentes possível. Um outro critério que poderia condicionar a separação da lógica implementada pelas duas FPGAs tem a ver com a ocupação dos recursos internos das próprias FPGAs.

Na Tabela 4.1, apresenta-se informação respeitante à utilização das três FPGAs.

	FPGA1		**FPGA2**		**FPGA3**		**XC4010e (PQ160)**
Número total de CLBs	35	(8,75%)	165	(41,25%)	122	(30,5%)	400
Número de '*Ports*'	68	(52,7%)	126	(97,7%)	122	(94,6%)	129
Número de '*Clock Pads*'	3	(37,5%)	1	(12,5%)	3	(37,5%)	8
Número de IOBs	65	(50,4%)	126	(97,7%)	119	(92,2%)	129
Número de *Flip Flops*	33	(2,9%)	108	(9,6%)	155	(13,8%)	1120
Número de *buffers tri-state*	0	(0,0%)	257	(29,2%)	102	(11,6%)	880
Freq. máx. de funcionamento (MHz)	32,143		7,384		25,146		

Tabela 4.1 - Estatística de ocupação das FPGAs utilizadas.

Outro resultado importante quando se utiliza este tipo de circuitos de lógica programável tem a ver com a frequência máxima de operação dos circuitos. Como se pode constatar, a frequência máxima de funcionamento das três FPGAs foi de 32,143 MHz, 7,384 MHz e 25,146 MHz para a FPGA1, FPGA2 e FPGA3, respectivamente. Esta frequências máximas são importantes na FPGA2 e na FPGA3, para que seja possível efectuar as transferências DMA o mais rapidamente possível. Na FPGA1, é necessário garantir que o circuito tem um funcionamento correcto à frequência de *pixel* do conversor A/D, que é cerca de 13,6MHz.

[19] Do inglês *Plastic Quad Flat Package*.

4.2 CODIFICAÇÃO H.263

Nesta secção apresentar-se-ão alguns resultados experimentais obtidos, respeitantes ao módulo de codificação de vídeo implementado. Estes resultados focam, essencialmente, os valores correspondentes ao tempo de processamento de alguns blocos que compõem o codificador (ver Figura A.2), nomeadamente, o bloco da transformada DCT e quantificação, o bloco de codificação variável dos coeficientes (VLC), o bloco de desquantificação (tanto no modo INTRA como no modo INTER) e o bloco da transformada inversa IDCT.

Foram recolhidos os tempos de processamento para o melhor e pior casos: o pior caso corresponde a todos os blocos de coeficientes da transformada não nulos e o melhor caso corresponde a ter todos os blocos de coeficientes da transformada nulos, excepto o primeiro coeficiente, correspondente ao valor DC das imagens INTRA.

Os valores obtidos apresentam-se na Tabela 4.2 e nas figuras seguintes (Figura 4.1 e Figura 4.2).

		Bloco completamente preenchido [μs]		Bloco só com coeficiente DC [μs]		Legenda
DCT+Quant.		118.8	(29.4%)	118.8	(92.6%)	
VLC		133.2	(33.0%)	1.8	(1.4%)	
InvQuant	**Intra**	49.4	(12.2%)	0.9	(0.7%)	
	Inter			1.1	---	
IDCT		102.7	(25.4%)	6.9	(5.3%)	
TOTAL (Intra)		**404.1**	**(100.0%)**	**128.4**	**(100.0%)**	

Tabela 4.2 - Tempo de processamento dos vários módulos do codificador.

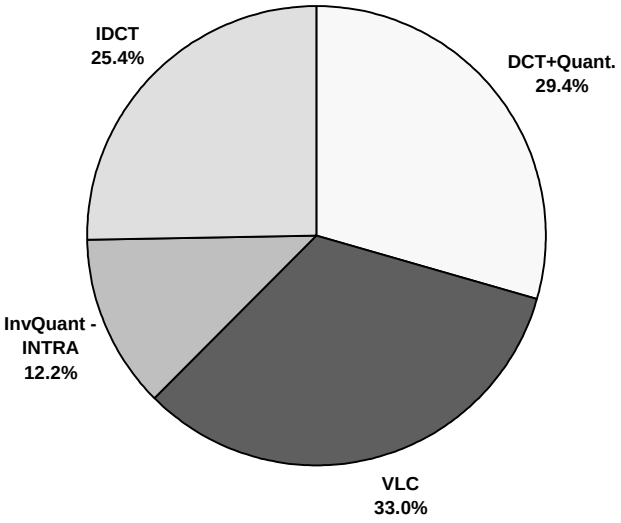


Figura 4.1 - Bloco completamente preenchido.

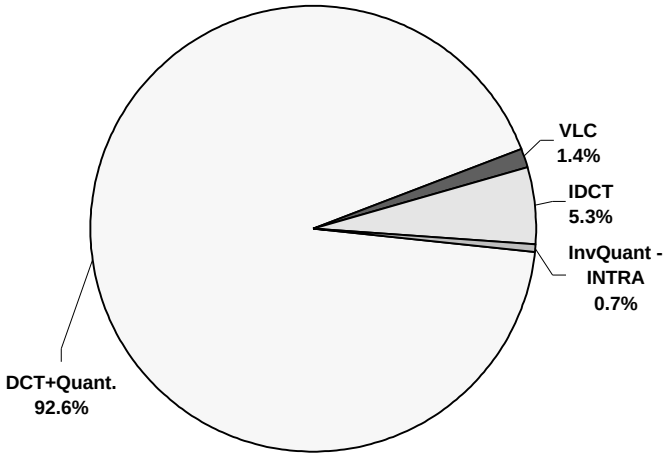


Figura 4.2 - Bloco só com coeficiente DC.

Como se pode verificar, o módulo que implementa a DCT apresenta um tempo constante em ambos os casos. Tal facto deve-se a ser este o único módulo que realiza processamento independente dos

dados, gerando a informação de qualificação dos macroblocos (tabela CBPC descrita na secção 2.1.3.2).

A partir dos resultados apresentados, é também possível verificar que, ao contrário do que era esperado, o bloco de codificação de comprimento variável dos coeficientes (VLC) apresenta-se, em certas circunstâncias, como aquele que acarreta um maior peso computacional (chegando mesmo a ultrapassar o tempo de processamento do cálculo da DCT). Esta situação deve-se, essencialmente, à natureza não sequencial em que os coeficientes de um dado bloco têm de ser analisados. De facto, seguindo uma sequência de processamento em *zig-zag* (ver Figura A.5), há a necessidade de efectuar inúmeras operações de indexação de memória e de, dependendo do coeficiente a processar ser nulo ou não, efectuar a chamada (condicional) à rotina de codificação VLC do coeficiente. Assim, verifica-se que, para efectuar este algoritmo utilizando um DSP, o mesmo dispenderá uma parte significativa do tempo a efectuar operações de controlo.

Em relação ao cálculo da IDCT, e tal como se previu na secção 2.1.3.4, o tempo de processamento pode assumir valores bastantes diferenciados consoante a natureza do bloco a processar. Tal deve-se ao algoritmo que permitiu acelerar este processamento utilizando o conjunto de informação de qualificação de cada macrobloco baseado na tabela CBPC descrita na secção 2.1.3.2.

As tabelas seguintes (Tabela 4.3 e Tabela 4.4) permitem obter uma ideia mais ampla no que se refere ao tempo total de processamento de uma imagem e, por consequência, ao número total de imagens passíveis de serem codificadas durante um segundo.

		Nível Bloco [µs]	Nível MB [µs]	Nível GOB [µs]	Nível Picture [µs]	Codificador [ms]
Bloco	Total	404.1 x 6 =	2424.9			
MB	*Header*		3.36			
	Total		2428.2 x 11 =	26710.7		
GOB	*Header*			2.8		
	Total			26713.5 x 5 =	133567.7	
Picture	*Header*				8.7	
	Tail				6.1	
	Total				133582.5	133.583
OverHead	Total					24.731
Total [ms]						**158.314**
Frequencia [imag./s]						**6.3**

Tabela 4.3 - Pior caso - Todos os blocos têm a totalidade dos coeficientes não nulos.

		Nível Bloco [µs]	Nível MB [µs]	Nível GOB [µs]	Nível Picture [µs]	Codificador [ms]
Bloco	Total	129.4 x 6 =	776.6			
MB	*Header*		3.36			
	Total		780.0 x 11 =	8579.9		
GOB	*Header*			2.8		
	Total			8582.7 x 5 =	42913.4	
Picture	*Header*				8.7	
	Tail				6.1	
	Total				42928.2	42.928
OverHead	Total					24.731
Total [ms]						**67.659**
Frequencia [imag./s]						**14.8**

Tabela 4.4 - Melhor caso - Todos os blocos têm todos os coeficientes nulos excepto o coeficiente DC.

A partir dos resultados obtidos, pode-se concluir que, dependendo do conteúdo de informação da imagem, o codificador implementado permite obter taxas de codificação variáveis entre 6 e 14 imagens por segundo. Contudo, faz-se notar que estes valores podem ser inferiores visto que não se considerou, nos cálculos anteriores, a influência do nível de enchimento dos *buffers* de emissão (dependente da taxa de transmissão da informação), assim como a componente do tempo total correspondente à fase de DMA entre o co-processador de aquisição e os dois DSPs.

Em relação ao descodificador H.263, não foram realizadas medidas semelhantes às apresentadas para o caso do codificador por duas razões distintas:

- em primeiro lugar, como o bloco de descodificação apresenta um algoritmo computacionalmente menos pesado do que o bloco de codificação, verifica-se que o tempo total para o processamento da descodificação não é determinante para o correcto funcionamento do sistema global;
- em segundo lugar, como as rotinas implementadas no bloco de descodificação são praticamente idênticas às rotinas correspondentes do bloco de codificação, os valores medidos seriam muito semelhantes aos apresentados nas tabelas Tabela 4.3 e Tabela 4.4).

5 Conclusões e Trabalho Futuro

Neste capítulo, apresenta-se uma breve compilação de algumas conclusões que se podem tirar após a realização deste trabalho. Apresenta-se, também, um conjunto de reflexões e sugestões que permitem continuar o trabalho realizado, no sentido de melhorar o desempenho do sistema.

5.1 DESCRIÇÃO DO TRABALHO REALIZADO

Na Figura 5.1, apresenta-se um diagrama de blocos que permite descrever o sistema de transmissão de vídeo através da rede eléctrica.

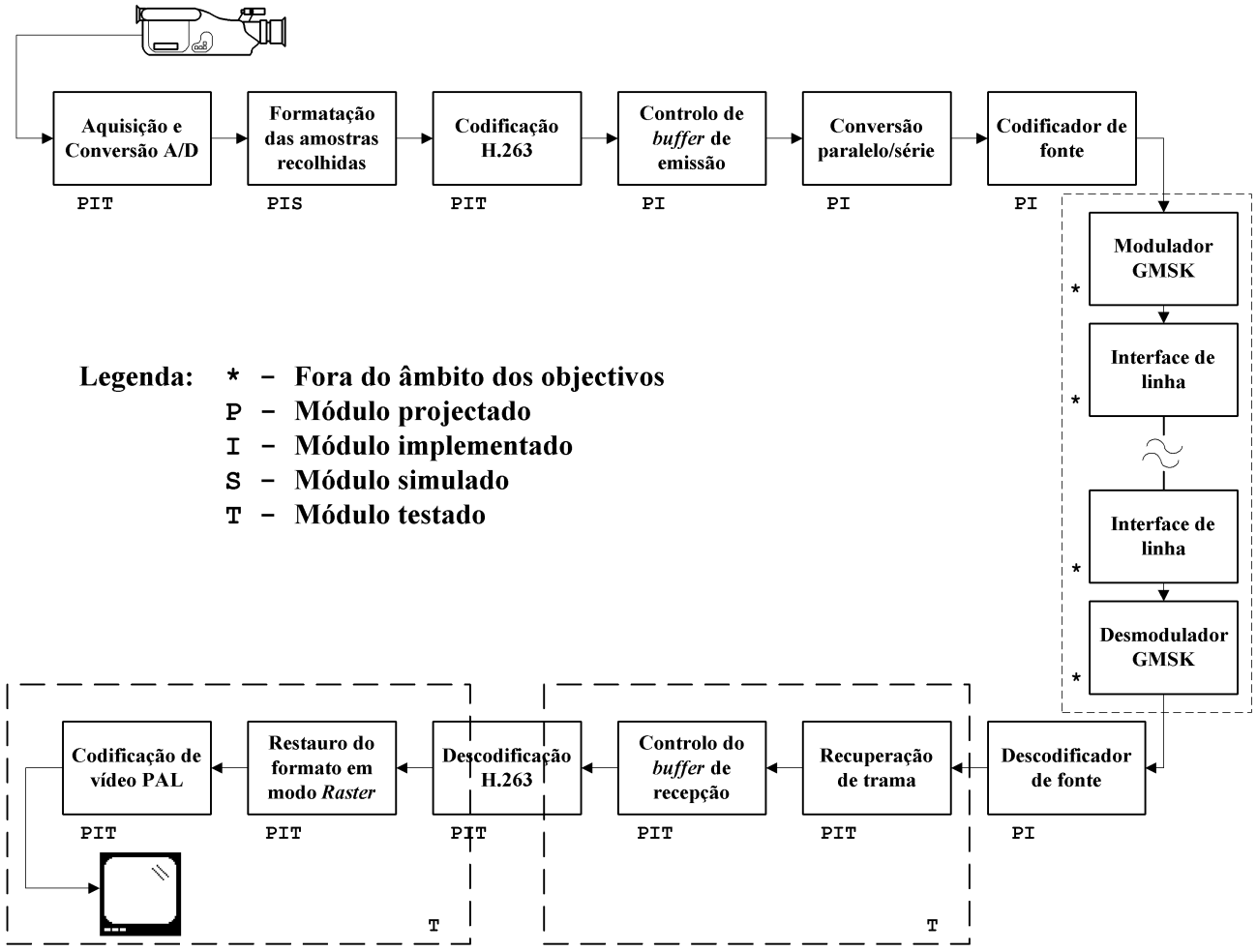


Figura 5.1 - Fase de desenvolvimento do sistema implementado.

Como se pode observar, representam-se neste diagrama os vários módulos descritos ao longo deste relatório, nomeadamente nos capítulos 2 e 3 referentes aos sistemas emissor e receptor, respectivamente.

Tal como se referiu na secção dedicada aos objectivos do trabalho, efectuou-se o projecto e implementação de um sistema de codificação e descodificação de vídeo segundo a norma H.263, para ser utilizado com o módulo de transmissão suportado pela rede de distribuição de energia eléctrica. Contudo, dada a dimensão global deste sistema e visto que existe a intenção de o mesmo continuar a ser desenvolvido, num futuro próximo, por outra equipa de trabalho, alguns módulos do sistema não se encontram ainda testados, embora estejam completamente projectados e

implementados. Assim, cada um dos módulos foi representado no diagrama de blocos anterior com um conjunto de símbolos a ele associados que permitem identificar as fases de desenvolvimento atingidas em cada um deles.

Assim, de acordo com a Figura 5.1, é possível identificar os estágios de desenvolvimento de cada um dos módulos do sistema:

- o módulo de *aquisição e conversão A/D* foi implementado e testado, encontrando-se a funcionar;
- o módulo de *formatação das amostras recolhidas* encontra-se completamente implementado, sendo que os circuitos das duas FPGAs que compõem o co-processador foram completamente simulados através das ferramentas de desenvolvimento utilizadas [11];
- o módulo respeitante à *codificação H.263*, desenvolvido utilizando dois DSPs do tipo TMS320C50, foi implementado e testado integralmente, utilizando amostras de imagens inseridas na sua memória;
- os módulos de *controlo do buffer de emissão*, de *conversão paralelo-série* da trama gerada e de *codificação de fonte* estão completamente implementados, não se tendo ainda terminado a sua fase de teste;
- em relação aos módulos de *modulação* e *desmodulação* e de *interface* do sinal a introduzir na linha de transmissão, não foi efectuado qualquer tipo de desenvolvimento dado não se enquadrarem nos objectivos propostos para este trabalho;
- tal como o *codificador de fonte*, o *descodificador de fonte* encontra-se implementado, faltando ainda realizar o teste deste módulo;
- os restantes módulos do sistema correspondentes à *recuperação da trama H.263*, ao *controlador do buffer de recepção*, à *descodificação de vídeo*, ao *restauro do formato em modo Raster* assim como a *codificação do vídeo para o formato PAL*, encontram-se na sua fase final de desenvolvimento, tendo sido completado o teste de todos estes módulos. Contudo, devido a algumas dificuldades encontradas na interligação da placa desenvolvida para o DSP com os restantes módulos, não foi possível, ainda, efectuar o teste completo do descodificador H.263.

Desta forma, a situação actual do projecto permite que o desenvolvimento do mesmo seja facilmente concluído num futuro próximo, conforme se indicará na secção seguinte.

5.2 IMPLEMENTAÇÕES FUTURAS

Tal como referido anteriormente, existem ainda algumas etapas no desenvolvimento deste projecto que necessitam de ser concluídas por forma a finalizar a implementação do sistema global. Estas etapas resumem-se, praticamente, ao teste e interligação dos módulos realizados.

Dadas as dificuldades inerentes ao funcionamento das memórias FIFO utilizadas (tal como se descreveu na secção 2.1.4), sugere-se a sua substituição por umas memórias estáticas FIFO, permitindo assim simplificar, substancialmente, o controlador de *buffer* realizado. Como exemplo, sugere-se a utilização de memórias do tipo IDT7208 da IDT [16] que implementam uma estrutura FIFO baseada numa memória estática com a capacidade de armazenamento de 64k x 9 bits. Estas memórias disponibilizam, ainda, sinais indicadores do nível de enchimento que permitem simplificar o desenvolvimento do controlador. Além disso, dada a sua natureza estática, permitem eliminar os problemas associados com o refrescamento referidos na secção 2.1.4.

Para melhorar a eficiência da codificação, sugere-se também a implementação de estimação de movimento nos módulos de codificação e descodificação H.263. Para isso, pode-se adoptar uma de duas soluções:

- Uma solução consiste na utilização dos DSPs para a implementação do algoritmo. Esta solução apenas obriga a ligeiras alterações nos circuitos implementados na FPGA2, que faz parte do co-processador de formatação das amostras recolhidas. Tal facto deve-se a que a área de pesquisa utilizada no algoritmo abrange 31 *pixels* em torno do macrobloco em processamento nas direcções ortogonais. Como tal, seria necessário o preenchimento de uma linha de macroblocos adicional na memória de cada um dos DSPs e que seria usada apenas com vista à estimação de movimento;
- A segunda solução passaria pela utilização de um co-processador dedicado, encarregue de efectuar a estimação de movimento. Esta opção apresenta-se como a mais favorável dado o tempo de processamento adicional que a primeira hipótese implicaria.

Por último, sugere-se também que, por forma a implementar o sistema de transmissão de vídeo através da rede eléctrica, é ainda necessário efectuar o projecto e a implementação do módulo que efectua a transmissão dos dados pela linha de distribuição de energia, compreendendo os blocos de modulação e desmodulação e os blocos de interface de linha (ver Figura 5.1).

6 Referências Bibliográficas

[1] - *Draft ITU-T Recommendation H.263 - "Video Coding for Low Bitrate Communication"*, Maio de 1996.

[2] - *TMS320C5x User's Guide*, Texas Instruments, 1993.

[3] - *TMS320C3x User's Guide*, Texas Instruments, 1994.

[4] - *NTSC/PAL to RGB/YCrCb Decoder (Bt812)*, Brooktree Corporation, Fevereiro de 1996.

[5] - *Colour Television - System Principles, Engineering Practice and Applied Technology*, G. Hutson, P. Shepherd, J. Brice, 2ª edição, McGraw-Hill, 1990.

[6] - *PIC16/17 – Microcontroller Data Book*, Microchip, 1996/1997.

[7] - *Discrete Cosine Transform*, K.R. Rao and P. Yip, Academic Press, 1990.

[8] - *IS61C1024 – 128K x 8 High Speed CMOS static RAM*, Integrated Silicon Solution, Inc., Julho de 1997.

[9] - *XC4000E and XC4000X series Field Programmable Gate Array*, Xilinx, Junho de 1997.

[10] - *VHDL - Analysis and Modeling of Digital Systems*, Zainalabedin Navabi, McGraw-Hill, 1993.

[11] - *Synopsys Online Documentation*, versão 1997.08, Synopsys Inc., 1997.

[12] - *XACTstep Alliance – Quick Start Guide vM1.3*, Xilinx, 1997.

[13] - *Computer Architecture: a Quantitative Approach*, 2ª edição, Patterson & Hennessy, 1996.

[14] - *Am27C64 – 64 Kilobit CMOS EPROM*, Advanced Micro Devices, Maio de 1995.

[15] - *TMS4C1050 – 1 Megabit First-In First-Out Pseudo Static Memory - Preliminary*, Texas Instruments, 1987.

[16] - *IDT7208 - CMOS Asynchronous FIFO*, Integrated Device Technology, Inc., 1996.

[17] - *Signalling on low-voltage electrical installations in the frequency range 3kHz to 148,5kHz - European Standard*, CENELEC – Janeiro de 1991.

[18] - *Q1900 Viterbi/Trellis Decoder*, QUALCOMM Inc, Fevereiro de 1997.

[19] - *IDT7007L - High Speed 32k x 8 Dual-Port Static RAM*, Integrated Device Technology, Inc., Janeiro de 1998.

[20] - *12-15 MHz RGB/YCrCb to NTSC/PAL Decoder (Bt858)*, Brooktree Corporation, Maio de 1995.

[21] - *The TTL Data Book (Vol.II)*, Texas Instruments, 1989.

[22] - *Digital Integrated Electronics*, Herbert Taub, Donald Schilling, McGraw-Hill, 1997.

[23] - *Microelectronics Circuits*, Adel S. Sedra, Kenneth C. Smith, Saunders College Publishing, 1991.

[24] - *Sistema de aquisição e visualização de imagem em tempo real*, Pedro Leitão, Pedro Sequeira, Dezembro de 1998.

[25] - *Compressão de Vídeo para Canais de Banda Estreita*, Kevin Ferreira, Março de 1996.

[26] - *Digital Video Transmission Through the Electrical Power Lines*, Alexandre Abreu, Nuno Roma, José Gerald, Leonel Sousa, "*The Second European DSP Education and Research Conference*" - *Proceedings*, págs. 16-21, Texas Instruments, Setembro de 1998.

A Norma H.263

O codificador de vídeo implementado baseia-se na norma H.263. Esta norma, é vocacionada para a codificação e transmissão de vídeo em canais de banda estreita (<64 kbit/s) e tem como base a recomendação ITU-T H.261. Neste apêndice, apresentam-se as principais características desta norma de codificação.

A.1 CODIFICAÇÃO DE VÍDEO

A.1.1 Formato das Imagens

As sequências de imagens nesta recomendação são constituídas por imagens não entrelaçadas com uma resolução temporal de 30000/1001 (aproximadamente 29,97) imagens por segundo. Os codificadores de H.263 podem admitir cinco formatos espaciais: 16CIF, 4CIF, CIF, QCIF e sub-QCIF. Estes formatos são compostos por uma componente de luminância Y e duas componentes de crominância C_B e C_R, obtidas a partir das componentes RGB normalizadas (dadas por $\tilde{R}$, $\tilde{G}$ e $\tilde{B}$):

$$Y' = 0{,}299.\tilde{R} + 0{,}587.\tilde{G} + 0{,}114.\tilde{B} \tag{A1}$$

$$\begin{cases} Y = 219.Y' + 16 \\ C_B = \dfrac{112.\left(\tilde{B} - Y'\right)}{0{,}886} + 128 \\ C_R = \dfrac{112.\left(\tilde{R} - Y'\right)}{0{,}701} + 128 \end{cases} \tag{A2}$$

As resoluções espaciais de cada um destes formatos são as indicadas na Tabela A.1:

Formato	Y (linhas x colunas)	C_B e C_R (linhas x colunas)
16CIF	1152 x 1408	576 x 704
4CIF	576 x 704	288 x 352
CIF	288 x 352	144 x 176
QCIF	144 x 176	72 x 88
Sub-QCIF	96 x 128	48 x 64

Tabela A.1 - Formatos de imagem da norma H.263.

A norma refere ainda que todos os descodificadores devem ser capazes de operar com os formatos QCIF e sub-QCIF, podendo alguns também operar com os formatos CIF, 4CIF ou 16CIF. No que se refere aos codificadores, a norma indica que todos devem ser capazes de operar com os formatos QCIF ou sub-QCIF, não sendo obrigados a operar com os dois.

A.1.2 Representação Hierárquica das Imagens

Segundo a norma H.263, cada imagem é dividida numa estrutura hierárquica de 4 níveis:

- PIC (*Picture*) – a imagem que constitui o nível superior;
- GOB (*Group of Blocks*) – grupo de blocos;
- MB (*Macroblock*) – macrobloco;
- Block – bloco, que constitui o nível mais baixo da hierarquia.

Assim, a imagem é dividida em blocos de 8x8 *pixels*. Cada MB é então constituído por quatro blocos de luminância, representando um quadrado de 16x16 *pixels* e por dois blocos de crominância com 8x8 *pixels* (C_B e C_R), espacialmente correspondentes. Cada GOB é constituído por *k* linhas de macroblocos, onde *k*=1 para os formatos sub-QCIF, QCIF e CIF; *k*=2 para 4CIF e *k*=4 para 16CIF. Por sua vez, a imagem (*Picture*) é constituída por 6 GOBs no formato sub-QCIF, 9 GOBs no QCIF e 18 GOBs nos formatos CIF, 4CIF e 16CIF. Na Figura A.1 apresenta-se um exemplo desta hierarquia para o formato QCIF.

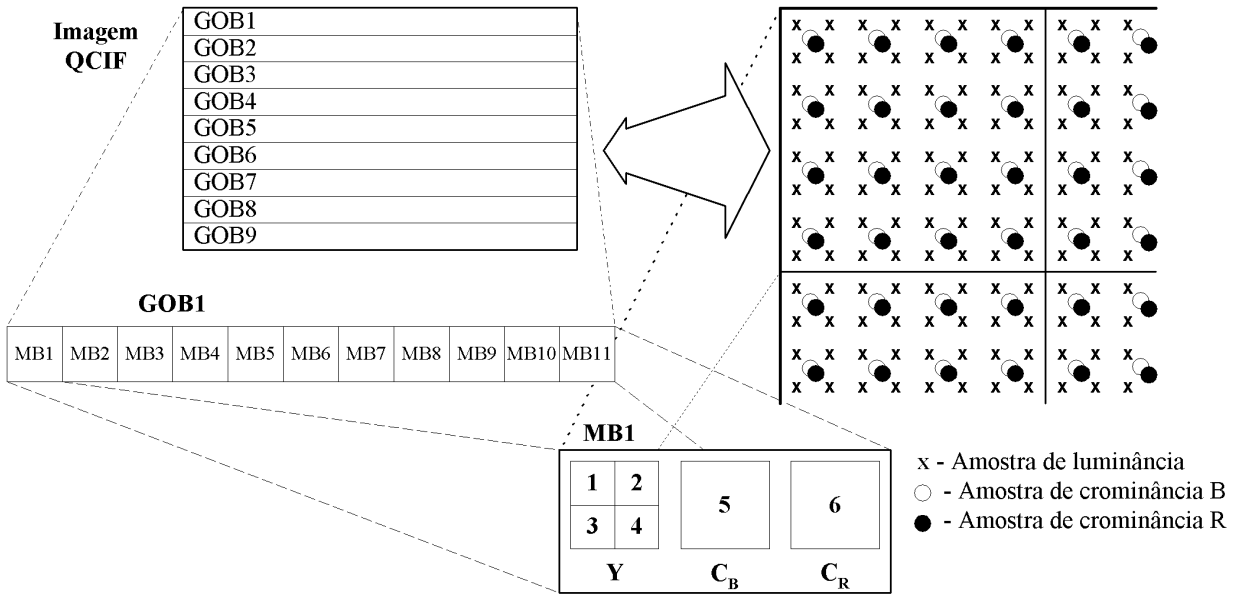


Figura A.1 - Estrutura hierárquica do formato QCIF.

A.1.3 Algoritmo de Compressão

O algoritmo de compressão tenta tirar partido das características das imagens e do sistema visual humano, explorando a redundância temporal (ou redundância estatística), relacionada com a correlação das imagens ao longo do tempo; e a redundância espacial (ou irrelevância do detalhe), relacionada com a correlação espacial dos *pixels*, tentando-se aproveitar a incapacidade de perceber para além de uma determinada resolução da imagem.

Assim, a redundância espacial explora a existência de várias zonas de luminosidade constante ou com pequenas variações, que constituem a maior parte da imagem. Desta forma, é de esperar que numa representação da imagem no domínio da frequência espacial, as componentes de frequência se encontrem muito menos correlacionadas e que as componentes de baixa frequência concentrem grande parte da energia.. Assim, o objectivo da codificação é o de concentrar a energia de uma imagem num pequeno número de coeficientes muito descorrelacionados, com possível desprezo dos

coeficientes correspondentes a frequências mais elevadas e de menor energia. A transformada adoptada por esta recomendação foi a DCT (*Discrete Cosine Transform*):

$$F(u,v) = \frac{C(u).C(v)}{4} \cdot \sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y).\cos\left[\frac{p\ (2.x+1)}{16}\cdot u\right].\cos\left[\frac{p\ (2.y+1)}{16}\cdot v\right] \qquad \text{(A3)}$$

e a correspondente transformada inversa:

$$f(x,y) = \frac{C(u).C(v)}{4} \cdot \sum_{x=0}^{7}\sum_{y=0}^{7} F(u,v).\cos\left[\frac{p\ (2.x+1)}{16}\cdot u\right].\cos\left[\frac{p\ (2.y+1)}{16}\cdot v\right] \qquad \text{(A4)}$$

com $u, v, x, y = 0, 1, 2, \ldots 7$
e onde x, y – coordenadas espaciais no domínio do *pixel*;
u, v – coordenadas no domínio da transformada;

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & ,k = 0 \\ 1 & ,k \neq 0 \end{cases}.$$

A redundância temporal é explorada, fazendo-se a predição do valor de um determinado *pixel* a codificar a partir do valor de um *pixel* da imagem anterior. Para tal, utilizam-se códigos do tipo diferencial, transmitindo-se a diferença entre os *pixels* correspondentes em imagens consecutivas.

Uma forma de explorar a redundância temporal e obter ainda maiores factores de compressão consiste na utilização de técnicas de compensação de movimento. Nestas técnicas aproveita-se o movimento lento de regiões entre imagens adjacentes que permite pesquisar a movimentação de um bloco de referência sobre uma área de pesquisa. Neste tipo de técnicas identifica-se, geralmente, apenas movimentos de translação. Na codificação com estimação de movimento envia-se, associado às diferenças, os respectivos vectores de movimento.

Contudo, uma das principais dificuldades encontradas no uso destas técnicas de codificação e compressão é a acumulação de erros, que pode degradar consideravelmente a qualidade das imagens transmitidas. Para evitar este problema, a norma H.263 impõe dois modos de funcionamento distintos: Modo Inter-Imagem[20], a que se pode também chamar de modo diferencial; e o modo Intra-Imagem[21], que consiste na codificação isolada de uma imagem. Assim, em funcionamento normal o sistema deverá codificar no modo diferencial (INTER). No entanto, deverá enviar uma imagem codificada no modo INTRA com uma periodicidade não superior a 132 imagens, tentando assim eliminar os erros acumulados durante a transmissão de uma sequência de imagens no modo diferencial.

Os principais blocos da norma de codificação H.263 encontram-se representados na Figura A.2.

[20] Do inglês *Interframe*.
[21] Do inglês *Intraframe*.

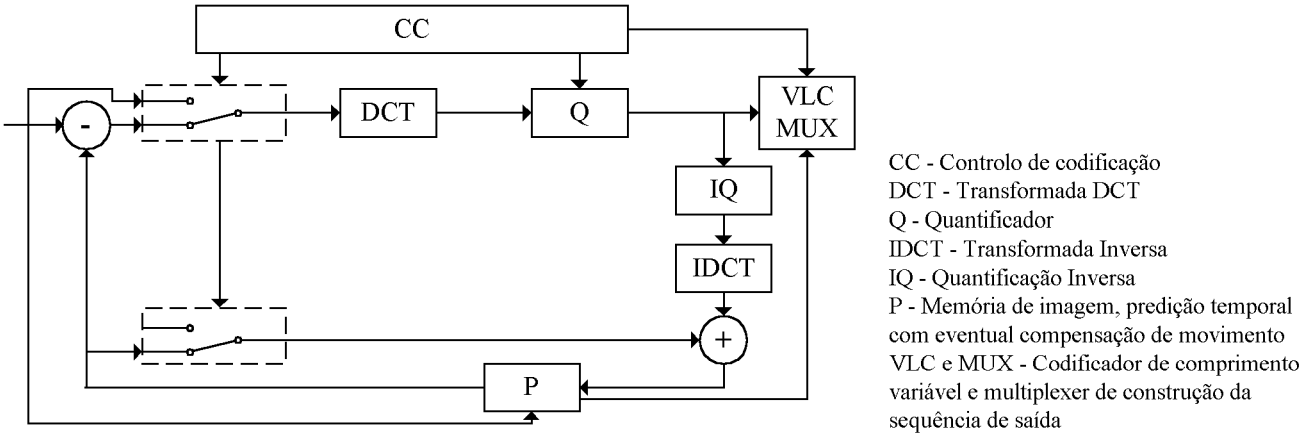


Figura A.2 - Diagrama de blocos do codificador.

A.1.3.1 Bloco de Predição Temporal e Estimação de Movimento (P)

Como se referiu anteriormente, no bloco de predição temporal utiliza-se codificação diferencial e técnicas de compensação de movimento. Quando se utiliza predição, os blocos são codificados no modo INTER e é necessário manter actualizada a memória de imagem com a imagem anterior à imagem a codificar.

A compensação de movimento é de utilização opcional. Quando implementada, deve ser utilizada ao nível do Macrobloco e deve usar vectores de movimento com precisão de um ou meio *pixel*, restringidos à gama [-16;15,5]. Dado que existe uma elevada correlação entre os vectores de movimento de MBs vizinhos, utiliza-se o método de codificação diferencial para tornar a codificação dos vectores de movimento mais eficiente. Assim, o vector de movimento diferencial a transmitir é obtido por subtracção de um vector de previsão ao vector do MB actual. O vector de previsão é composto pelos valores medianos nas suas componentes, de três vectores vizinhos, conforme se ilustra na Figura A.3.

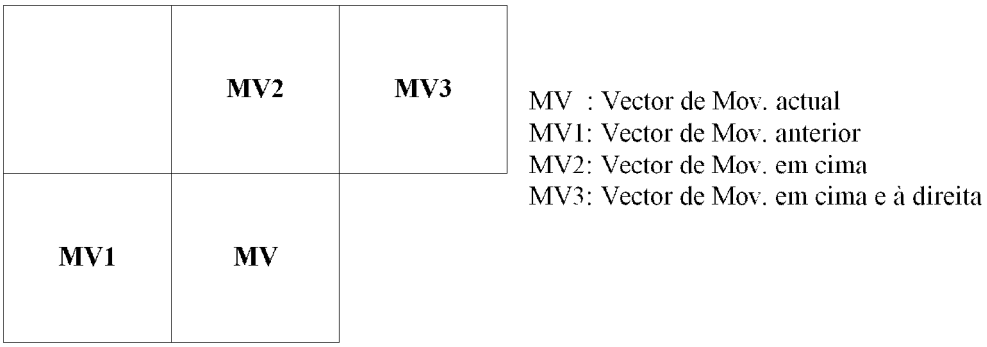


Figura A.3 - Vectores de movimento usados para a previsão do vector de movimento a transmitir.

Os vectores resultantes são posteriormente codificados com códigos de comprimento variável, de acordo com as tabelas indicadas na recomendação.

A.1.3.2 Bloco da Transformada DCT e IDCT

Estes blocos realizam a transformada da imagem ou do resultado da diferença entre a imagem actual e a imagem anterior e a transformada inversa para obtenção da imagem após a codificação.

A.1.3.3 Bloco Quantificador/Desquantificador (Q e IQ)

Estes blocos têm como função estabelecer uma correspondência entre o conjunto de valores resultante do bloco que efectua a DCT e um conjunto limitado de valores, conseguindo-se assim obter compressão da informação. Como os elementos resultantes são apenas uma aproximação dos valores de entrada, este bloco introduz erro na codificação, sendo que a norma H.263 efectua codificação com erro.

A recomendação H.263 utiliza dois tipos de quantificação: o primeiro coeficiente dos blocos INTRA (coeficiente DC) é quantizado com um quantificador uniforme de passo 8. Todos os restantes coeficientes são quantizados com um quantificador com uma zona morta em torno da origem e em que o passo de quantificação é um número par que pode variar entre 2 e 62, tal como se ilustra Figura A.4.

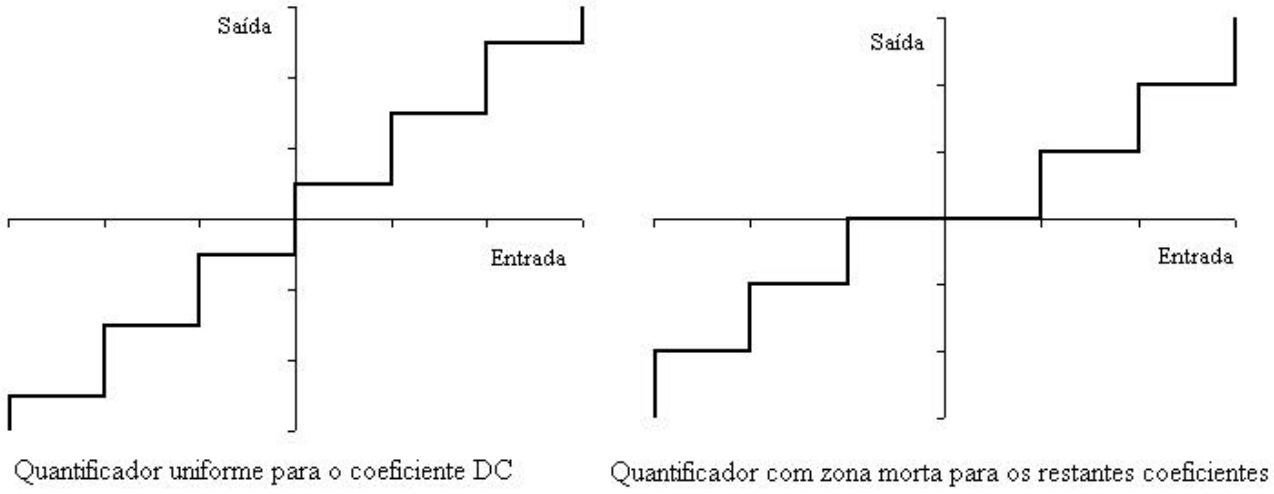


Figura A.4 - Quantificadores utilizados na norma H.263.

Dentro de cada MB o quantificador utilizado é sempre o mesmo, com a eventual excepção do coeficiente DC no caso das imagens INTRA.

A.1.3.4 Bloco de Codificação de Comprimento Variável de Coeficientes (VLC)

Após a quantificação dos coeficientes da transformada, estes são percorridos em *zig-zag* segundo a ordem representada na Figura A.5, ou seja, desde os coeficientes correspondentes às frequências mais baixas até aos coeficientes correspondentes às frequências mais altas. Como é provável que a partir de um determinado ponto muitos dos coeficientes sejam nulos, com a sequência em *zig-zag* obtém-se uma maior taxa de compressão, atribuindo-se códigos de comprimento variável com menor número de bits aos coeficientes lidos nas primeiras diagonais da matriz. Estes códigos encontram-se tabelados na recomendação [1].

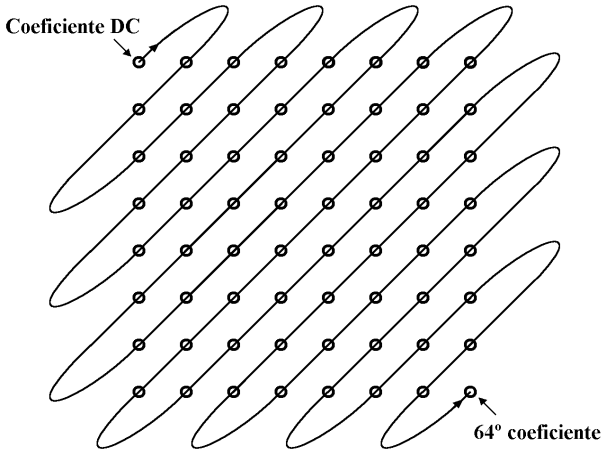


Figura A.5 - Codificação em *zig-zag* dos coeficientes.

A.1.3.5 Bloco de Controlo de Codificação (CC)

Este bloco tem como função efectuar o controlo de todo o sistema codificador, podendo alterar a taxa de compressão através do ajuste de vários parâmetros. Entre estes parâmetros salienta-se o passo de quantificação, o modo de codificação dos Macroblocos (INTRA/INTER) e a decisão de realizar sub-amostragem temporal, eliminado imagens inteiras. Contudo, a estratégica de controlo não é definida na norma.

No entanto, como se referiu anteriormente, a norma exige que cada MB seja codificado em modo INTRA pelo menos uma vez em cada 132 imagens - codificação INTRA forçada.

A.2 SINTAXE E SEMÂNTICA

A sintaxe da sequência de bits resultante da codificação obedece à estrutura hierárquica, tal como foi descrita na secção anterior. Assim, podem-se distinguir 4 níveis: *Picture*, *GOB*, *MB* e *Bloco*. Nas figuras seguintes apresenta-se a estrutura da trama gerada. Nestas figuras, representaram-se com um rectângulo os códigos de comprimento fixo e com um rectângulo ovalizado os códigos de comprimento variável.

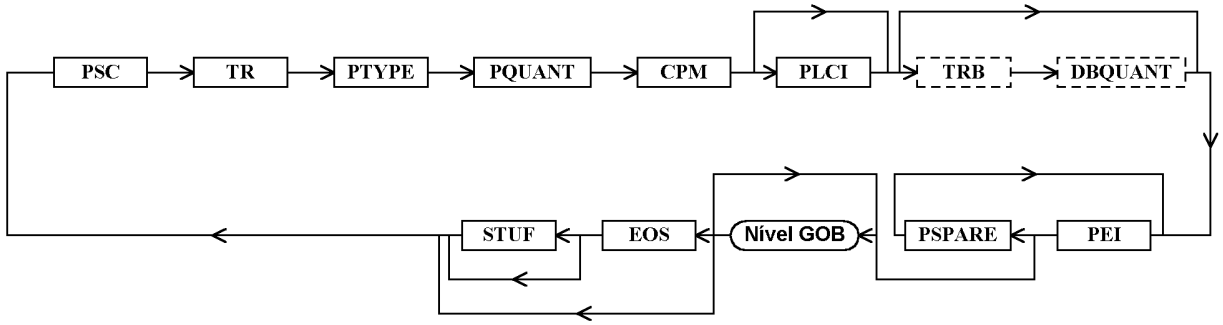


Figura A.6 - Estrutura do nível *picture* da trama H.263 resultante do processo de codificação.

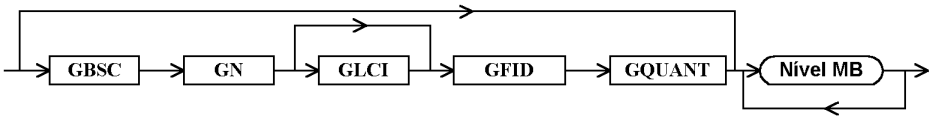


Figura A.7 - Estrutura do nível GOB da trama H.263 resultante do processo de codificação.

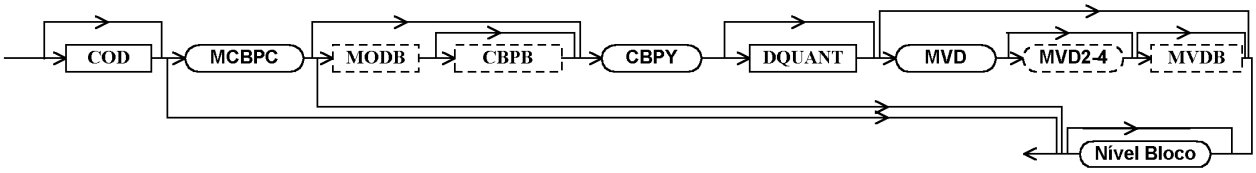


Figura A.8 - Estrutura do nível MB da trama H.263 resultante do processo de codificação.

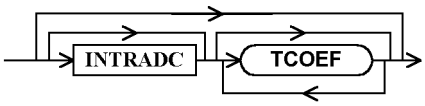


Figura A.9 - Estrutura do nível de bloco da trama H.263 resultante do processo de codificação.

A recomendação H.263 obriga a um alinhamento de bits dos códigos de início de *Picture* e *GOB*. Assim, torna-se necessário inserir bits de "*stuffing*" (0 a 7 zeros) de forma a que o primeiro bit dos códigos de início (*Start Codes*) seja o primeiro bit de um byte (ou seja, o mais significativo). É também necessário inserir bits no fim da *Picture*, de forma a que o último bit transmitido seja o menos significativo de um byte.

A recomendação prevê ainda a inclusão de alguns campos pertencentes a implementações opcionais e negociáveis, não obrigatórias. Para obter mais pormenores sobre estas e outras opções, recomenda-se a consulta da recomendação H.263 referida em [1].

De seguida, passar-se-á a descrever os vários campos de trama H.263.

Campos no Nível *Picture*

- PSC - *Picture Start Code* - 22 bits + (0 - 7 bits de alinhamento)
- TR - *Temporal Reference* - 8 bits
  Obtido por incremento do TR da *Picture* anterior mais o número de *Pictures* não transmitidas (a 29,97Hz).
- PTYPE - 13 *bits*
  Tipo de *Picture*. Contém informação, nomeadamente sob o formato da imagem (QCIF, CIF etc.), se é INTER ou INTRA, se estão presentes modos negociáveis activos, etc..
- PQUANT - 5 bits
  Informação sobre o quantificador. Indica o passo de quantificação a utilizar.
- CPM - *Continuos Presence Multipoint* - 1 *bit*
  Indica multiponto.

- PLCI - *Picture Logical Channel Indicator* - 2 *bits*
  Indica o canal quando em multiponto.
- TRB - Temporal Reference B-frames - 3 bits
  Referência temporal para as imagens B, quando activada a opção imagens-PB.
- DBQUANT - 2 *bits*
  Informação sobre o quantificador para as imagens B, quando activada a opção imagens-PB.
- PEI - 1 *bit*
  Indica a presença do próximo campo.
- PSPARE - 0,8,16 ….*bits*
  Consiste em 8 *bits* extra, mais um bit de PEI, que poderá indicar a repetição, ou não, desta estrutura. PSPARE não deverá ser utilizado pelos codificadores e deverá ser descartado pelos descodificadores até futuras especificações.
- EOS - *End Of Sequence*- 22 *bits* + (0 - 7 *bits* de alinhamento)
  Opcional no codificador.
- STUF - Comprimento variável
  Bits (zeros) extra de forma a efectuar o alinhamento de *Picture.*

Campos no Nível GOB

O cabeçalho do GOB (GBSC, GN, GLCI, GFID e GQUANT) não é transmitido para o primeiro GOB de cada *Picture.* Para os restantes GOBs, a inserção do cabeçalho é opcional.

- GBSC - *Group of Block Start Code* - 17 bits + (0-7 bits de alinhamento)
- GN - *Group Number* - 5 bits
- GLCI - *GOB Logical Channel Indicator* - 2 *bits*
  Apenas presente se em multiponto
- GHD - *GOB Frame ID* -2 bits
- GQUANT -5 *bits*
  Alteração do passo de quantificação para o presente GOB.

Campos no Nível MB

- COD - 1 *bit*
  Indica se o MB é codificado ou não (o MB mantém-se idêntico ao temporalmente anterior, os restantes campos estão vazios). Este campo apenas existe no modo INTER.
- MCBPC - *Macroblock type & Coded block pattern for crominance* - comprimento variável
  Indica o tipo do MB e indica quais os blocos de crominância que têm coeficientes não nulos.
- MODB - Comprimento variável
  Modo para MB do tipo B. Só presente com a opção negociável PB.
- CBPB - *Coded block pattern for B-blocks - 6 bits*
  Só presente se indicado por MODB. Indica os blocos com coeficientes não nulos.
- CBPY - *Coded block pattern for luminance* - comprimento variável.
  Indica quais os blocos de luminância codificados.

- DQUANT - 2 *bits*
  Indica a alteração ao quantificador. Presente quando indicado em MCBPC.
- MVD - *Motion Vector Data* - Comprimento variável
  Vector de movimento, sempre presente para MB INTER.
- MVD2-4 - Comprimento variável
  Os restantes 3 vectores de movimento quando activada a opção negociável de modo avançado de previsão.
- MVDB - Comprimento variável
  Vector de movimento para blocos do tipo B. Presente quando indicado por MODB.

Campos no Nível *Bloco*

- INTRADC - 8 *bits*
  Coeficiente DC para blocos INTRA. Presente para blocos INTRA.
- TCOEF - Comprimento variável
  Codificação Run-Length dos restantes coeficientes do bloco.

B Módulo de Detecção e Correcção de Erros

B.1 DESCRIÇÃO GERAL

O módulo de detecção e correcção de erros tem como objectivo a melhoria, em termos de desempenho, do sistema de transmissão ao nível do BER[22]. Este objectivo é atingido através da introdução de redundância (estruturada) nos dados transmitidos.

O módulo implementado foi concebido através do uso de dois circuitos integrados Q1900 [18] da Qualcomm Incorporated, sendo um deles utilizado como codificador e o outro como descodificador. Estes circuitos integrados são particularmente vocacionados para aplicações de transmissão de dados sob canais onde se verifica a presença de ruído branco gaussiano - AWGN[23].

Cada um destes circuitos integrados permite a codificação dos dados a transmitir segundo dois algoritmos distintos: Algoritmo de Viterbi e Algoritmo de Trellis.

O modo de operação segundo o algoritmo de Viterbi é tipicamente utilizado em sistemas com limitações em termos de potência e onde se verifique a necessidade de efectuar compressão ao nível da largura de banda do sinal transmitido. O sinal de saída neste modo de operação pode ser utilizado para modular uma portadora utilizando modulações do tipo BPSK[24] ou QPSK[25].

O modo de operação segundo o algoritmo de Trellis é também utilizado em sistemas com limitações em termos de potência, mas que apresentam restrições maiores em termos da largura de banda utilizada. Assim, este modo é utilizado quando se verifique a necessidade de efectuar maior nível de compressão da largura de banda do sinal transmitido usando-se, para tal, esquemas de modulação do tipo 8-PSK[26] ou 16-PSK.

Em face do que foi dito, optou-se por utilizar o modo de funcionamento baseado no algoritmo de Viterbi, visto ser este que se ajusta melhor às especificações deste trabalho.

Na Figura B.1 ilustra-se um esquema onde é visível a introdução dos módulos de codificação e descodificação nos sistemas emissor e receptor, respectivamente.

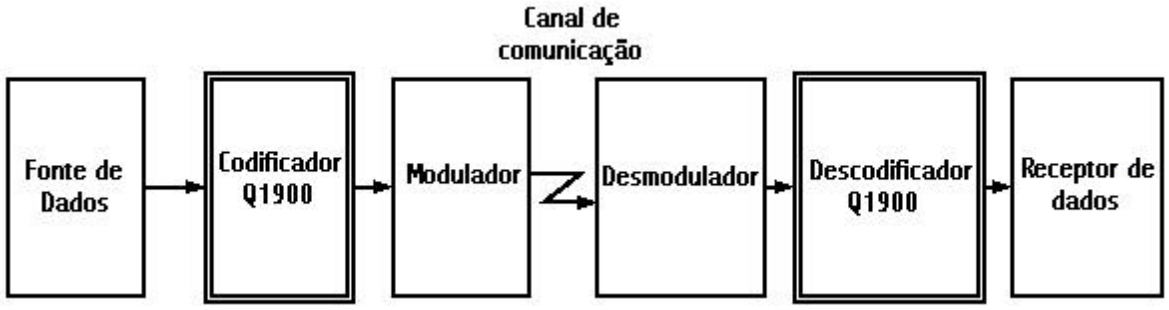


Figura B.1 - Localização dos módulos de codificação e descodificação no sistema global.

[22] Do inglês *Bit Error Rate.*
[23] Do inglês *Additive White Gaussian Noise.*
[24] Do inglês *Binary Phase Shift Keying.*
[25] Do inglês *Quadrature Phase Shift Keying.*
[26] Do inglês *Phase Shift Keying.*

O modo de operação utilizado permite a utilização das seguintes taxas de codificação: 1/3, 1/2, 3/4 e 7/8, em que a nomenclatura n/m significa que o sistema transmite m símbolos por cada n bits de dados. Dadas as limitações em termos de largura de banda disponível verificadas neste projecto, decidiu-se utilizar a taxa de codificação 3/4, visto ser esta que mais se aproxima de uma solução de compromisso entre a redução da taxa de erros dos dados transmitidos e a utilização dos recursos disponíveis, nomeadamente, em termos da largura de banda do canal.

O circuito utilizado suporta, também, sincronização automática da fase dos sinais recebidos modulados em BPSK, QPSK ou OQPSK[27].

Em termos dos sinais de entrada, o circuito descodificador permite a utilização de uma de duas alternativas seguintes:

- O sinal de entrada na forma digital, em que o sinal apresenta apenas dois símbolos possíveis ('0' ou '1') e que se denomina de "Decisão Forçada"[28] ;
- O sinal de entrada apresenta-se quantificado em amplitude (utilizando-se, por exemplo, um conversor A/D) constituído por três bits e que se denomina de "Decisão Ligeira" [29].

O circuito integrado apresenta ainda a possibilidade de se monitorizar o estado de sincronização do circuito e de efectuar medidas ao nível da taxa de erros do sinal recebido (BER).

A programação deste circuito integrado é feita por intermédio de uma interface que incorpora um barramento de endereços de 5 bits e alguns sinais de controlo permitindo, assim, uma interface fácil com um microprocessador, um DSP ou um microcontrolador (como foi o caso neste projecto).

B.2 TEORIA DE FUNCIONAMENTO

B.2.1 Codificador

A conversão da informação da forma de bit para símbolos codificados encontra-se ilustrada na Figura B.2. O número de funções envolvidas neste processo pode ser configurado e pode variar entre uma e quatro.

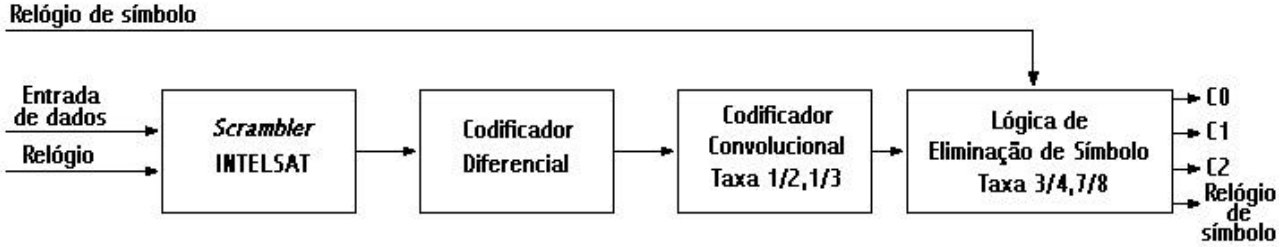


Figura B.2 - Conversão do fluxo binário em símbolos Viterbi.

B.2.1.1 Scrambler

O primeiro módulo efectua uma função de *Scrambler* segundo a especificação do sistema de transmissão via satélite INTELSAT e que não é mais do que uma versão modificada do algoritmo descrito na recomendação V.135 do CCITT.

[27] Do inglês *Offset Quadrature Phase Shift Keying.*
[28] Do inglês *Hard Decision.*
[29] Do inglês *Soft Decision.*

Este módulo é usado frequentemente em sistemas de detecção e correcção de erros para garantir uma densidade de transmissão do sinal mínima por forma a assegurar a manutenção do estado de sincronismo de dispositivos utilizados em sistemas de comunicações tais como PLL's[30].

O circuito de *scrambler* implementado neste circuito integrado consiste num registo de deslocamento de 20 bits, um contador síncrono de 5 bits e alguma lógica adicional, tal como se ilustra na Figura B.3. A operação XNOR das saídas T_1 e T_9 do registo de deslocamento incrementa o contador quando T_1 toma o mesmo valor que T_9 e efectua a inicialização do contador quando T_1 e T_9 tomam valores distintos. A operação AND de todas as saídas do contador (sinais Qas) é definida como a condição crítica ou estado crítico.

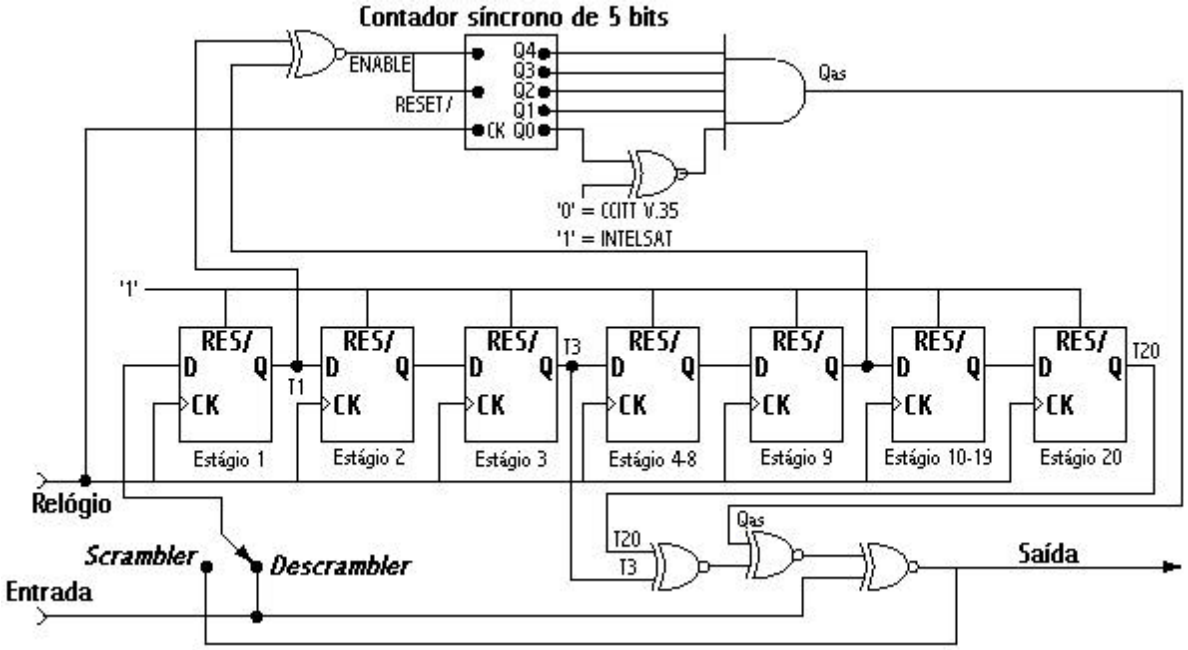


Figura B.3 - *Scrambler* implementado no codificador Trellis/Viterbi.

A saída do *scrambler/descrambler* consiste na combinação lógica das saídas T_3, T_{20}, Qas e da entrada scrambler/descrambler, usada para definir um de dois modos de funcionamento possíveis deste circuito. Desta forma, é possível definir por intermédio de um interruptor o funcionamento do circuito como *scrambler* ou como *descrambler*.

Assim, quando o interruptor se encontra na posição de *scrambler*, o circuito implementa uma operação recursiva, em que a saída do *scrambler* é realimentada na entrada do registo de deslocamento. Quando o interruptor se encontra na posição de *descrambler*, o circuito implementa uma operação não recursiva, onde a saída é combinada com valores da entrada anteriores.

Há ainda a salientar que o *scrambler/descrambler* implementado segundo a norma INTELSAT difere do especificado segundo a recomendação CCITT V.35 apenas na geração do estado crítico Qas. Na norma INTELSAT o estado crítico é definido quando a saída do contador (Q_4-Q_0) assume o valor '`11111`' ao passo que segundo a recomendação V.35 do CCITT este estado é definido quando a saída deste contador assume o valor '`11110`'. A finalidade deste contador síncrono de 5 bits e por conseguinte, da definição do estado crítico Qas é a de garantir uma transição na saída uma vez em cada conjunto de 32 bits para sequências longas de 0's ou de 1's, mantendo assim um número mínimo de transições no sinal de saída.

[30] Do inglês *Phase Locked Loop.*

Para finalizar, convém ainda salientar que, embora as operações de *scrambler* e de *descrambler* venham trazer ao sinal transmitido características que facilitam a sua transmissão pelo canal, a sua utilização pode, em certa medida, deteriorar o desempenho do sistema de codificação/descodificação de Viterbi. Tal facto deve-se ao fenómeno na multiplicação e propagação (pelos ciclos seguintes) de erros dos bits de saída do descodificador. De facto, um único bit errado na entrada do descodificador pode, teoricamente, gerar até 3 bits errados na saída do mesmo. Este fenómeno pode ser compreendido através da Figura B.3, onde se pode observar que o bit errado é introduzido na entrada da porta XNOR cuja saída é realimentada e introduzida na entrada do registo de deslocamento. Assim, à medida que este bit é deslocado pelo *descrambler*, irá dar origem a duas saídas erradas adicionais, verificadas aquando da sua passagem pelos pontos T_3 e T_{20}.

B.2.1.2 Codificador Diferencial

O segundo bloco funcional do codificador de Viterbi consiste num codificador diferencial. O seu diagrama de blocos e o do descodificador respectivo estão ilustrados nas Figura B.4 e Figura B.5.

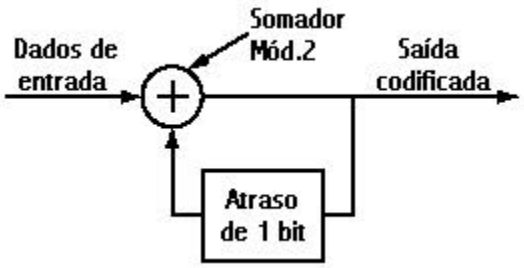


Figura B.4 - Codificador diferencial.

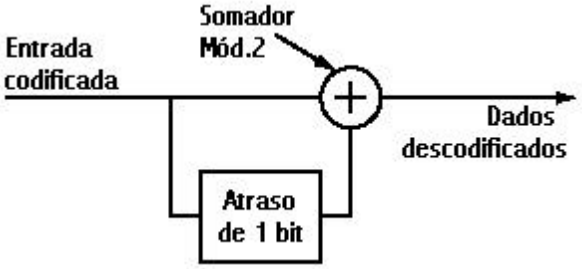


Figura B.5 - Descodificador diferencial.

O codificador diferencial transforma a cadeia de bits de entrada, constituída por sinais lógicos a '1' ou a '0', numa indicação de transições de nível lógico. Assim, se o sinal de entrada do codificador assumir o valor lógico '0', a saída mantém-se constante. Se o sinal de entrada assumir o valor lógico '1', o sinal de saída apresentará uma transição de '0' para '1' ou de '1' para '0'.

B.2.1.3 Codificador Convolucional

O terceiro bloco funcional consiste num codificador convolucional de sete níveis, com taxa de codificação de 1/2 ou 1/3, ilustrado na Figura B.6.

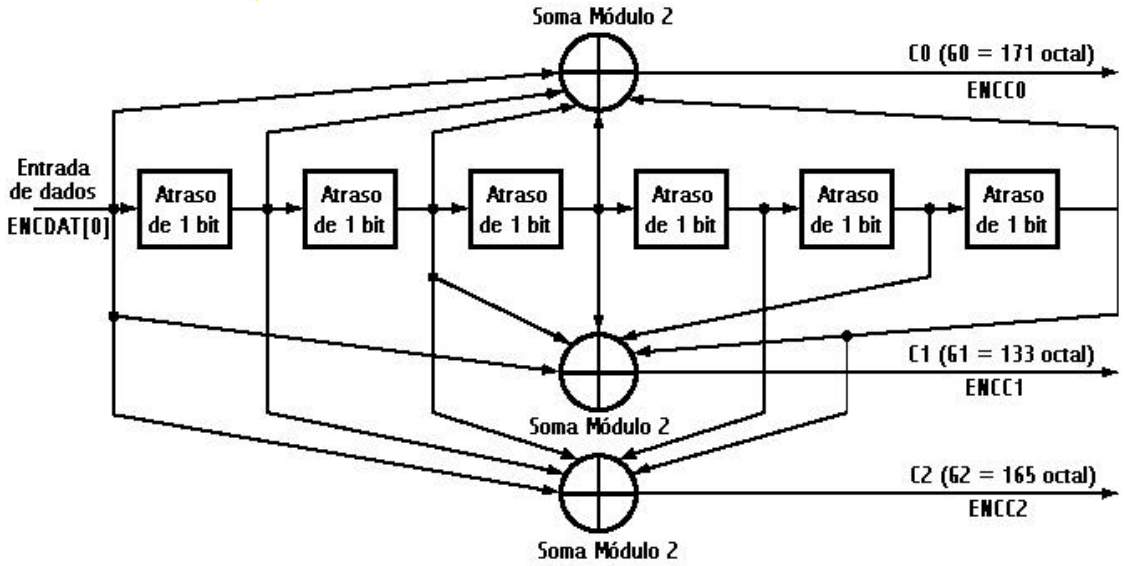


Figura B.6 - Codificador convolucional.

Este módulo é usado para codificar o bit de entrada `ENCDAT[0]` em duas ou três saídas: `ENCC0`, `ENCC1` e `ENCC2` que resultam da operação de adição módulo dois de vários pontos de um registo de deslocamento de seis entradas com a entrada actual.

B.2.1.4 Lógica de Eliminação de Símbolo[31]

O quarto e último bloco funcional do codificador de Viterbi consiste na lógica de eliminação de símbolos, cuja principal função é gerar as taxas de codificação de 3/4 e 7/8 a partir da taxa de codificação 1/2, gerada na saída do bloco anterior. Este processo é ilustrado na Figura B.7 para o caso da taxa de codificação de 3/4.

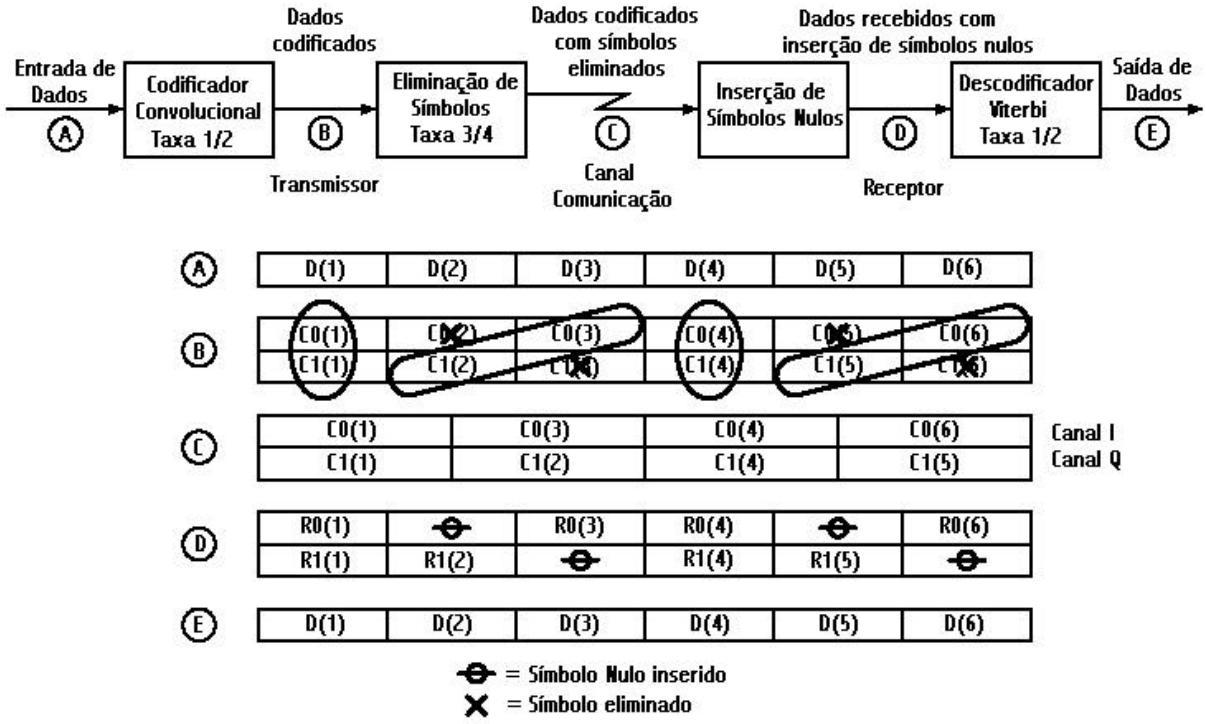


Figura B.7 - Processo de eliminação de símbolos.

[31] Do inglês *Symbol Puncture Logic*

Em primeiro lugar, os dados de entrada são codificados com uma taxa de codificação de 1/2 (significando que são gerados dois símbolos por cada bit de entrada) através do codificador convolucional descrito anteriormente (ponto B).

De seguida são eliminados alguns bits desta trama gerada não sendo, portanto, transmitidos (ponto C). Assim, para gerar a taxa de codificação de 3/4, dois em cada seis bits da trama anteriormente gerada são eliminados segundo um padrão periodicamente repetido. Desta forma, por cada três bits de entrada serão transmitidos quatro bits codificados, dando assim origem a uma taxa de codificação de 3/4.

Para a taxa de codificação de 7/8, seis em cada catorze bits da trama anteriormente gerada são eliminados segundo um padrão repetitivo pelo que, por cada conjunto de sete bits de entrada serão transmitidos oito bits, dando assim origem à taxa de codificação de 7/8.

No receptor, os bits eliminados são substituídos por bits nulos antes de se proceder ao processo de descodificação através de um descodificador de taxa de codificação de 1/2 (ponto D). Esta inserção de bits nulos é efectuada de forma automática pelo circuito integrado desde que este se encontre configurado para funcionar à taxa de 3/4 ou 7/8.

Segundo o fabricante deste circuito, o desempenho deste codificador com taxa de codificação de 3/4 e com eliminação de símbolos é equivalente à de um codificador clássico correspondente a essa taxa de codificação sem eliminação de símbolos. Refere ainda que a vantagem principal desta arquitectura composta por um codificador convolucional *standard* (1/2 ou 1/3) é que um único descodificador pode descodificar uma gama de códigos diferentes, na forma (n-1)/n usando esta estrutura. No entanto, deixa-se claro que o comprimento da máquina de estados ("*chainback depth*") deverá aumentar à medida que a taxa de codificação aumenta. Assim, são indicados comprimentos de 35 a 40 estados para a descodificação de 1/2, 70 estados para a taxa de 3/4 e mais de 90 estados para a taxa de 7/8. Contudo, e devido ao esquema de eliminação de símbolos usados por este circuito, é usado um comprimento mínimo 96 estados para as taxas de 3/4 e 7/8 por forma a garantir a maximização do ganho do codificador. Assim, verifica-se que a arquitectura utilizada apresenta níveis de eficiência bastante aceitáveis para taxas de codificação de até 7/8.

Durante a fase de descodificação, e caso sejam usadas taxas de codificação que envolvam a eliminação de símbolos, o descodificador terá de sincronizar o padrão de inserção de símbolos nulos com o padrão do codificador. Todo este procedimento, assim como os de eliminação de símbolos e de inserção/remoção de símbolos nulos é efectuado de forma automática por este circuito integrado. O circuito inclui ainda uma memória do tipo FIFO[32] que permite sincronizar os dados de saída com um relógio, permitindo assim gerar um fluxo de dados regular.

B.2.2 Descodificador

O descodificador de Viterbi apresenta uma estrutura recíproca da do codificador, tal como se ilustra nas Figura B.8 e Figura B.9. Contudo, se a implementação do codificador convolucional apresenta uma estrutura e implementação simples, o mesmo já não acontece no descodificador respectivo.

[32] Do inglês *First In First Out.*

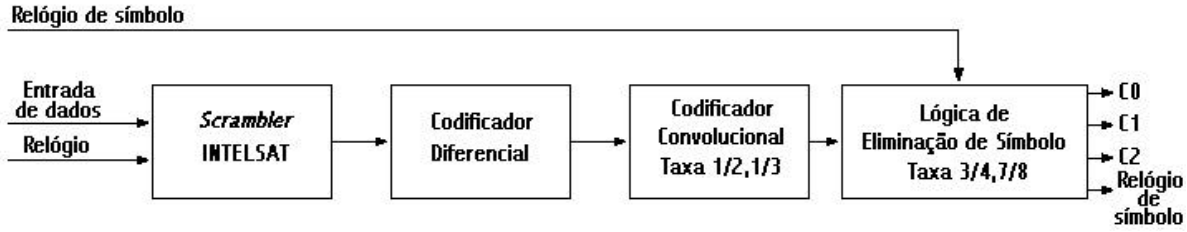


Figura B.8 - Estrutura geral do codificador de Viterbi.

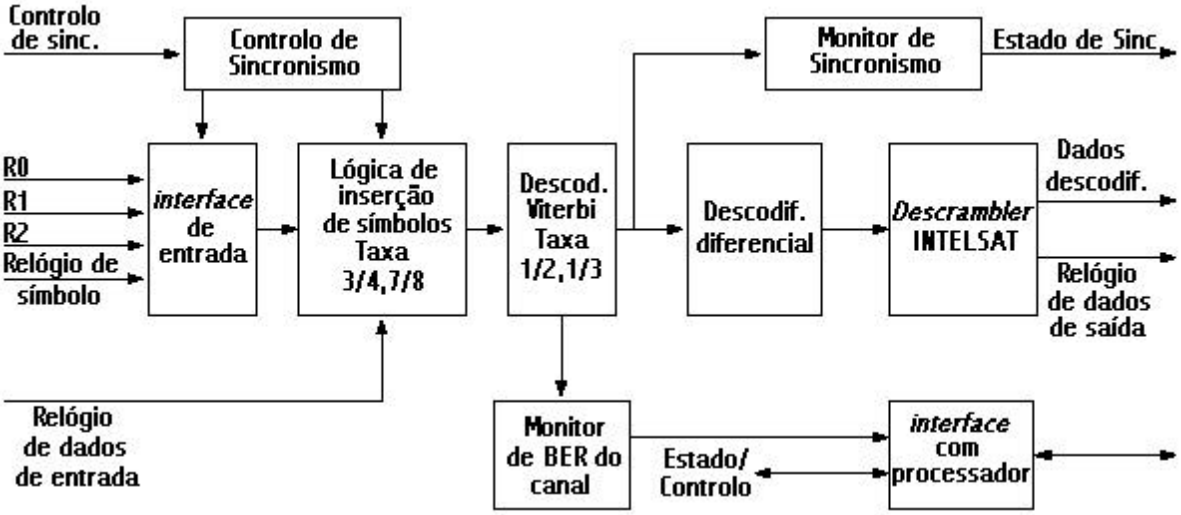


Figura B.9 - Estrutura geral do descodificador de Viterbi.

B.2.2.1 Descodificador de Viterbi

A descodificação de Viterbi é composta por três passos fundamentais.

O primeiro passo consiste na geração de um conjunto de medidas de correlação denominados de *"branch metrics"* para cada grupo de *m* símbolos de entrada, onde *m* toma o valor 2 para taxas de codificação de 1/2, 3 para taxas de codificação de 1/3, etc. Estas medidas indicam a correlação entre os símbolos recebidos e as *2m* combinações possíveis.

O descodificador de Viterbi determina o estado do codificador convolucional de 7 bits usando a técnica de maximização da probabilidade de o codificador se encontrar num determinado estado. Assim, uma vez determinado o estado corrente do codificador, torna-se possível a determinação da informação original visto esta se encontrar armazenada na cadeia de flip-flops que constitui o codificador. Por forma a determinar o estado do codificador, é gerado no segundo passo do algoritmo um conjunto de 2^{k-1} medidas de estado, onde *k* toma, para este circuito, o valor ´7´ e mede a probabilidade de ocorrência de cada um dos 2^{k-1} possíveis estados do codificador. À medida que estas medidas são calculadas, é feita uma operação de decisão por forma a determinar o caminho mais provável para chegar a um estado particular. Estas decisões são então armazenadas em memória.

Por fim, no terceiro passo é armazenada a informação de saída. Tal é possível determinando o caminho do estado actual até um determinado ponto ou estado anterior, consultando-se, para tal, as várias decisões anteriormente efectuadas e guardadas em memória (2º passo). Assim, verifica-se que o efeito de ruído acaba por ser minimizado, visto que o resultado final acaba por convergir para o valor correcto depois de algumas iterações. Para além disso, quanto maior for o comprimento da

máquina de estados do codificador/descodificador, maior é a probabilidade de o resultado final não apresentar qualquer erro. Como consequência, taxas de codificação maiores requerem máquinas de estado com comprimentos maiores para atingir níveis de desempenho superiores.

B.3 DESEMPENHO DA CODIFICAÇÃO

Na Figura B.10 são apresentados os valores das taxas de bits errados (BER[33]) indicados pelo fabricante para diferentes taxas de codificação (1/3, 1/2, 3/4 e 7/8).

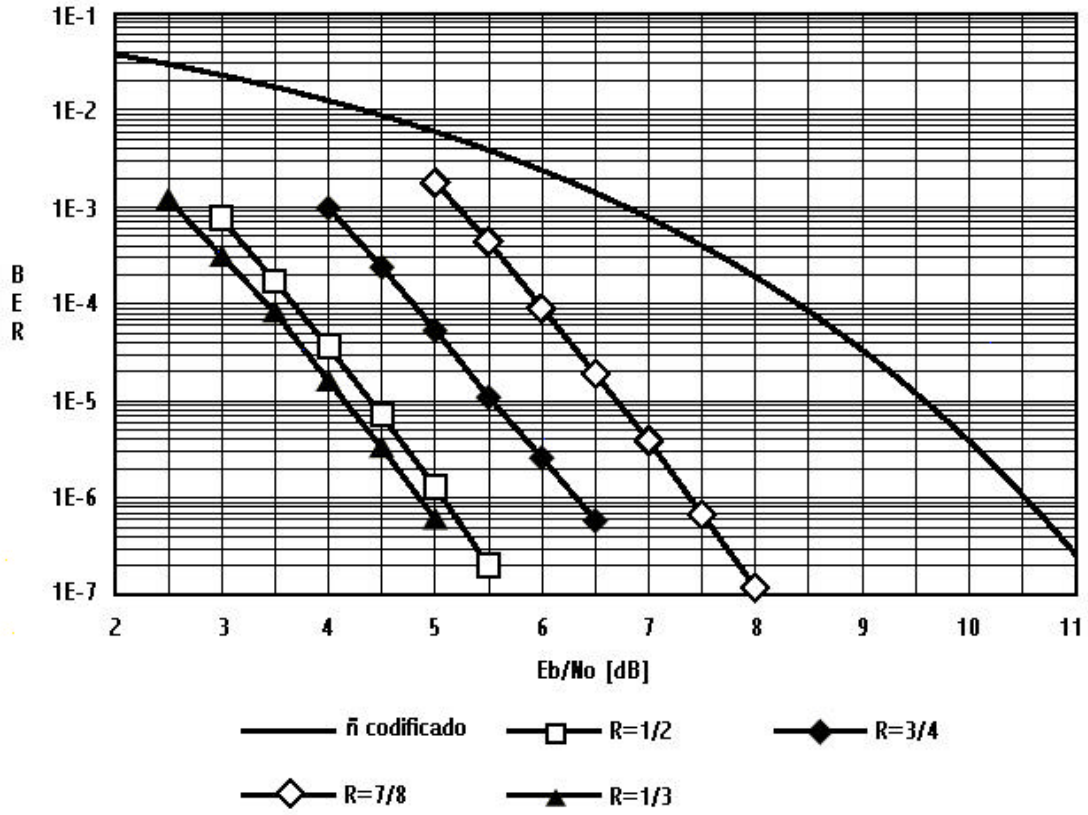


Figura B.10 - Variação do BER para diversas taxas de codificação.

Como se pode concluir após uma análise cuidada deste gráfico, é possível obter valores de ganho de descodificação de até 5,5 dB quando o circuito é utilizado com uma taxa de codificação de 1/3, e valores de BER de 10^{-5} para modulações de BPSK ou QPSK com entradas configuradas para o modo de "decisão ligeira"[34].

O ganho de descodificação apresenta um valor de cerca de 4,2 dB quando o circuito é utilizado nas mesmas configurações mas operando com uma taxa de codificação de 3/4.

B.4 MONITOR DA TAXA DE BITS ERRADOS - BER

O monitor de BER permite, em qualquer instante, efectuar a medida de eficiência do codificador de Viterbi. Este bloco funciona segundo o esquema de recodificação dos dados descodificados e respectiva comparação com os símbolos anteriormente recebidos e devidamente atrasados, ilustrado na Figura B.11. Assim, o monitor indicará a ocorrência de um erro sempre que esta comparação falhar. Para além disso, indicará também a ocorrência de um erro quando o descodificador não conseguir corrigir um bit correctamente. Contudo, nota-se que a probabilidade de este último

[33] Do inglês *Bit Error Rate.*
[34] Do inglês *Soft Decision.*

descodificar um bit incorrectamente é, segundo o fabricante, pelo menos duas ordens de grandeza inferior à probabilidade de ocorrência de um bit errado no canal. Assim, verifica-se que o efeito dos erros do descodificador assume uma influência desprezável na medida do BER total.

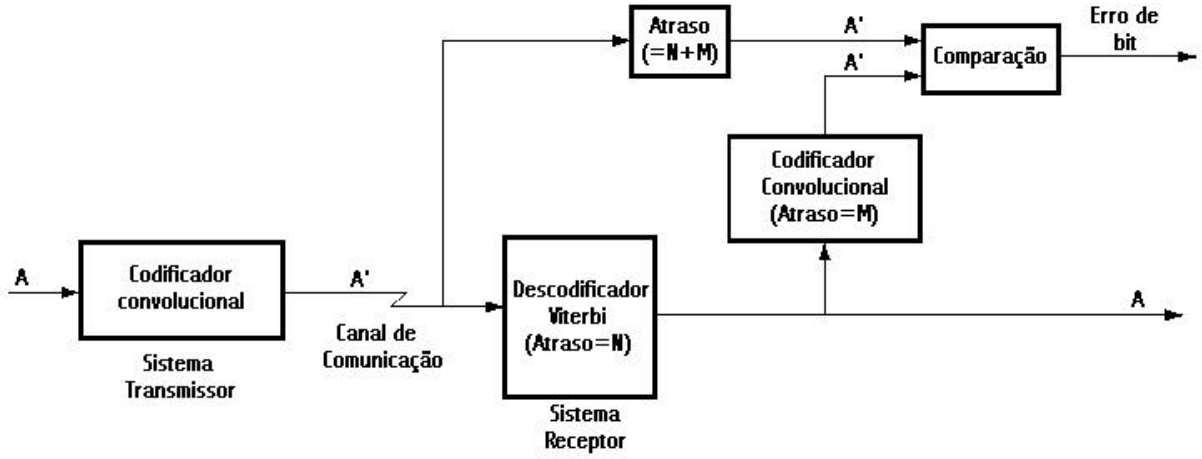


Figura B.11 - Esquema funcional do circuito de recodificação e comparação.

A saída deste circuito de recodificação e comparação pode ser usada para efeitos de medida do BER usando o circuito inserido neste módulo. Este circuito consiste em dois acumuladores funcionando como contadores, tal como se mostra na Figura B.12.

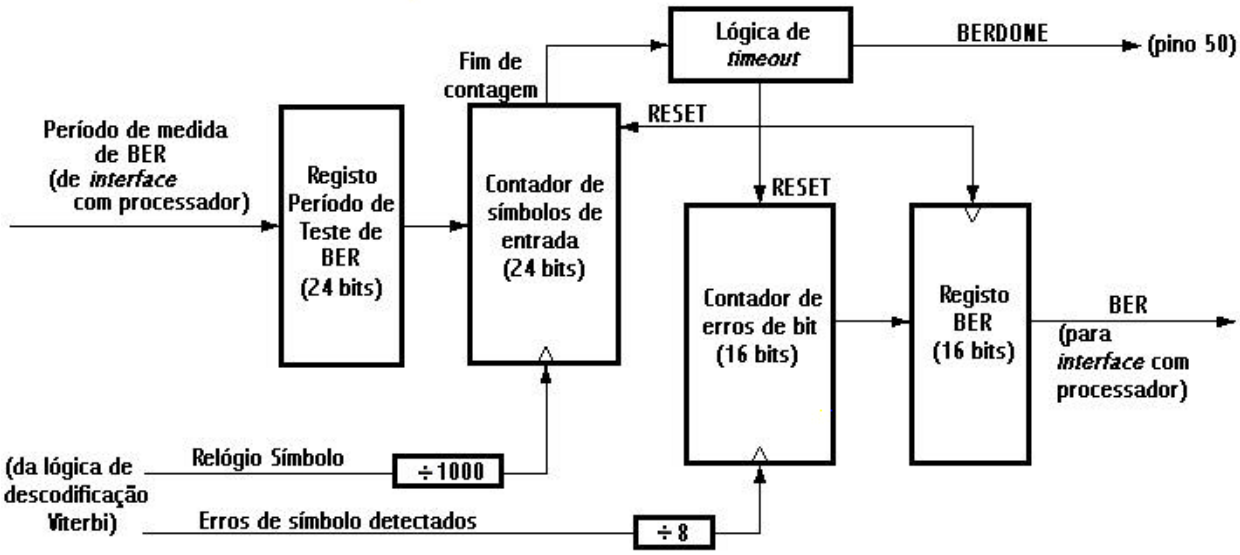


Figura B.12 - Esquema funcional do monitor de BER.

O primeiro acumulador efectua a contagem do números de símbolos de entrada no descodificador, ao passo que o segundo efectua a contagem de bits errados detectados pelo circuito de recodificação e comparação. Para além deste acumuladores, existe ainda um registo onde é possível programar o número de períodos de relógio de bit durante o qual se pretende efectuar a medida do BER. Na realidade, e por forma a diminuir a dimensão do contador, efectua-se uma divisão intermédia da frequência de bit por 1000, pelo que o registo referido terá a indicação do número de bits de entrada/1000 que corresponderão ao período de medidas do BER. Esta configuração pode ser efectuada por intermédio do interface deste circuito com o processador.

Assim, a medida do BER é calculada sempre que ocorre um impulso de *clock* DECOUTCLK, em que os bits errados vão sendo contados no acumulador respectivo. Este acumulador de 16 bits deve ser inicializado a zero no início do período de medida do BER.

Desta forma, logo que o valor de 24 bits corresponde ao período de medida do BER é introduzido, o circuito inicia a sua contagem, terminando quando o acumulador de bits de entrada atingir este valor. Neste instante, o valor da contagem de bits errados entretanto detectados é passado para um registo de 16 bits, ficando assim disponível para uso futuro por um processador. Além disso, sempre que termina uma medida do BER, esta é assinalada ao circuito exterior através do sinal BERDONE, que pode ser usado para desencadear uma interrupção no processador ou para sistemas de *polling* mais simples.

Assim, o valor realmente medido da contagem de bits errados poderá ser determinado através da seguinte fórmula:

$$Contagem\,de\,Bits\,Errados = (Valor\,Lido - 1) \times 8 \tag{B1}$$

O valor do BER é determinado dividindo o valor anterior pelo número total de períodos de bit preenchidos no registo de configuração multiplicado por 1000 (factor de escala):

$$Contagem\,de\,Bits\,Errados = \frac{(Valor\,Lido - 1) \times 8}{N^{o}\,de\,Períodos\,de Bit \times 1000} \tag{B2}$$

Por forma a obter medidas precisas, o fabricante indica que o período de teste deverá ser configurado por forma a obter um mínimo de cerca de 100 erros detectados no final da medida, evitando assim valores de variância elevados. Assim, se o número de erros detectados for inferior a 100 o período de teste deverá ser aumentado até que esta situação deixe de se verificar.

B.5 CONTROLADOR DE SINCRONISMO

O controlador de sincronismo é utilizado para controlar o estado de sincronismo do descodificador e para permitir uma monitorização do desempenho do mesmo. Assim, cabe ao projectista do sistema determinar um nível aceitável para a taxa de correcção de sincronismo, especificando-a nos registos de controlo apropriados. Estes registos permitem o controlo simultâneo do período de medida desta taxa e do número de correcções permitidas dentro deste intervalo de tempo. Assim, se este número limite de correcções não for excedido durante o período de teste, o sinal de saída INSYNC indicará que o descodificador se encontra sincronizado. Se este limite for ultrapassado durante o período de medida, o sinal OUTOFSYNC indicará a detecção de perda de sincronismo. O fabricante refere ainda que este sinal deverá ser usado para actuar um *flip-flop* tipo D externo, cuja saída constituirá, assim, um sinal de correcção a ser aplicado na entrada SYNCCHNG. Desta forma, o descodificador auto-sincronizar-se-á através da alteração do estado de sincronismo.

Na Figura B.13 apresenta-se a constituição deste circuito de controlo, composto por dois acumuladores actuando como contadores, cujo período é programado através de registos. O primeiro contador (T) mede o número de bits descodificados divididos por um factor de escala igual a 256. O segundo contador (N) mede o número de correcções de sincronismo efectuadas durante o período de teste dado pelo contador T. Assim, a taxa de correcção de sincronismo é dada pela razão entre o número de correcções efectuadas (dado pelo contador N) e o período de medida (dado pelo contador T). A inicialização destes registos de controlo é feita pela *interface* do microprocessador, com a ressalva de que o tempo de medida é, devido ao factor de escala referido, dado pelo produto

do valor de inicialização de T por 256: $t = 256 \times T$, sendo t o número real de bits descodificados durante o período de medida.

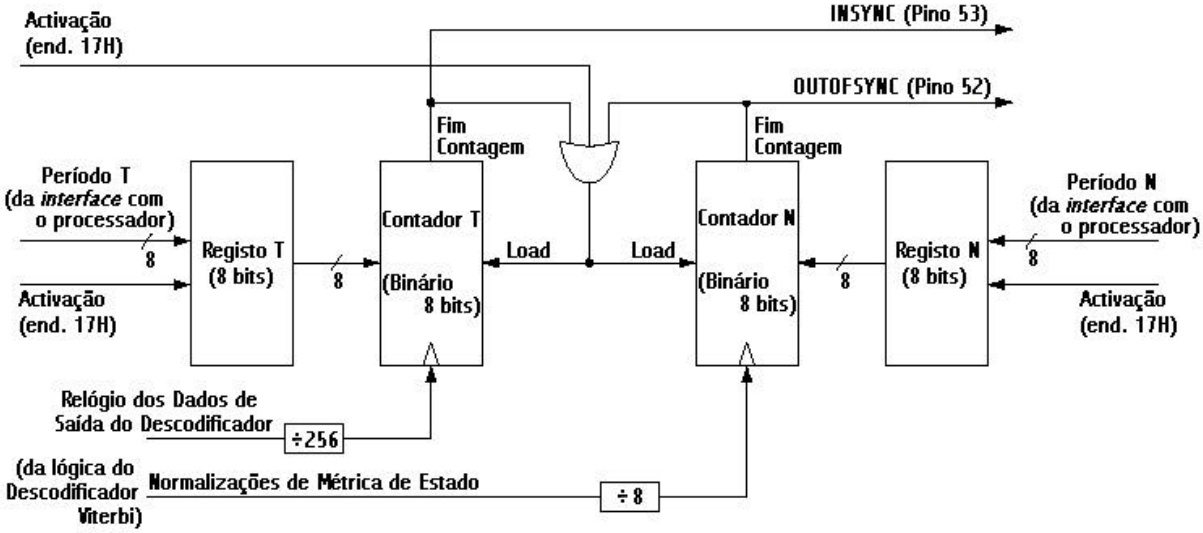


Figura B.13 - Controlador de sincronismo.

O número efectivo de correcções verificadas (n) é determinado pela seguinte fórmula: $n = (N-1) \times 8 + 4$, em que N é o valor inscrito no registo de controlo.

Através destes registos, torna-se assim possível a programação do valor limite que estabelece a fronteira da taxa de correcção entre estados de sincronismo e de dessincronismo, dada pela fórmula:

$$Taxa\,de\,Correcção = \frac{(N-1) \times 8 + 4}{256 \times T} \tag{B3}$$

O fabricante estabelece alguns valores indicadores para esta taxa, sendo de 10% para a taxa de codificação de 1/2, 1,7% para a taxa de 3/4 e 0,8% para a taxa de codificação de 7/8. O fabricante indica ainda que, por forma a que esta medida se processe com um valor de precisão razoável, é conveniente detectar entre 20 a 30 indicações de correcção de sincronismo antes de ser definido o estado de perda de sincronismo.

B.6 FORMATO DOS DADOS DE ENTRADA

O descodificador de Viterbi tem dois formatos de dados de entrada possíveis:

- Formato de decisão ligeira (*Soft Decision*) de 3 bits, correspondente às entradas R0[0..2], R1[0..2] e R2[0..2] e que pode ser programado segundo as configurações de sinal/amplitude ou representação binária natural, de acordo a Tabela B.1:
- Formato de decisão forçada[35] de apenas 1 bit em que, neste caso, as entradas R0[0..2], R1[0..2] e R2[0..2] deverão ser configuradas para a notação sinal/amplitude. Assim as entradas Rx[0] (LSB de amplitude) deverão ser ligadas a '1' lógico. As entradas Rx[1] (MSB de amplitude) deverão ser ligadas a '0' lógico. Finalmente, as entradas Rx[2] (bits de sinal) deverão ser ligadas à linha que constitui a entrada de dados digitais.

[35] Do inglês *Hard Decision.*

	Formato de Codificação					
	Binário Natural			Sinal/Amplitude		
Bit R0[x], R1[x], R2[x]	**[2]**	**[1]**	**[0]**	**[2]**	**[1]**	**[0]**
'1' mais forte:	1	1	1	1	1	1
	1	1	0	1	1	0
	1	0	1	1	0	1
'1' mais fraco:	1	0	0	1	0	0
'0' mais fraco:	0	1	1	0	0	0
	0	1	0	0	0	1
	0	0	1	0	1	0
'0' mais forte:	0	0	0	0	1	1

Tabela B.1 - Configurações para formato de decisão ligeira.

Na Figura B.14 ilustra-se a diferença do desempenho deste circuito dependente de se usar entradas com decisão forçada ou decisão ligeira.

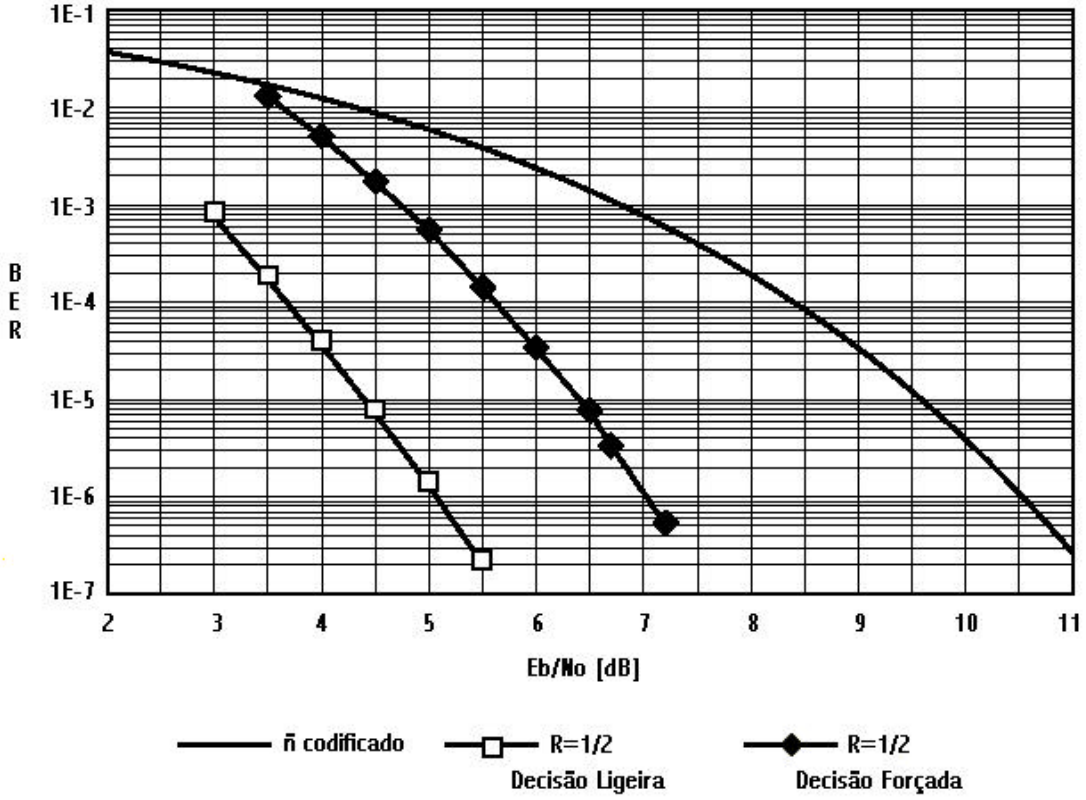


Figura B.14 - Desempenho do sistema para os modos de "decisão ligeira" e "decisão forçada".

B.7 INICIALIZAÇÃO DO CIRCUITO

O circuito integrado Q1900 possui duas entradas distintas de inicialização do codificador e do descodificador. O fabricante refere que a inicialização deverá ser efectuada após terem sido escritos todos os registos de controlo que definem o funcionamento do circuito.

A inicialização poderá ser feita através da ligação a '1' lógico das entradas ENCRESET e DECRESET ou através de registos de controlo na interface com o microprocessador. Contudo, refere-se que toda a operação de inicialização deverá ser efectuada após terem sido estabelecidos os vários relógios no circuito, visto esta processar-se de um modo síncrono no flanco ascendente do relógio.

A inicialização, tanto do codificador como do descodificador coloca todos os estados e registos internos a '0' lógico.

B.8 ATRASO DE CODIFICAÇÃO/DESCODIFICAÇÃO

O atraso de codificação/descodificação do codificador e do descodificador dependem do modo e da taxa de codificação definidos. De facto, os únicos modos para os quais o atraso é fixo são os correspondentes às taxas de 1/2 e 1/3 com entradas e saídas em paralelo.

Assim, o atraso de codificação para as taxas de 1/2 e 1/3 será de 10 ½ ciclos de ENCCLOCK, ao passo que o atraso de descodificação será de 102 ½ ciclos de DECCLOCK para o funcionamento de memória reduzida e de 182 ½ ciclos de DECCLOCK para o modo de funcionamento de memória estendida.

O atraso correspondente a estas taxas de codificação para o modo de configuração com entradas/saídas série varia cerca de ±4 períodos do relógio DECCLOCK dos valores definidos atrás para o modo paralelo.

O atraso correspondente às taxas de codificação de 3/4 e 7/8 em modo paralelo variam de cerca de ±4 períodos do relógio DECCLOCK dos valores mencionados em cima para as configurações de memória reduzida e de memória estendida.

B.9 MODOS DE FUNCIONAMENTO

Tal como se referiu anteriormente, este circuito integrado pode ser utilizado em dois modos de funcionamento distintos: *Modo Série* e *Modo Paralelo*. Nas figuras Figura B.15 e Figura B.16 ilustram-se estes modos de funcionamento.

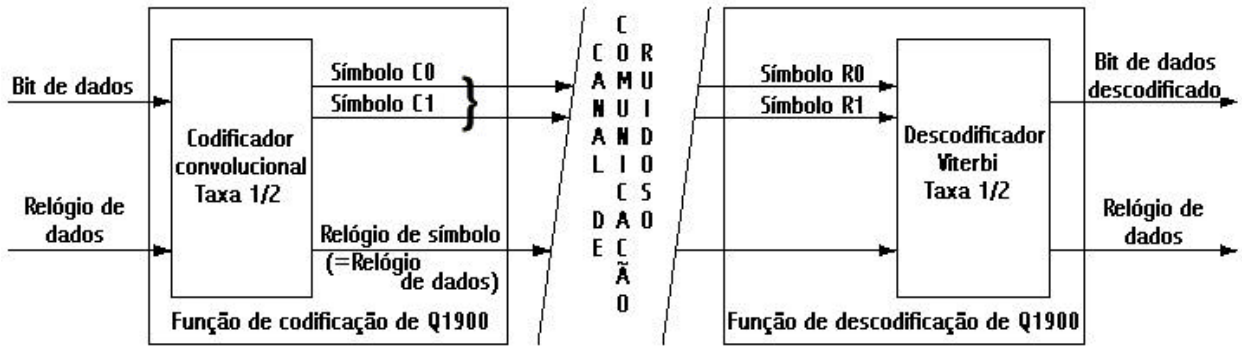


Figura B.15 - Modo de funcionamento paralelo.

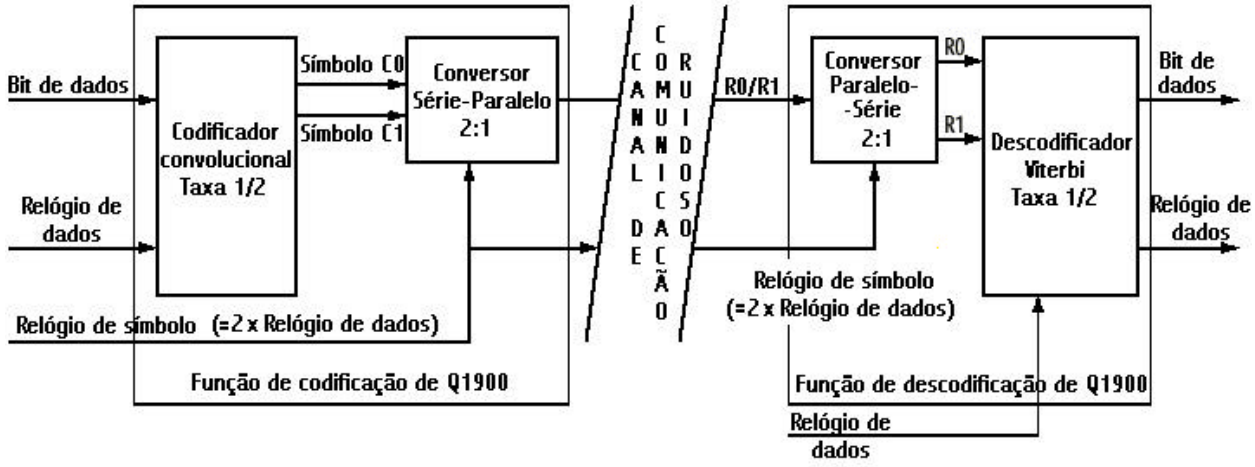


Figura B.16 - Modo de funcionamento série.

As taxas de codificação de 1/2 e de 1/3 podem ser operadas tanto em modo série como em modo paralelo, ao passo que as taxas de codificação de 3/4 e de 7/8 apenas podem ser utilizadas no modo de funcionamento paralelo.

Qualquer que seja a taxa de codificação R, o codificador debitará 1/R bits na sua saída por cada bit de entrada. No modo de funcionamento paralelo os dados são disponibilizados em paralelo para o circuito de modulação. Por forma a permitir o funcionamento tanto no modo série como no modo paralelo, é necessário fornecer dois relógios distintos, tendo um deles um período correspondente à taxa de bits de entrada (ENCDATIN) e o outro um período correspondente à taxa de símbolos nas saídas (C_0, C_1 e C_2).

B.10 DESCRIÇÃO DO MODO DE FUNCIONAMENTO UTILIZADO

No presente projecto optou-se por utilizar este circuito integrado no modo de funcionamento paralelo e taxa de codificação de 3/4. Esta opção teve como base o compromisso entre o ganho de codificação disponibilizado por este circuito com benefícios ao nível da taxa de erros de bit, e a sobrecarga que este código implica devido ao aumento da redundância da informação transmitida.

Quando este circuito é utilizado neste modo de funcionamento são gerados quatro bits codificados (dois pares C0-C1) pelo codificador por cada três bits de informação. Estes bits codificados são disponibilizados pelas saídas C0 e C1 a uma frequência de cerca de 2/3 da frequência de entrada.

Da mesma forma, quatro bits codificados (dois pares R0-R1) são necessários no descodificador por cada três bits descodificados na saída.

Como foi referido nas secções anteriores, a operação neste modo de funcionamento implica o uso de um padrão de eliminação de símbolos. Contudo, este facto não traz grandes diferenças na eficiência, à excepção de um pequeno aumento nos estados de sincronização, devido à ambiguidade inicial do padrão de eliminação de símbolos.

Quando este circuito é utilizado com a opção de sincronização automática, efectua, em primeiro lugar, o processo de resolução de possível erro de fase. De seguida, os pares de símbolos codificados (R0, R1) são processados pelo circuito de inserção de símbolos nulos utilizando o mesmo padrão utilizado no codificador. Assim, este processo de sincronização tem como consequência o aumento dos estados de sincronização por um factor de dois, resultando assim em

quatro estados correctos possíveis. O descodificador irá tentar cada um destes estados sequencialmente, efectuando uma mudança de estado sempre que a entrada SYNCCHNG for activada.

Na Figura B.17 é ilustrada a configuração do circuito integrado Q1900 como codificador, com o modo de funcionamento descrito anteriormente. Os dados de entrada são lidos pelo codificador no flanco ascendente do relógio ENCINCLK. Os dados de saída codificados são sincronizados pelo flanco ascendente do relógio ENCCLKOUT, que deve ter uma frequência de cerca de 2/3 da frequência do relógio ENCINCLK. O primeiro par de símbolos codificados na saída do codificador são indicados pela activação da saída C2.

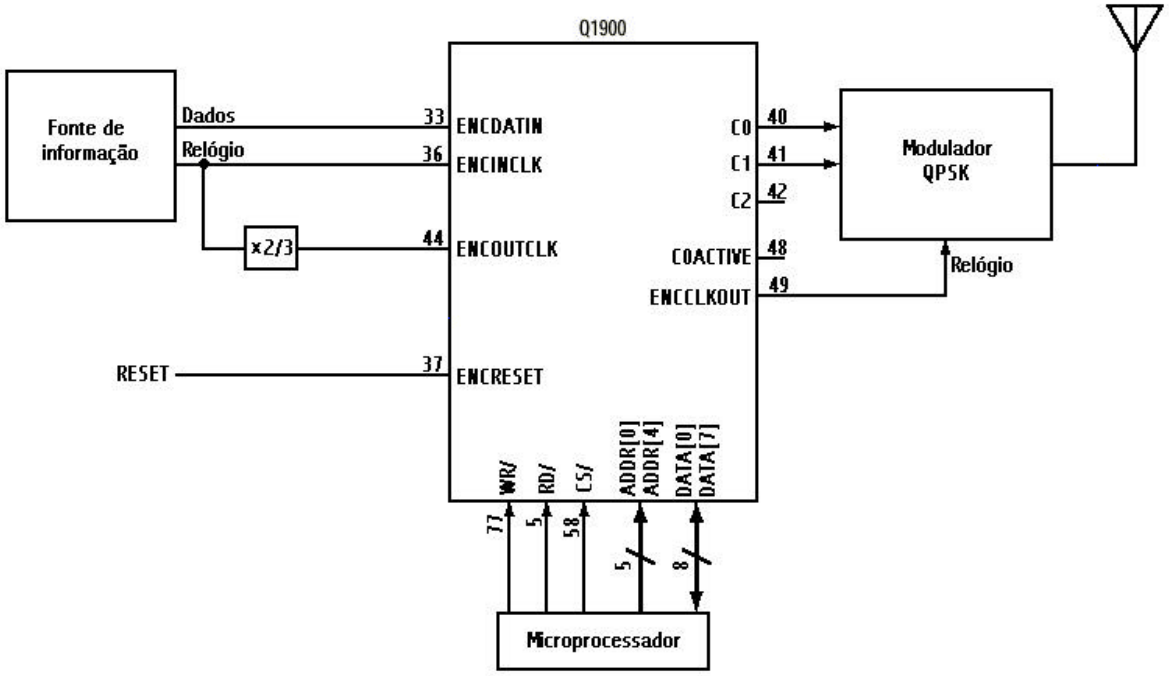


Figura B.17 - Utilização no modo de codificação paralelo com taxa de 3/4.

Na Figura B.18 é ilustrada a configuração do circuito descodificador utilizando o modo de funcionamento descrito. Os dados de entrada são lidos pelo descodificador no flanco ascendente do relógio DECINCLK. Os dados de saída descodificados são sincronizados pelo flanco ascendente do relógio DECCLKOUT, que deve ter uma frequência de cerca de 3/2 da frequência do relógio DECINCLK.

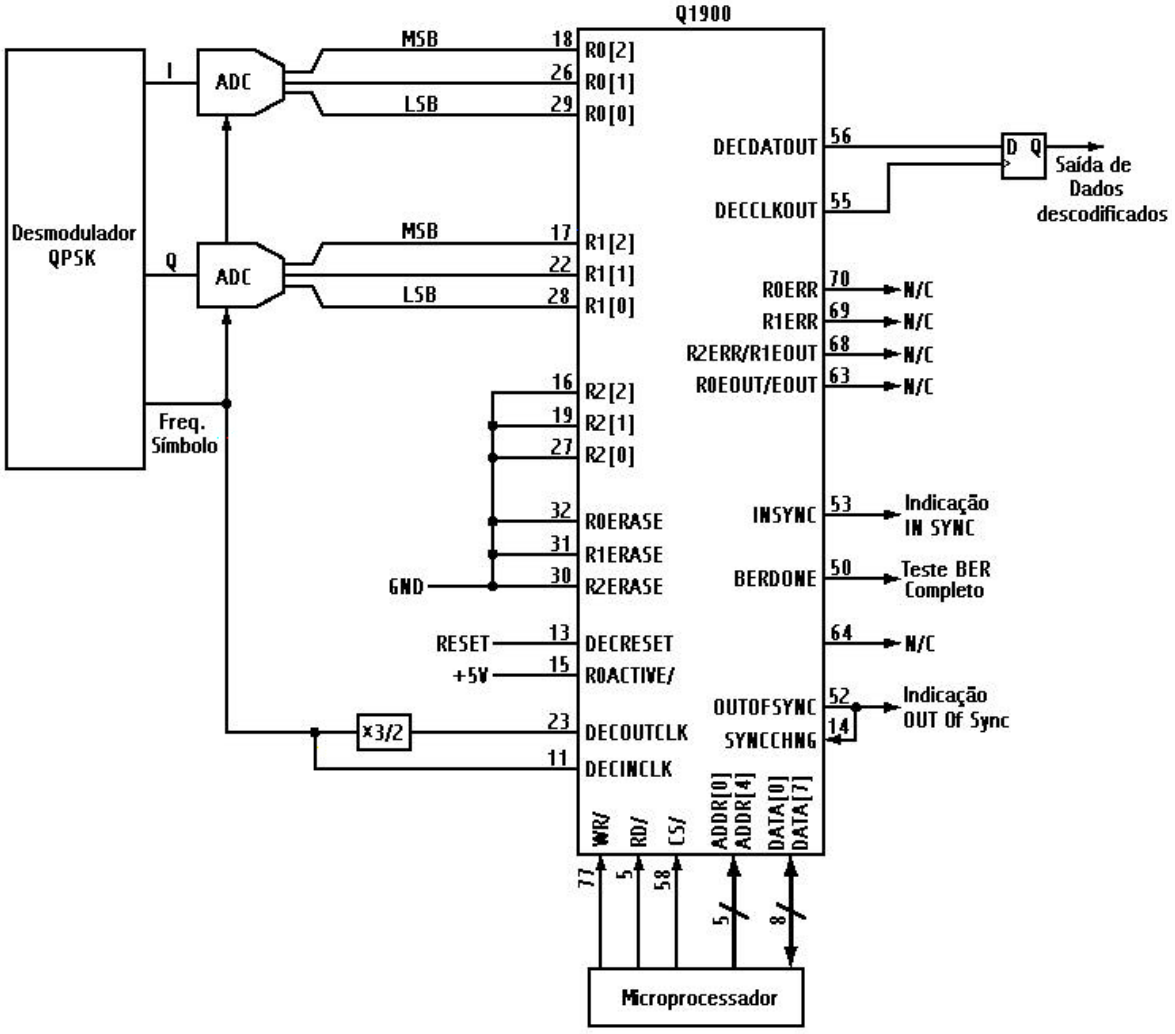


Figura B.18 - Utilização no modo de descodificação paralelo com taxa de 3/4.

Na tabela seguinte ilustra-se um exemplo de uma possível inicialização do circuito integrado utilizando o modo de funcionamento descrito. Os valores constantes nesta tabela foram utilizados para programar cada um dos registos correspondentes a cada endereço, utilizando a interface com o microcontrolador.

Passo	Nome do Registo	End.	Valor	Observações
1	Reservado	15H	00H	Devem conter 00H para correcto funcionamento
2	Reservado	16H	00H	
3	Controlo do descodificador 1	02H	08H	Modo Paralelo, Desmod. QPSK, Taxa ¾
4	Controlo do descodificador 2	03H	01H	Sinal/Amplitude, Modo Dados Directos, sem codificador diferencial nem *descrambler*
5	Controlo do descodificador 3	04H	01H	Modo Memória Longa
6	Registo de entrada do contador T do teste de normalização	08H	F4H	T=1,7%
7	Registo de entrada do contador N do teste de normalização	09H	F9H	N=1,7%
8	Período de BER (Byte menos significativo)	0AH	FCH	*Byte* menos significativo do valor de período do monitor de BER (24 bits)
9	Período de BER (Byte central)	0BH	FFH	*Byte* central …
10	Período de BER (Byte mais significativo)	0CH	FFH	*Byte* mais significativo …
11	Activação do valor de teste de normalização	17H	00H	Escrita para este registo começa teste de normalização
12	Activação do valor de teste de BER	18H	00H	Escrita para este registo começa teste de BER
13	Controlo do codificador 1	06H	08H	Modo Paralelo, Taxa ¾
14	Controlo do codificador 2	07H	00H	Modo Dados Directos, sem codificador diferencial nem *scrambler*
15	Controlo do descodificador 3	04H	05H	Modo Memória Longa, *reset* do descodificador
16	Controlo do codificador 1	06H	0AH	Modo Paralelo, Taxa ¾, *reset* codificador
17	Controlo do descodificador 3	04H	01H	Modo Memória Longa, fim de *reset* do descodificador*
18	Controlo do codificador 1	06H	08H	Modo Paralelo, Taxa ¾, fim de r*eset* do codificador**

* - Após um mínimo de dois ciclos dos relógios DECINCLK e DECOUTCLK.

** - Após um mínimo de dois ciclos dos relógios ENCINCLK e ENCOUTCLK.

Tabela B.2 - Valores dos registos de programação para uma operação em modo paralelo com taxa de codificação de 3/4.

C Listagem dos programas que implementam o codificador H.263

C.1 FICHEIRO "init.asm"

```
*******************************************************************************
*                          TRABALHO FINAL DE CURSO                            *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "init.asm"                                                        *
* DESCRIÇÃO: Este ficheiro contem o conjunto de procedimentos que constituem a *
*            inicializacao do Processador Digital de Sinal (DSP) TMS320C50.   *
*            Embora nao se utilize qualquer mecanismo suportadoem interrupcoes,*
*            optou-se por definir desde logo a tabela de interrupcoes corres-  *
*            pondente por forma a preencher o espaco de memoria correspondente *
*            com a uma instrucao de salto incondicional para uma funcao 'nula'.*
*            Na rotina de inicializacao propriamente dita e' tambem feita a    *
*            inicializaçao dos registos convenientes por forma a colocar o DSP *
*            no modo de funcionamento desejado.                                *
*******************************************************************************


*******************************************************************************
*    Tabela de Interrupcoes
*******************************************************************************

        .sect "vectors"         ; Endereco 00h da memoria de programa
RESET   B INIT                  ;RESET  (RESET)
        B NONE                  ;INT1   (EXTERNA MASCARAVEL)
        B NONE                  ;INT2   (EXTERNA MASCARAVEL)
        B NONE                  ;INT3   (EXTERNA MASCARAVEL)
        B NONE                  ;TINT   (TIMER)
        B NONE                  ;RINT   (RECEPCAO SERIE)
        B NONE                  ;XINT   (TRANSMISSAO SERIE)
        B NONE                  ;TRNT   (RECEPCAO TDM)
        B NONE                  ;TXNT   (TRANSMISSAO TDM)
        B NONE                  ;INT4   (EXTERNA MASCARAVEL)

        .space (14*16)          ;Zona de memoria reservada

        B NONE                  ;TRAP   (TRAP SOFTWARE)
        B NONE                  ;NMI    (EXTERNA NAO MASC)

        .space (2*16)           ;Zona de memoria reservada


*******************************************************************************
* ROTINA: "INIT"                                                              *
* DESCRICAO: Rotina de Inicializacao do Processador                           *
*            Inicializa:                                                      *
*               Estrutura de interrupcoes (INTM, IMR, IPTR)                   *
*               Controlo de modo (OVM, SXM, PM, AVIS, NDX, TRM)               *
*               Controlo de memoria (RAM, OVLY, CNF)                          *
*               Wait States (PDWSR, IOWSR, CWSR)                              *
*            Apos inicializar as variaveis do DSP procede 'a transferencia do *
*            programa para memoria interna e executa um salto para a rotina   *
*            "START".                                                         *
*******************************************************************************

        .sect   "ini"

INIT:   setc INTM               ; Inactiva as interrupcoes
        ldp #0                  ; pagina 0

        apl #0000h,IMR          ; Desactiva todas as interrupcoes

        setc OVM
        spm 1                   ; deslocamento do registo P de  bit para a esquerda
        apl #0008h,PMST
        opl #3eH,PMST           ; RAM=1, OVLY=0, TRM=1, NDX=1
        clrc CNF

   ; Programacao dos Wait States
```

```
        splk #00000H,PDWSR      ; 0WS para a memoria
        splk #00000H,IOWSR      ; 0WS para I/O
        splk #0001FH,CWSR       ; BIG=1,  mode=1

   ; Inicializacao da pagina 0

        larp AR1                ; inicializa ARP -> AR1
        lar AR1,#96             ; carrega AR1 com o endereco inicial da pagina 0
        zap
        rpt #31                 ; repete 32 vezes
        sacl *+                 ; inicializacao de todas as posicoes com 'zero'

   ; Inicializacao das paginas 2 a 9

        larp AR1
        lar AR1,#256            ; carrega AR1 com o endereco inicial da pagina 2
        zap
        rpt #(128*8-1)
        sacl *+                 ; limpa todas as posicoes das paginas 2 a 9

   ; Carregamento do programa da RAM externa para a RAM interna

        larp AR1
        lar AR1,#prog_run
        rpt  #(prog_end-prog_start-1)
        blpd #prog_start, *+

        apl #0008h,PMST
        opl #01eH,PMST          ; RAM=1, OVLY=0, TRM=1, NDX=1

        B START

        .text
        .label prog_start
prog_run:
*******************************************************************************
* ROTINA DE INTERRUPCAO (nao utilizada)                                       *
*******************************************************************************
NONE:   rete
```

C.2 FICHEIRO "`main.asm`"

```
*******************************************************************************
*                          TRABALHO FINAL DE CURSO                            *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "main.asm"                                                        *
* DESCRIÇÃO: Este ficheiro ocupa o topo de toda a hierarquia de ficheiros, vis-*
*            to que engloba a rotina principal do programa.                   *
*            Este ficheiro e' executado apos a inicializacao do DSP. Nele, cada*
*            DSP comeca por invocar as varias rotinas que inicializam alguns   *
*            dos blocos do codificador, tais como blocos das transformadas DCT *
*            e IDCT  ("tranini") e do codificador de comprimento variavel      *
*            ("vlc_ini"). De seguida, e' realizada a actualizacao do passo de  *
*            quantificacao a utilizar, dependendo do nivel actual do buffer de *
*            emissao. Determina-se tambem se a imagem a codificar devera' ser  *
*            do tipo INTRA ou do tipo INTER. Para finalizar, e' efectuado o    *
*            preenchimento dos registos da FPGA2, que determinarao a forma como*
*            se ira' processar a transferencia dos dados para a memoria do DSP *
*            atraves de um ciclo de DMA.                                       *
*******************************************************************************
*
        .title "main.asm"
        .mmregs
        .version 50
*
        .include "mem_org.h"
        .include "init.asm"
        .include "transf.asm"
        .include "vlc.asm"
        .include "MBLayer.asm"
        .include "GOBLayer.asm"
        .include "PICLayer.asm"
```

```
        .include "dma_ini.asm"

        .include "pic_ini.asm"      ;-> para tirar depois!!!!
*
LIMITE    .set    PA0
TARG_OFF  .set    PA1
nCOR      .set    PA2
RES       .set    PA3
PORTO_OUT .set    PA4
COUNTER   .set    PA5
OUT2      .set    PA6
OUT3      .set    PA7
*
        .text
*
START:  call traini
        call vlc_ini
        call dma_ini

        ;+++++++++++++++++++++++++++++++++++++++
        ; Inicializacao do passo de quantificacao +
        ;+++++++++++++++++++++++++++++++++++++++
detIP:  ldp #CountINTER
        lacl MaxINTER
        sub CountINTER
        bcnd Pic_P,GT

Pic_I:  mar *,ar1               ; inicialização da imagem anterior a zero
        lar ar1,#OG0M1B1
        rptz 5280h              ; 6*64*11*5 = 21120 = 5280h
        sacl *+

        zac
        bd detPQDC
        sacl CountINTER         ;                       <-- Branch Delayed
        sacl Pic_PnI            ;                       <-- Branch Delayed

Pic_P:  lacl CountINTER
        add #1
        sacl CountINTER
*       bd detPQDC
        lacl #1                 ;                       <-- Branch Delayed
        sacl Pic_PnI            ;                       <-- Branch Delayed

detPQDC:
        mar *,ar1
        lar ar1,#pqdc           ; Inicializacao do passo de quant. DC
        lacl #3
        sacl *+,ar1

detPQnDC:
        lamm COUNTER            ; Inicializacao do passo de quant. nao DC
        nop
        sacb
        sub #64                 ; 0k < COUNTER < 64k ==> pqndc=2 (passoQ = 4)
        bcndd codPIC,LT
        lacl #2                 ;                       <-- Branch Delayed
        sacl *,ar1              ;                       <-- Branch Delayed

        lacb
        sub #128                ; 64k < COUNTER < 128k ==> pqndc=3 (passoQ = 8)
        bcndd codPIC,LT
        lacl #3                 ;                       <-- Branch Delayed
        sacl *,ar1              ;                       <-- Branch Delayed

        lacb
        sub #252                ; 128k < COUNTER < 252k ==> pqndc=4 (passoQ = 16)
        bcndd detPQnDC,GEQ      ; COUNTER > 252k ==> Espera !!!!
        lacl #4                 ;                       <-- Branch Delayed
        sacl *,ar1              ;                       <-- Branch Delayed

        ;+++++++++++++++++++++++++++++++++++++++
        ; inicializacao do registo Limite da FPGA2 +
        ;+++++++++++++++++++++++++++++++++++++++
codPIC: lacc #5600h
        samm LIMITE
        call delay
*       >>>>>> ciclo de DMA entre FPGA2 -> C50 <<<<<<
```

```
        call PIC_Layer

        b detIP

        .label prog_end
*
        .end
*
```

C.3 FICHEIRO "PIC_Layer.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "PIC_Layer.asm"                                                    *
* DESCRIÇÃO: Este modulo faz o tratamento dos dados correspondentes a cada     *
*            'Picture'.                                                        *
*******************************************************************************

        .text
*
PIC_Layer:
        lacc #end_pic
        samm ARCR           ; ARCR usado com a instrucao CMPR
        calld dct
        lar ar1,#NG0M1B1

        lar ar1,#NG0M1B1    ; ponteiro para o inicio do resultado da DCT
        lar ar6,#OG0M1B1    ; ponteiro para o destino da iDCT (a processar)
        lar ar2,#buf_vlc    ; ponteiro para o buf_vlc (necessario para o
        lar ar4,#CBPC       ; inicio)
        mar *,ar2
************
** HEADER **
************
    ;++++++++++++++++++++++++++++++++
    ;+ Campo Picture Start Code - PSC +
    ;++++++++++++++++++++++++++++++++
        zac
        sacl *+
        lacl #16
        sacl *+
    ;++++++++++++++++++++++++++
    ;+ Campo PSC (6 LSBs) & TR +
    ;++++++++++++++++++++++++++
        ldp #Tem_Ref
        lacl Tem_Ref
        add #1
        and #00ffh          ; modulo 256
        sacl Tem_Ref
        or #20h,8
        sacl *+,2
        lacl #14
        sacl *+
    ;+++++++++++++++
    ;+ Campo PTYPE +
    ;+++++++++++++++
        ldp #Pic_PnI
        lacc Pic_PnI,7
        or #8200h
        sacl *+
        lacl #13
        sacl *+
    ;++++++++++++++++++++++++++
    ;+ Campo PQUANT & CPM & PEI +
    ;++++++++++++++++++++++++++
        ldp #pqndc      ; Step= 2^pqndc;  Quant= Step/2= 2^(pqndc-1)
        lacc pqndc      ; PQUANT= (1 << Shift_x)
        add #3          ; Shift_x = 16[word] -5[PQUANT] +(pqndc-1)[Quant]
        samm TREG1      ;         = 16-5+(pqndc-1)= 10 + pqndc
        ldp #onevlc     ;         = (3 + pqndc)[lact] + 7[sacl]
        lact onevlc
        sacl *+,7
```

```
        lacl #7         ; NO. de bits = 5(PQUANT)+1(CPM)+1(PEI)=7
        sacl *+

        call PutBits
        lar ar2,#buf_vlc

        ldp #GOB_Nr
        zac             ; inicializacao de GOB_Nr a '0'
        sacl GOB_Nr     ; pondente ao 1. Macrobloco
***********
** GOB 0 **
***********
GOB0:   call GOB_Layer
***********
** GOB 1 **
***********
GOB1:   call GOB_Layer
***********
** GOB 2 **
***********
GOB2:   call GOB_Layer
***********
** GOB 3 **
***********
GOB3:   call GOB_Layer
***********
** GOB 4 **
***********
GOB4:   call GOB_Layer


********** ; Inserir o 'TAIL' só no processador B.
** TAIL **
**********
    ;+++++++++++++++++++++++++  ; insere entre 0 a 7 bits a '0' por forma
    ;+ Campo ESTUF (Stuffing) +  ; a alinhar a trama H.263 ao byte
    ;+++++++++++++++++++++++++
        mar *,ar2
        ldp #TREG1buf
        zac
        sacl *+
        lacl TREG1buf   ; TREG1buf tem o n. de bits de uma palavra de 16 bits
        and #0007       ; ainda nao utilizados. O numero de bits de stuffing
        sacl *+         ; necessarios corresponde a (TREG1buf mod 8) pelo que
                        ; basta colocar o bit 3 a zero para determinar quantos
                        ; bits de stuffinf sao necessarios.
    ;+++++++++++++++++++++++++++++++
    ;+ Campo End Of Sequence - EOS +
    ;+++++++++++++++++++++++++++++++
        zac
        sacl *+
        lacl #16
        sacl *+
        lacc #0fc00h
        sacl *+
        lacl #6
        sacl *+
    ;+++++++++++++++++++++++++  ; insere 2 bits a '0' por forma a alinhar
    ;+ Campo PSTUF (Stuffing) +  ; a trama H.263 ao byte
    ;+++++++++++++++++++++++++
        zac
        sacl *+
        lacl #2
        sacl *+

***************************************
** Bit 17 - Assinala o fim da sub-trama **
***************************************
        call PutBits
        lar ar2,#buf_vlc
*        mar *,ar2
*        ldp #TREG1buf
*        zac
*        sacl *+
*        lacl TREG1buf
*        sacl *+
*        setc xf                ; B17 = '1'
*        call PutBits
*        lar ar2,#buf_vlc
```

```
*        clrc xf               ; B17 = '0'


PIC_end  ret
```

C.4 FICHEIRO "GOB_Layer.asm"

```
*******************************************************************************
*                          TRABALHO FINAL DE CURSO                            *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "GOB_Layer.asm"                                                   *
* DESCRIÇÃO: Este modulo faz o tratamento dos dados correspondentes a cada GOB.*
*            E' executado apos se fazer o tratamento do cabecalho da "picture" *
*            e executa o modulo de tratamento dos macroblocos.                *
*******************************************************************************

        .text
*
GOB_Layer:
        ldp #GOB_Nr
        lacl GOB_Nr     ; no caso de se tratar do GOB numero zero, nao
        bcndd MB1,EQ    ; deve ser introduzido qualquer header, pelo que
        add #1          ; se deve enviar de imediato a informacao corres-
        sacl GOB_Nr     ; pondente ao 1. Macrobloco
************
** HEADER **
************
    ;+++++++++++++++++++++++++  ; insere entre 0 a 7 bits a '0' por forma
    ;+ Campo GSTUF (Stuffing) +  ; a alinhar a trama H.263 ao byte
    ;+++++++++++++++++++++++++
        mar *,ar2
        ldp #TREG1buf
        zac
        sacl *+
        lacl TREG1buf   ; TREG1buf tem o n. de bits de uma palavra de 16 bits
        and #0007       ; ainda nao utilizados. O numero de bits de stuffing
        sacl *+         ; necessarios corresponde a (TREG1buf mod 8) pelo que
                        ; basta colocar o bit 3 a zero para determinar quantos
                        ; bits de stuffinf sao necessarios.
    ;++++++++++++++++++++++
    ;+ Campo GBSC (16 MSB) +
    ;++++++++++++++++++++++
        zac             ; este campo e constituido por 16 bits a zero seguidos
        sacl *+         ; de um bit a 1, constituindo um total de 17 bits.
        lacl #16
        sacl *+
    ;++++++++++++++++++++++++++++++++++
    ;+ Campos GBSC (1 LSB) & GN & GFID +
    ;++++++++++++++++++++++++++++++++++
        ldp #GOB_Nr
        lacc GOB_Nr,2   ; campo GN
        sub #4          ; para compensar o incremento feito anteriormente
        or #0080h       ; bit 17 do campo GBSC
        ldp #Pic_PnI    ; usa-se o valor de Pic_PnI para o campo GFID, tor-
        or Pic_PnI      ; nando assim este campo diferente quando se usam
                        ; imagens INTRA ou INTER
        sfl             ; antes de escrever para o buf_VLC "shifta-se" o
        sacl *+,7       ; resultado 8 posicoes para a esquerda, tal como
                        ; requerido pela funcao PutBits.
        lacl #8
        sacl *+
    ;+++++++++++++++++
    ;+ Campo GQUANT +
    ;+++++++++++++++++
        ldp #pqndc      ; Step= 2^pqndc;  Quant= Step/2= 2^(pqndc-1)
        lacc pqndc      ; GQUANT= (1 << Shift_x)
        add #3          ; Shift_x = 16[word] -5[GQUANT] +(pqndc-1)[Quant]
        samm TREG1      ;         = 16-5+(pqndc-1)= 10 + pqndc
        ldp #onevlc     ;         = (3 + pqndc)[lact] + 7[sacl]
        lact onevlc
        sacl *+,7
        lacl #5
        sacl *+
```

```
********************
** MacroBloco MB1 **
********************
MB1:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB2 **
********************
MB2:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB3 **
********************
MB3:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB4 **
********************
MB4:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB5 **
********************
MB5:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB6 **
********************
MB6:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB7 **
********************
MB7:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB8 **
********************
MB8:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
********************
** MacroBloco MB9 **
********************
MB9:    call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
*********************
** MacroBloco MB10 **
*********************
MB10:   call MB_Layer
        call PutBits
        lar ar2,#buf_vlc
*********************
** MacroBloco MB11 **
*********************
MB11:   call MB_Layer
        call PutBits
        lar ar2,#buf_vlc

GOB_end  ret
```

C.5 FICHEIRO "MB_Layer.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "MB_Layer.asm"                                                     *
* DESCRIÇÃO: Este modulo faz o tratamento dos dados correspondentes a cada ma- *
*            crobloco. E' executado apos se fazer o calculo da DCT da imagem   *
*            corrente, e executa as funcoes correspondentes a: - VLC, QuantInv,*
*            IDCT e Putbits.                                                   *
*                                                                              *
*            Registos utilizados:     Ar0: No. de runs[vlc]; contador[invqnt]  *
*                                     Ar1: Coeficiente a tratar                *
*                                     Ar2: Buffer VLC[vlc]                     *
*                                     Ar3: Tab. de coef VLC[vlc]               *
*                                     Ar4: Tabela CBPC[invqnt]                 *
*                                     Ar5: Mascaras de CBPC                    *
*                                     Ar6: Destino de escrita[idct]            *
*                                     Ar7: Contador de multiplicacoes[idct]    *
*                                     TREG0: [vlc]                             *
*                                     TREG1: [vlc],[invqnt]                    *
*                                     INDEX: [vlc]                             *
*                                     AccB: [vlc]                              *
*******************************************************************************

        .text
*
MB_Layer:
        ldp #pqndc
        lar ar5,#CBPCMskP
        lacl #0ffh      ; carrega bit 4 de TREG1 com '1' para usar
        samm TREG1      ; a instrucao SATH, permitindo assim shift
                        ; right de 16 posicoes
************
** HEADER **
************
        ldp #Pic_PnI
        lacc Pic_PnI
        bcndd MB_P,NEQ
MP_I:                   ; -->> Imagem INTRA
        mar *,ar4
        lacc *,1,ar3    ; carrega CBPC multiplicada por dois para AccLow
                        ; para aceder 'a tabela MCBPC_I
    ;+++++++++++++++++
    ;+ Campos MCBPC +
    ;+++++++++++++++++
        and #6h         ; filtra apenas os bits 5&6 (Cb e Cr)
        add #MCBPC_I
        samm ar3        ; ar3 aponta para a tabela MCBPC_I
        nop             ; -> Pipeline STALL devido a 'samm'
        nop             ; -> Pipeline STALL devido a 'samm'
        lacl *+,ar2
        sacl *+,ar3
        lacl *,ar2
        sacl *+,ar4
    ;+++++++++++++++
    ;+ Campo CBPY +
    ;+++++++++++++++
        lacc *,ar3      ; carrega os bits 1,2,3,4 (Y1, Y2, Y3 e Y4) do
        sfr             ; codigo CBPC para AccLow dividido por dois
        and #1eh        ; para aceder 'a tabela CBPY

        add #CBPY
        samm ar3        ; ar3 aponta para a tabela CBPY
        nop             ; -> Pipeline STALL devido a 'samm'
        nop             ; -> Pipeline STALL devido a 'samm'
        lacl *+,ar2
        sacl *+,ar3
        lacl *,ar2
        bd BY1
        sacl *+,ar4
        lacc *,ar5

MB_P:                   ; -->> Imagem INTER
```

```
        bcnd MB_Pcod,NEQ
        mar *,ar2
        lacl #80h       ; coloca campo COD=1, indicando que o MB nao esta
        sacl *+         ; codificado e que nada mais e' transmitido
        lacl #1h        ; COD = 1 bit
        sacl *+,ar1
        adrk #192
        adrk #192       ; ar1= ar1+192+192= ar1+6*64    (6 blocos)
        retd
        mar *,ar4
        adrk #1         ; soma o ponteiro da tabela CBPC para o proximo MB

MB_Pcod:
    ;++++++++++++++++++++++
    ;+ Campos COD & MCBPC +
    ;++++++++++++++++++++++
        bsar 8
        and #6h         ; filtra apenas os bits 5&6 (Cb e Cr)
        add #MCBPC_P
        samm ar3        ; ar3 aponta para a tabela MCBPC_P
        nop             ; -> Pipeline STALL devido a 'samm'
        nop             ; -> Pipeline STALL devido a 'samm'
        lacl *+,ar2
        sfr             ; acrescenta 1 bit 'a esquerda (MSB) a zero que
        sacl *+,ar3     ; constitui o campo COD (=0)
        lacl *,ar2
        add #1          ; soma uma unidade para contar com o bit COD
        sacl *+,ar4
    ;+++++++++++++++
    ;+ Campos CBPY +
    ;+++++++++++++++
        lacc *,ar3      ; carrega os bits 1,2,3,4 (Y1, Y2, Y3 e Y4) do
        bsar 9          ; codigo CBPC para AccLow dividido por dois
        and #1eh        ; para aceder 'a tabela CBPY
        xor #1eh        ; nega os bits 1,2,3,4 do Acc   <- 2 ciclos !!!!
        add #CBPY
        samm ar3        ; ar3 aponta para a tabela CBPY
        nop             ; -> Pipeline STALL devido a 'samm'
        nop             ; -> Pipeline STALL devido a 'samm'
        lacl *+,ar2
        sacl *+,ar3
        lacl *,ar2
        sacl *+,ar2     ; dos campos MVD (=1) para indicar vectores nulo
    ;+++++++++++++++++++++++++
    ;+ Campos MVD_h & MVD_v +
    ;+++++++++++++++++++++++++
        lacc #0c000h    ; acrescenta 2 bits a 'um' que constituem os dois
        sacl *+         ; campos MVD com vectores nulos segundo as direccoes
                        ; horizontal e vertical
        lacl #2         ; duas unidades para contar 2 vezes com o bit MVD
        sacl *+,ar4
        lacc *,ar5
**************
** Bloco Y1 **
**************
BY1:    and *+,ar1
        bcndd Y1nNULL,NEQ
        and #0ffh       ;           -> Branch Delayed   2 ciclos!!!!!!!
    ;+++++++++++++++++
    ;+ Bloco Y1 nulo +
    ;+++++++++++++++++
Y1NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar6
        adrk #64        ; soma 64 a ar6
        bd BY2                              ; Funcao invQuant nula
        mar *,ar4                           ; Funcao iDCT nula
        lacc *,ar5
    ;++++++++++++++++++++
    ;+ Bloco Y1 nao nulo +
    ;++++++++++++++++++++
Y1nNULL bcndd Y1cCoef,NEQ
        nop             ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed
    ;++++++++++++++++++++++++++
    ;+ Bloco Y1 so com coef. DC +
    ;++++++++++++++++++++++++++
        call vlc_DC     ; Funcao VLC "fast"  - retorna com arp=ar4
        ldp #pqndc      ; Funcao invQuant "fast"
        lacc *,16,ar1
```

```
        bcndd Y1_DCP,GT ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                       -> Branch Delayed
        sub pqndc       ;                       -> Branch Delayed

Y1_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y1fIDCT,NEQ
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        lacl #128
        bd Y1fIDCT
        sacl *,3,ar1    ; 128*3=1024            -> Branch Delayed
        lacc *,13       ;                       -> Branch Delayed

Y1_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y1DCneg,LT
Y1DCpos sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd Y1fIDCT
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y1DCneg ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y1fIDCT adrk #64        ; limita-se a espalhar o coeficiente DC
        mar *,ar6       ; pelo resto do bloco
        rpt #63
        sach *+
        bd BY2
        mar *,ar4
        lacc *,ar5
    ;++++++++++++++++++++++++++++++++
    ;+ Bloco Y1 com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++
Y1cCoef call vlc        ; Funcao VLC "slow"  - retorna com arp=ar4
        ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16,ar1
        bcndd Y1coefP,GT
        lar ar0,#63     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

Y1coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y1next,NEQ
        lar ar0,#62     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y1coefP ;lar ar0,#63
Y1next  bcnd Y1Nzero,NEQ
        mar *+,ar0
        banzd Y1next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y1sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y1Nzero bcndd Y1coefn,LT
Y1coefp sfl             ; Acc=2*Level           -> Branch delayed
```

```
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y1next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y1sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y1coefn ;sfl            ; Acc=2*Level      -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y1next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

*       b Y1sIDCT
        sbrk #64

Y1sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar4
        lacc *,ar5

**************
** Bloco Y2 **
**************
BY2:    and *+,ar1
        bcndd Y2nNULL,NEQ
        and #0ffh       ;           -> Branch Delayed   2 ciclos!!!!!!!
    ;+++++++++++++++++
    ;+ Bloco Y2 nulo +
    ;+++++++++++++++++
Y2NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar6
        adrk #64        ; soma 64 a ar6
        bd BY3                              ; Funcao invQuant nula
        mar *,ar4                           ; Funcao iDCT nula
        lacc *,ar5
    ;++++++++++++++++++++
    ;+ Bloco Y2 nao nulo +
    ;++++++++++++++++++++
Y2nNULL bcndd Y2cCoef,NEQ
        nop             ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed
    ;++++++++++++++++++++++++++
    ;+ Bloco Y2 so com coef. DC +
    ;++++++++++++++++++++++++++
        call vlc_DC     ; Funcao VLC "fast"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant "fast"
        lacc *,16,ar1
        bcndd Y2_DCP,GT ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                       -> Branch Delayed
        sub pqndc       ;                       -> Branch Delayed

Y2_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y2fIDCT,NEQ
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        lacl #128
        bd Y2fIDCT
        sacl *,3,ar1    ; 128*3=1024            -> Branch Delayed
        lacc *,13       ;                       -> Branch Delayed

Y2_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y2DCneg,LT
Y2DCpos sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
```

```
        sacl *,ar1
        bd Y2fIDCT
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y2DCneg ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y2fIDCT adrk #64        ; limita-se a espalhar o coeficiente DC
        mar *,ar6       ; pelo resto do bloco
        rpt #63
        sach *+
        bd BY3
        mar *,ar4
        lacc *,ar5
    ;++++++++++++++++++++++++++++++++
    ;+ Bloco Y2 com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++
Y2cCoef call vlc        ; Funcao VLC "slow"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16,ar1
        bcndd Y2coefP,GT
        lar ar0,#63     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

Y2coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y2next,NEQ
        lar ar0,#62     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y2coefP ;lar ar0,#63
Y2next  bcnd Y2Nzero,NEQ
        mar *+,ar0
        banzd Y2next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y2sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y2Nzero bcndd Y2coefn,LT
Y2coefp sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y2next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y2sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y2coefn ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y2next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

*       b Y2sIDCT
        sbrk #64

Y2sIDCT call idctslw    ; Funcao iDCT "slow"
```

```
        mar *,ar4
        lacc *,ar5

**************
** Bloco Y3 **
**************
BY3:    and *+,ar1
        bcndd Y3nNULL,NEQ
        and #0ffh       ;           -> Branch Delayed   2 ciclos!!!!!!!
    ;+++++++++++++++++
    ;+ Bloco Y3 nulo +
    ;+++++++++++++++++
Y3NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar6
        adrk #64        ; soma 64 a ar6
        bd BY4                              ; Funcao invQuant nula
        mar *,ar4                           ; Funcao iDCT nula
        lacc *,ar5
    ;+++++++++++++++++++++
    ;+ Bloco Y3 nao nulo +
    ;+++++++++++++++++++++
Y3nNULL bcndd Y3cCoef,NEQ
        nop             ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed
    ;++++++++++++++++++++++++++++
    ;+ Bloco Y3 so com coef. DC +
    ;++++++++++++++++++++++++++++
        call vlc_DC     ; Funcao VLC "fast"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant "fast"
        lacc *,16,ar1
        bcndd Y3_DCP,GT ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                       -> Branch Delayed
        sub pqndc       ;                       -> Branch Delayed

Y3_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y3fIDCT,NEQ
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        lacl #128
        bd Y3fIDCT
        sacl *,3,ar1    ; 128*3=1024            -> Branch Delayed
        lacc *,13       ;                       -> Branch Delayed

Y3_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y3DCneg,LT
Y3DCpos sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd Y3fIDCT
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y3DCneg ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y3fIDCT adrk #64        ; limita-se a espalhar o coeficiente DC
        mar *,ar6       ; pelo resto do bloco
        rpt #63
        sach *+
        bd BY4
        mar *,ar4
        lacc *,ar5
    ;+++++++++++++++++++++++++++++++++
    ;+ Bloco Y3 com 64 coef. nao nulos +
    ;+++++++++++++++++++++++++++++++++
Y3cCoef call vlc        ; Funcao VLC "slow"  - retorna com arp=ar4
```

```
        ldp #pqndc      ; Funcao invQuant nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16,ar1
        bcndd Y3coefP,GT
        lar ar0,#63     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

Y3coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y3next,NEQ
        lar ar0,#62     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y3coefP ;lar ar0,#63
Y3next  bcnd Y3Nzero,NEQ
        mar *+,ar0
        banzd Y3next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y3sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y3Nzero bcndd Y3coefn,LT
Y3coefp sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y3next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y3sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y3coefn ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y3next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

*       b Y3sIDCT
        sbrk #64

Y3sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar4
        lacc *,ar5

**************
** Bloco Y4 **
**************
BY4:    and *+,ar1
        bcndd Y4nNULL,NEQ
        and #0ffh       ;           -> Branch Delayed   2 ciclos!!!!!!!
    ;+++++++++++++++++
    ;+ Bloco Y4 nulo +
    ;+++++++++++++++++
Y4NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar6
        adrk #64        ; soma 64 a ar6
        bd BCb                              ; Funcao invQuant nula
        mar *,ar4                           ; Funcao iDCT nula
        lacc *,ar5
    ;+++++++++++++++++++++
    ;+ Bloco Y4 nao nulo +
    ;+++++++++++++++++++++
Y4nNULL bcndd Y4cCoef,NEQ
```

```
        nop             ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed
    ;+++++++++++++++++++++++++++
    ;+ Bloco Y4 so com coef. DC +
    ;+++++++++++++++++++++++++++
        call vlc_DC     ; Funcao VLC "fast"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant "fast"
        lacc *,16,ar1
        bcndd Y4_DCP,GT ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                       -> Branch Delayed
        sub pqndc       ;                       -> Branch Delayed

Y4_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y4fIDCT,NEQ
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        lacl #128
        bd Y4fIDCT
        sacl *,3,ar1    ; 128*3=1024            -> Branch Delayed
        lacc *,13       ;                       -> Branch Delayed

Y4_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y4DCneg,LT
Y4DCpos sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd Y4fIDCT
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y4DCneg ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y4fIDCT adrk #64        ; limita-se a espalhar o coeficiente DC
        mar *,ar6       ; pelo resto do bloco
        rpt #63
        sach *+
        bd BCb
        mar *,ar4
        lacc *,ar5
    ;+++++++++++++++++++++++++++++++++
    ;+ Bloco Y4 com 64 coef. nao nulos +
    ;+++++++++++++++++++++++++++++++++
Y4cCoef call vlc        ; Funcao VLC "slow"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16,ar1
        bcndd Y4coefP,GT
        lar ar0,#63     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

Y4coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y4next,NEQ
        lar ar0,#62     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y4coefP ;lar ar0,#63
Y4next  bcnd Y4Nzero,NEQ
        mar *+,ar0
```

```
        banzd Y4next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y4sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y4Nzero bcndd Y4coefn,LT
Y4coefp sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y4next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd Y4sIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Y4coefn ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y4next,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

*       b Y4sIDCT
        sbrk #64

Y4sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar4
        lacc *,ar5

**************
** Bloco Cb **
**************
BCb:    and *+,ar1
        bcndd CbnNULL,NEQ
        and #0ffh       ;           -> Branch Delayed   2 ciclos!!!!!!!
    ;+++++++++++++++++
    ;+ Bloco Cb nulo +
    ;+++++++++++++++++
CbNULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar6
        adrk #64        ; soma 64 a ar6
        bd BCr                              ; Funcao invQuant nula
        mar *,ar4                           ; Funcao iDCT nula
        lacc *,ar5
    ;++++++++++++++++++++
    ;+ Bloco Cb nao nulo +
    ;++++++++++++++++++++
CbnNULL bcndd CbcCoef,NEQ
        nop             ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed
    ;++++++++++++++++++++++++++
    ;+ Bloco Cb so com coef. DC +
    ;++++++++++++++++++++++++++
        call vlc_DC     ; Funcao VLC "fast"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant "fast"
        lacc *,16,ar1
        bcndd Cb_DCP,GT ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                       -> Branch Delayed
        sub pqndc       ;                       -> Branch Delayed

Cb_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd CbfIDCT,NEQ
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        lacl #128
        bd CbfIDCT
        sacl *,3,ar1    ; 128*3=1024            -> Branch Delayed
```

```
        lacc *,13       ;                       -> Branch Delayed

Cb_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd CbDCneg,LT
CbDCpos sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd CbfIDCT
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

CbDCneg ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
CbfIDCT adrk #64        ; limita-se a espalhar o coeficiente DC
        mar *,ar6       ; pelo resto do bloco
        rpt #63
        sach *+
        bd BCr
        mar *,ar4
        lacc *,ar5
    ;++++++++++++++++++++++++++++++++++
    ;+ Bloco Cb com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++++
CbcCoef call vlc        ; Funcao VLC "slow"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16,ar1
        bcndd CbcoefP,GT
        lar ar0,#63     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

CbcoefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Cbnext,NEQ
        lar ar0,#62     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

CbcoefP ;lar ar0,#63
Cbnext  bcnd CbNzero,NEQ
        mar *+,ar0
        banzd Cbnext,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd CbsIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

CbNzero bcndd Cbcoefn,LT
Cbcoefp sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Cbnext,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd CbsIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Cbcoefn ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
```

```
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Cbnext,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

*       b CbsIDCT
        sbrk #64

CbsIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar4
        lacc *,ar5

**************
** Bloco Cr **
**************
BCr:    and *,ar1
        bcndd CrnNULL,NEQ
        and #0ffh       ;           -> Branch Delayed   2 ciclos!!!!!!!
    ;+++++++++++++++++
    ;+ Bloco Cr nulo +
    ;+++++++++++++++++
CrNULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar6
        adrk #64        ; soma 64 a ar6     ; Funcao invQuant nula
        retd                                ; iDCT nula
        mar *,ar4       ; soma 1 a ar4          -> Branch Delayed
        adrk #1         ;                       -> Branch Delayed

    ;++++++++++++++++++++
    ;+ Bloco Cr nao nulo +
    ;++++++++++++++++++++
CrnNULL bcndd CrcCoef,NEQ
        nop             ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed
    ;++++++++++++++++++++++++++
    ;+ Bloco Cr so com coef. DC +
    ;++++++++++++++++++++++++++
        call vlc_DC     ; Funcao VLC "fast"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant "fast"
        lacc *,16,ar1
        bcndd Cr_DCP,GT ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                       -> Branch Delayed
        sub pqndc       ;                       -> Branch Delayed

Cr_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd CrfIDCT,NEQ
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        lacl #128
        bd CrfIDCT
        sacl *,3,ar1    ; 128*3=1024            -> Branch Delayed
        lacc *,13       ;                       -> Branch Delayed

Cr_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd CrDCneg,LT
CrDCpos sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd CrfIDCT
        lacc *,13       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

CrDCneg ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
CrfIDCT adrk #64        ; limita-se a espalhar o coeficiente DC
```

```
        mar *,ar6       ; pelo resto do bloco
        rpt #63
        sach *+
        retd
        mar *,ar4       ; soma 1 a ar4          -> Branch Delayed
        adrk #1         ;                       -> Branch Delayed
    ;++++++++++++++++++++++++++++++++++
    ;+ Bloco Cr com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++++
CrcCoef call vlc        ; Funcao VLC "slow"  - retorna com arp=ar4

        ldp #pqndc      ; Funcao invQuant nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16,ar1
        bcndd CrcoefP,GT
        lar ar0,#63     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

CrcoefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Crnext,NEQ
        lar ar0,#62     ;                       -> Branch Delayed
        lacc *,16       ;                       -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

CrcoefP ;lar ar0,#63
Crnext  bcnd CrNzero,NEQ
        mar *+,ar0
        banzd Crnext,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd CrsIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

CrNzero bcndd Crcoefn,LT
Crcoefp sfl             ; Acc=2*Level           -> Branch delayed
        add one,16      ; Acc=2*Level+1         -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Crnext,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

        bd CrsIDCT
        sbrk #64        ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

Crcoefn ;sfl            ; Acc=2*Level           -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Crnext,*-,ar1
        lacc *,16       ;                       -> Branch Delayed
        nop             ;                       -> Branch Delayed

*       b CrsIDCT
        sbrk #64

CrsIDCT call idctslw    ; Funcao iDCT "slow"

MB_end  retd
        mar *,ar4       ; soma 1 a ar4          -> Branch Delayed
        adrk #1         ;                       -> Branch Delayed
```

C.6 FICHEIRO "transf.asm"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "transf.asm"                                                       *
* DESCRIÇÃO: Este ficheiro contem a chamada 'as varias rotinas que efectuam a  *
*            inicializacao das funcoes que efectuam as duas transformadas uti- *
*            lizadas na codificacao: DCT e IDCT.                               *
*            Efectuam tambem a definicao de algumas variaveis necessarias ao   *
*            processamento.                                                    *
*                                                                              *
*******************************************************************************
*
        .include "dct.asm"
        .include "idct.asm"
*
        .text
*
*******************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
*******************************************************************************
traini: ldp #rndoff
        lacc #8000h
        sacl rndoff
        lacl #1
        sacl one
        sacl two,1
*
        call dctini
        call idctini
*
* inicializacao da tabela CBPC. Chamada para cada picture.
*
        mar *,ar4
        lar ar4,#CBPC
        rptz #54
        sacl *+
        mar *,ar1
        retd
        lar ar4,#CBPC

*******************************************************************************
* Definicao de variaveis
*******************************************************************************
temp    .usect  "B1",64     ; resultados temporarios das transformadas
rndoff  .usect  "B1",1      ; roundoff factor
one     .usect  "B1",1      ; posicao de memoria inicializada com '1'
two     .usect  "B1",1      ; posicao de memoria inicializada com '2'
src     .usect  "B1",1      ; ponteiro para o inicio do bloco a transformar
dst     .usect  "B1",1      ; endereco do destino da operacao corrente
nextsrc .usect  "B1",1      ; endereco do destino final da idct
pqdc    .usect  "B1",1      ; passo de quantificacao do valor DC
pqndc   .usect  "B1",1      ; passo de quantificacao dos restantes coef.
TregBuf .usect  "B1",1      ; Buffer do TREG1
AccBuf  .usect  "B1",1      ; Buffer do Acc
*

* NOTA: Se a variavel 'pqdc' tiver o valor nulo, isso significa que se
*       trata de uma picture INTER, visto que nesse caso todo o bloco sera'
*       quantizado om o passo de quantizacao dado por 'pqndc'
```

C.7 FICHEIRO "dct.asm"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "dct.asm"                                                          *
```

```
* DESCRIÇÃO: Este ficheiro contem a rotina que executa o calculo da transfor-  *
*            mada de coseno discreta - DCT. O algoritmo utilizado foi o baseado*
*            na multiplicacao de matrizes da forma:                            *
*               [F] = [C].[A].[C]T                                             *
*            O calculo deste produto é realizado por aplicacao consecutiva de  *
*            duas operacoes da forma:                                          *
*               [Y] = ( [Md].[Mp]T )T                                          *
*            em que [Md] e' a matriz dos pixels armazenada em memoria de dados *
*            e Mp e' a matriz dos coeficientes armazenada em memoria de progra-*
*            ma.                                                               *
*            O calcula do DCT e' obtido sa seguinte forma:                     *
*               [F] = {( [A].[C]T )T .[C]T}T                                   *
*                   =  [C].[A].[C]T                                            *
*                                                                              *
********************************************************************************

        .text
********************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
********************************************************************************
dctini: spm 1                   ; Programacao do registo PM para 1 "shift"
        clrc CNF
        lar ar1,#coeff          ; coeficientes da DCT
        rpt #(edata-idata-1)
        bldd #idata,*+

        retd
        setc CNF                ; a instrucao "mac" tem um dos operandos arma-
                                ; zenado em memoria de programa
        nop                     ;                              <- Branch delayed

********************************************************************************
* Funcao que executa o calculo da DCT
********************************************************************************
dct:    setc sxm
        ldp #src
        sar ar1,src
        lar ar6,#temp
        sar ar6,dst             ; endereco de destino (primeira coluna)
*
        mar *,ar1
        lar ar7,#5
*
        lamm TREG1
        sacl TregBuf
        lacb
        sacl AccBuf
*
        lar ar4,#CBPC
        lacc pqdc
        bcndd nextdct,EQ        ; so' executa um destes dois lar's
        lar ar5,#CBPCMskP       ;                              <- Branch delayed
        lar ar5,#CBPCMskI       ;                              <- Branch delayed
*
*  primeiro conjunto de coeficientes
*
nextdct:
*       setc XF
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
```

```
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
*
*  segundo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  terceiro conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
```

```
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  quarto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  quinto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c40,*+
        lta *,ar6
```

```
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  sexto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c50,*+
        lta *,ar6
```

```
        adds rndoff
        sach *+,ar1
*
*  setimo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  oitavo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
```

```
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
* prepara o calculo da proxima dimensao
*
rnd2    lacc dst
        dmov src
        sacl src
*       clrc XF
        lacc pqdc
        samm TREG1
        bcndd round2P,EQ        ; ainda executa estes dois lar's
        lar ar1,src
        lar ar6,dst             ; endereco de destino (primeira coluna)

*
*  primeiro conjunto de coeficientes
*
round2I rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        nop
        xc 1,NEQ
        adds rndoff
        satl
        xc 1,GT
        add one,16              ; soma 1 a Acc_High
*
        sach *+,ar1             ; guarda resultado final
        and #0ffffh,16
        nop
        xc 1,NEQ
        lacl #0ffh
        sacb
        bd round2x              ; ainda executa 'lacc' e 'samm'
round2P lacc pqndc              ;                        <-- Branch Delay
        samm TREG1              ;                        <-- Branch Delay
*
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        and #0ffffh,16
        nop
        xc 1,NEQ
        lacl #0ffh
        sacb
*
round2x rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High (2 words)
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
```

```
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz    7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c00,*+              ; acc = 0 ,preg= x0 * c00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  segundo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
```

```
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c10,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  terceiro conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
```

```
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
        sach *+,ar1              ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c20,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16               ; soma 1 a Acc_High
```

```
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  quarto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
```

```
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c30,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  quinto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
```

```
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c40,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  sexto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
```

```
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c50,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  setimo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
```

```
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c60,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
*  oitavo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
```

```
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar1             ; guarda resultado final
        orb
        sacb
*
        rptz 7
        mac c70,*+
        lta *,ar6
        adds rndoff
        satl
        xc 1,LT
        add one,16              ; soma 1 a Acc_High
        sach *+,ar5             ; guarda resultado final
        orb
        sacb
*
putmask:and #0ffffh,16
        bcnd semcoef,EQ
        lacc *+,ar4             ; ar5 aponta para as mascaras
        or *
        sacl *,ar7
        banz predct,*-,ar6
*
        mar *,ar4
        mar *+,ar5              ; incrementa para a posicao do proximo MB
        sbrk #6
        bd predct
        mar *,ar6
        lar ar7,#5
*
semcoef lacb
```

```
        and #0ffh
        bcndd NextBl,EQ
        mar *,ar5
        lacc *+,ar4             ; ainda e' executada se saltar, para incrementar ar5
        and #0ff00h
        or *
        sacl *
*
NextBl  mar *,ar7
        banz predct,*-,ar6
NextMB  mar *,ar4
        mar *+,ar5
        sbrk #6
        mar *,ar6
        lar ar7,#5
*
* loop for next dimension
*
predct  cmpr 0                  ; testa se ar6 > ARCR
        bcndd enddct,TC
        dmov src
        sar ar6,src
*
        bd nextdct,*,ar1
        lar ar6,dst             ; endereco de destino (primeira coluna)
        lar ar1,src
*
enddct: lacc TregBuf
        samm TREG1
        retd
        lacc AccBuf
        sacb
*******************************************************************************
*
*  DEFINICAO E DECLARACAO DA TABELA DE COEFICIENTES
*
*******************************************************************************
        .asect "DCT_COEF",0fe00h
                                ; esta tabela deve ser armazenada em  memoria
                                ; de programa atraves de um comando 'CNFP'
*
        .label idata            ; coeficientes DCT
c00     .word  11585            ; primeira linha
c01     .word  11585            ; 11585 = (1/2) * 2^(-1/2) no formato Q15
c02     .word  11585
c03     .word  11585
c04     .word  11585
c05     .word  11585
c06     .word  11585
c07     .word  11585
c10     .word  16069            ; segunda linha
c11     .word  13623
c12     .word   9102
c13     .word   3196
c14     .word  -3196            ; 3196 = (1/2) * sin(Pi/16) no formato Q15
c15     .word  -9102            ; 9102 = (1/2) * sin(3Pi/16) no formato Q15
c16     .word -13623            ; 13622 = (1/2) * cos(3Pi/16) no formato Q15
c17     .word -16069            ; 16069 = (1/2) * cos(Pi/16) no formato Q15
c20     .word  15137            ; terceira linha
c21     .word   6270            ; 6269 = (1/2) * sin(Pi/8) no formato Q15
c22     .word  -6270            ; 15136 = (1/2) * cos(Pi/8) no formato Q15
c23     .word -15137
c24     .word -15137
c25     .word  -6270
c26     .word   6270
c27     .word  15137
c30     .word  13623            ; quarta linha
c31     .word  -3196
c32     .word -16069
c33     .word  -9102
c34     .word   9102
c35     .word  16069
c36     .word   3196
c37     .word -13623
c40     .word  11585            ; quinta linha
c41     .word -11585
c42     .word -11585
c43     .word  11585
c44     .word  11585
```

```
c45     .word -11585
c46     .word -11585
c47     .word  11585
c50     .word   9102            ; sexta linha
c51     .word -16069
c52     .word   3196
c53     .word  13623
c54     .word -13623
c55     .word  -3196
c56     .word  16069
c57     .word  -9102
c60     .word   6270            ; setima linha
c61     .word -15137
c62     .word  15137
c63     .word  -6270
c64     .word  -6270
c65     .word  15137
c66     .word -15137
c67     .word   6270
c70     .word   3196            ; oitava linha
c71     .word  -9102
c72     .word  13623
c73     .word -16069
c74     .word  16069
c75     .word -13623
c76     .word   9102
c77     .word  -3196
        .label edata            ; fim da tabela de coeficientes

*******************************************************************************
*
*  Definicao do espaco para armazenamento desta tabela
*
*******************************************************************************
coeff   .usect "B0",64          ; coeficientes DCT (Bloco de memoria 'B0')
*
```

C.8 FICHEIRO "idct.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "idct.asm"                                                        *
* DESCRIÇÃO: Este ficheiro contem a rotina que executa o calculo da transfor-  *
*            mada inversa de coseno discreta - IDCT. O algoritmo utilizado foi *
*            o baseado na multiplicacao de matrizes da forma:                 *
*               [f] = [C].[F].[C]T                                            *
*            O calculo deste produto é realizado por aplicacao consecutiva de  *
*            duas operacoes da forma:                                         *
*               [Y] = ( [Md].[Mp]T )T                                         *
*            em que [Md] e' a matriz dos pixels armazenada em memoria de dados *
*            e Mp e' a matriz dos coeficientes armazenada em memoria de progra-*
*            ma.                                                              *
*            O calcula do IDCT e' obtido sa seguinte forma:                   *
*               [f] = {( [F].[C]T )T .[C]T}T                                  *
*                   =  [C].[F].[C]T                                           *
*                                                                             *
*******************************************************************************

        .text
*******************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
*******************************************************************************
*
idctini:
        spm 1                   ; Programacao do registo PM para 1 "shift"
        clrc CNF
        lar ar1,#icoeff         ; coeficientes da IDCT
        rpt #(iedata-iidata-1)
        bldd #iidata,*+
        retd
        setc CNF                ; a instrucao "mac" tem um dos operandos arma-
                                ; zenado em memoria de programa
```

```
        nop                     ;                               <- Branch delayed

********************************************************************************
* Funcao que executa o calculo da IDCT (versao mais lenta)
********************************************************************************
idctslw:setc sxm
        ldp #src
        sar ar1,src
        adrk #64
        sar ar1,nextsrc         ; ponteiro para o proximo bloco
        sar ar6,dst
        lar ar7,#1              ; ar7 := dimensao-1 = 2-1 = 1
*
*       inicio do ciclo (executado 2 vezes)
*
        lar ar6,#temp           ; endereco de destino (primeira coluna)
*
*  primeiro conjunto de coeficientes
*
imult:  lar ar1,src
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
*
*  segundo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
```

```
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  terceiro conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  quarto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
```

```
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  quinto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
```

```
*   sexto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*   setimo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
```

```
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
*  oitavo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar7             ; ar7 := dimensao
*
*       reinicio do ciclo
*
        lar ar1,#temp
        banzd imult,*-,ar1
        sar ar1,src             ;                       <-- Branch Delayed
        lar ar6,dst             ;                       <-- Branch Delayed
*
        lar ar1,nextsrc         ; por forma a que na proxima chamada a esta
                                ; funcao, ar1 aponte para o proximo bloco
        mar *,ar6
        retd
        adrk #64                ;                       <-- Return Delayed
        mar *,ar1               ;                       <-- Return Delayed
*******************************************************************************
*
*  DEFINICAO E DECLARACAO DA TABELA DE COEFICIENTES
*
*******************************************************************************
        .asect "DCT_COEF"
                                ; esta tabela deve ser armazenada em  memoria
                                ; de programa atraves de um comando 'CNFP'
*
        .label iidata           ; coeficientes IDCT
ct00    .word  11585
ct01    .word  16069
ct02    .word  15137
```

```
ct03    .word  13623
ct04    .word  11585
ct05    .word   9102
ct06    .word   6270
ct07    .word   3196
ct10    .word  11585            ; 11585 = (1/2) * 2^(-1/2) no formato Q15
ct11    .word  13623
ct12    .word   6270            ; 6269 = (1/2) * sin(Pi/8) no formato Q15
ct13    .word  -3196
ct14    .word -11585
ct15    .word -16069
ct16    .word -15137
ct17    .word  -9102
ct20    .word  11585
ct21    .word   9102
ct22    .word  -6270            ; 15136 = (1/2) * cos(Pi/8) no formato Q15
ct23    .word -16069
ct24    .word -11585
ct25    .word   3196
ct26    .word  15137
ct27    .word  13623
ct30    .word  11585
ct31    .word   3196
ct32    .word -15137
ct33    .word  -9102
ct34    .word  11585
ct35    .word  13623
ct36    .word  -6270
ct37    .word -16069
ct40    .word  11585
ct41    .word  -3196            ; 3196 = (1/2) * sin(Pi/16) no formato Q15
ct42    .word -15137
ct43    .word   9102
ct44    .word  11585
ct45    .word -13623
ct46    .word  -6270
ct47    .word  16069
ct50    .word  11585
ct51    .word  -9102            ; 9102 = (1/2) * sin(3Pi/16) no formato Q15
ct52    .word  -6270
ct53    .word  16069
ct54    .word -11585
ct55    .word  -3196
ct56    .word  15137
ct57    .word -13623
ct60    .word  11585
ct61    .word -13623            ; 13622 = (1/2) * cos(3Pi/16) no formato Q15
ct62    .word   6270
ct63    .word   3196
ct64    .word -11585
ct65    .word  16069
ct66    .word -15137
ct67    .word   9102
ct70    .word  11585
ct71    .word -16069            ; 16069 = (1/2) * cos(Pi/16) no formato Q15
ct72    .word  15137
ct73    .word -13623
ct74    .word  11585
ct75    .word  -9102
ct76    .word   6270
ct77    .word  -3196
        .label iedata           ; fim da tabela de coeficientes

*******************************************************************************
*
*  Definicao do espaco para armazenamento desta tabela
*
*******************************************************************************
icoeff  .usect "B0",64          ; coeficientes iDCT (Bloco de memoria 'B0')
*
```

C.9 FICHEIRO "vlc.asm"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "vlc.asm"                                                         *
* DESCRIÇÃO: Este ficheiro contem a rotina que executa a codificacao dos coefi-*
*            cientes de acordo com um codigo de comprimento variavel, descrito *
*            na norma H.263.                                                  *
*            A determinacao dos codigos e' feita efectuando a leitura dos coe- *
*            ficientes de cada bloco segundo um padrao em zig-zag e baseia-se  *
*            nos valores das variaveis RUN (numero de coeficientes nulos ao    *
*            longo da diagonal) e LAST (indica se o coeficiente lido corres-   *
*            ponde ao ultimo coeficiente do bloco).                           *
*            No caso de nao se encontrar definido qualquer codigo variavel para*
*            o valor do coeficiente em processamento, o codigo a enviar e' de- *
*            terminado atraves de uma sequencia de "escape", tambem prevista   *
*            na recomendacao H.263                                            *
*                                                                             *
*******************************************************************************

        .text
*
*******************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
*******************************************************************************
vlc_ini:
        lar ar2,#buf_vlc    ; ar2 -> Buffer VLC
        ldp #TCOEF_0
        lacc #(TCOEF_0-2)
        ldp #CoefPnL
        sacl CoefPnL
        add #1
        sacl CoefNnL
        ldp #TCOEF_0
        lacc #(TCOEF_1-2)
        ldp #CoefPnL
        sacl CoefPL
        adds onevlc
        sacl CoefNL
        lacl #7
        samm INDX
        zac
        sacb
        retd
        sacl AccbHbuf   ;                               -> Return Delayed
        sacl TREG1buf   ;                               -> Return Delayed

*******************************************************************************
* Funcao VLC
*******************************************************************************
vlc:    ldp #Pic_PnI
        lacl Pic_PnI
        bcndd vlc_P,NEQ
        lacc *          ;                               -> Branch Delayed
        setc sxm        ;                               -> Branch Delayed
vlc_I:
        ldp #ffffvlc
        lacc ffffvlc    ;
        samm ar0        ; run=-1

        lacc *+,8,ar2   ; aponto P2
        sacl *+         ; escreve codigo vlc
        sub #128
        bcndd vlc_y,NEQ
        lacl #8         ;                               -> Brunch Delayed
        sacl *+,ar1     ; escreve No. de bits do codigo VLC (8)
                        ;                               -> Brunch Delayed
        mar *,ar2
        sbrk #2
        lacl #255
        sacl *
        adrk #2
        mar *,ar1
```

```
        bd vlc_y
        sbrk #1         ;                               -> Brunch Delayed
        sacl *+         ;                               -> Brunch Delayed

vlc_P:  ;lacc *         ;                               -> Branch Delayed
        ;setc sxm       ;                               -> Branch Delayed

        bit *,0         ; regista em TC o bit de sinal
        abs             ; verifica se existe a necessidade de efectuar "clipping"
        sub #127
        bcndd vlc_x,LT
        lacl #127       ;                               -> Branch Delayed
        ldp #ffffvlc    ;                               -> Branch Delayed

        xc 1,TC
        neg

        sacl *

        ;setc sxm
vlc_x:  ;ldp #ffffvlc   ;                               -> Branch Delayed
        lacc ffffvlc    ;
        samm ar0        ; run=-1

        lacc *+,ar0     ;   carrego P1 ; aponto P2
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

vlc_y:  lacc *0+,ar0    ;   carrego P2 ; aponto P3
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *,ar0      ;   carrego P3 ; aponto P3
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        adrk #8         ;   aponto P4
        lacc *0-,ar0    ;   carrego P4 ; aponto P5
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P5 ; aponto P6
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P6 ; aponto P7
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P7 ; aponto P8
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P8 ; aponto P9
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P9 ; aponto P10
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *,ar0      ;   carrego P10; aponto P10
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        adrk #8         ;   aponto P11
        lacc *0-,ar0    ;   carrego P11; aponto P12
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
```

```
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P12; aponto P13
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P13; aponto P14
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P14; aponto P15
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P15; aponto P16
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P16; aponto P17
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P17; aponto P18
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P18; aponto P19
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P19; aponto P20
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P20; aponto P21
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *,ar0      ;   carrego P21; aponto P21
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        adrk #8         ;   aponto P22
        lacc *0-,ar0    ;   carrego P22; aponto P23
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P23; aponto P24
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P24; aponto P25
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P25; aponto P26
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P26; aponto P27
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P27; aponto P28
```

```
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P28; aponto P29
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P29; aponto P30
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P30; aponto P31
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P31; aponto P32
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P32; aponto P33
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P33; aponto P34
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P34; aponto P35
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P35; aponto P36
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P36; aponto P37
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P37; aponto P38
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P38; aponto P39
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P39; aponto P40
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P40; aponto P41
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P41; aponto P42
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P42; aponto P43
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed
```

```
        lacc *,ar0      ;   carrego P43; aponto P43
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        adrk #8         ;   aponto P44
        lacc *0+,ar0    ;   carrego P44; aponto P45
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P45; aponto P46
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P46; aponto P47
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P47; aponto P48
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P48; aponto P49
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P49; aponto P50
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P50; aponto P51
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P51; aponto P52
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P52; aponto P53
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P53; aponto P54
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *,ar0      ;   carrego P54; aponto P54
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        adrk #8         ;   aponto P55
        lacc *0+,ar0    ;   carrego P55; aponto P56
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P56; aponto P57
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0+,ar0    ;   carrego P57; aponto P58
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P58; aponto P59
        ccd VLCcoef,NEQ
```

```
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P59; aponto P60
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *0-,ar0    ;   carrego P60; aponto P61
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *,ar0      ;   carrego P61; aponto P61
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        adrk #8             ;   aponto P62
        lacc *0+,ar0    ;   carrego P62; aponto P63
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

        lacc *+,ar0     ;   carrego P63; aponto P64
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed
        lacc *+,ar0     ;   carrego P64; aponto P1
        ccd VLCcoef,NEQ
        mar *+,ar1      ;   incrementa contador de RUNs (=ar0)
        nop             ;                               -> Call Delayed

*       call VLCLast
*
*****************************************************************************
* Funcao VLC (caso do ultimo coeficiente - LAST)
*****************************************************************************
VLCLast:
        lacb            ; AccB tem o Level anterior
        bcndd LNegLevel,LT
        sbrk #64        ; Ar1 aponta para o inicio deste bloco -> Branch Delayed
        lar ar2,vlc_ptr ; valor de ar2 antes da ultima escrita -> Branch Delayed

LPosLevel:
        sub #3
        bcndd LEscape_P,GT
        lamm TREG0      ; carrega numero de RUNs anterior -> Branch Delayed
        sub #40         ;                                   -> Branch Delayed

        bcndd LEscape_P,GT
        mar *,ar3       ;                               -> Branch Delayed
        mpy #1          ; multiplica RUNs por 1 (1*2(devido a PM))
                        ;                               -> Branch Delayed
        lamm TREG0      ; soma TREG0 => Acc=2*TREG0+TREG0=3*TREG0
        apac
        addb
        sfl
        adds CoefPL     ; Acc tem o endereco da tabela de COEF - 2
        samm ar3        ; ar3 aponta a tabela (codigo VLC corespondente)
        nop             ; -> Pipeline STALL devido a 'samm'
        nop             ; -> Pipeline STALL devido a 'samm'
        lacc *+,ar2     ; carrega codigo VLC ; ar2 aponta para o buffer VLC
        bcndd LEscape_P,EQ
        nop             ;                               ->Branch Delayed
        nop             ;                               ->Branch Delayed

        sacl *+,ar3     ; escreve codigo VLC
        retd
        lacc *,ar2      ; le No. de bits do codigo VLC
        sacl *+,ar4     ; escreve No. de bits do codigo VLC

LEscape_P:              ;   '0000011'+last(1)+run(6)+level(8)
        mar *,ar2
        lamm TREG0      ; carrega numero de RUNs

        or Lesc_cod     ; Lesc_cod='0000 0001 1100 0000'
        sacl *+,2       ; escreve codigo VLC - [esc_cod+last+run]
        lacc #14
```

```
        sacl *+         ; escreve No. de bits do codigo VLC (7+1+6=14)

        lacb
        sfl
        sacl *+,7       ; escreve codigo VLC - [level(8)]
        retd
        lacl #8         ;
        sacl *+,ar4     ; escreve No. de bits do codigo VLC (8)

LNegLevel:
        add #3          ; |Level|-3>0 <=> -Level-3>0 <=> Level+3<0
        bcndd LEscape_N,LT
        lamm TREG0      ; carrega numero de RUNs        -> Branch Delayed
        sub #40         ;                               -> Branch Delayed

        bcndd LEscape_N,GT
        mar *,ar3       ;                               -> Branch Delayed
        mpy #1          ; multiplica RUNs por 1 (1*2(devido a PM))
                        ;                               -> Branch Delayed
        lamm TREG0      ; soma TREG0 => Acc=2*TREG0+TREG0=3*TREG0
        apac
        sbb
        sfl
        adds CoefNL     ; Acc tem o endereco da tabela de COEF - 1
        samm ar3        ; ar3 aponta a tabela (no. de bits do cod. VLC)
        lacc #16
        sub *-
        samm TREG1
        lacc *+,ar2     ; carrega codigo VLC ; ar2 aponta para o buffer VLC
        bcndd LEscape_N,EQ
        addt onevlc     ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        sacl *+,ar3     ; escreve codigo VLC
        retd
        lacc *,ar2      ; le No. de bits do codigo VLC
        sacl *+,ar4     ; escreve No. de bits do codigo VLC

LEscape_N:              ;   '0000011'+last(1)+run(6)+level(8)
        mar *,ar2
        lamm TREG0      ; carrega numero de RUNs

        or Lesc_cod     ; Lesc_cod='0000 0001 1100 0000'
        sacl *+,2       ; escreve codigo VLC - [esc_cod+last+run]
        lacc #14
        sacl *+         ; escreve No. de bits do codigo VLC (7+1+6=14)

        lacb
        sfl
        sacl *+,7       ; escreve codigo VLC - [level(8)]
        retd
        lacl #8         ;
        sacl *+,ar4     ; escreve No. de bits do codigo VLC (8)

*******************************************************************************
* Funcao VLC (coeficiente intermedio)
*******************************************************************************
VLCcoef:
        bcndd NegLevel,LT
        sacb            ;   AccB fica com Level         -> Branch Delayed
        sar ar2,vlc_ptr ;                               -> Branch Delayed

PosLevel:
        sub #12
        bcndd Escape_P,GT
        lamm ar0        ; carrega numero de RUNs        -> Branch Delayed
        samm TREG0      ;                               -> Branch Delayed

        sub #26
        bcndd Escape_P,GT
        mar *,ar3       ;                               -> Branch Delayed
        mpy #6          ; multiplica RUNs por 12 (6*2(devido a PM))
                        ;                               -> Branch Delayed
        lacb
        apac
        sfl
        adds CoefPnL    ; Acc tem o endereco da tabela de COEF - 2
        samm ar3        ; ar3 aponta a tabela (codigo VLC corespondente)
        nop             ; -> Pipeline STALL devido a 'samm'
```

```
        nop             ; -> Pipeline STALL devido a 'samm'
        lacc *+,ar2     ; carrega codigo VLC ; ar2 aponta para o buffer VLC
        bcndd Escape_P,EQ
        nop             ;                               ->Branch Delayed
        nop             ;                               ->Branch Delayed

        sacl *+,ar3     ; escreve codigo VLC
        lacc *,ar2      ; le No. de bits do codigo VLC
        sacl *+,ar1     ; escreve No. de bits do codigo VLC
        retd
        lacc ffffvlc    ;                               ->Branch Delayed
        samm ar0        ; run=-1                        ->Branch Delayed

Escape_P:               ;   '0000011'+last(1)+run(6)+level(8)
        mar *,ar2
        lamm TREG0      ; carrega numero de RUNs

        or esc_cod      ; esc_cod='0000 0001 1000 0000'
        sacl *+,2       ; escreve codigo VLC - [esc_cod+last+run]
        lacc #14
        sacl *+         ; escreve No. de bits do codigo VLC (7+1+6=14)

        lacb            ; carrega Level
        sfl
        sacl *+,7       ; escreve codigo VLC - [level(8)]
        lacl #8         ;
        sacl *+,ar1     ; escreve No. de bits do codigo VLC (8)
        retd
        lacc ffffvlc    ;                               ->Branch Delayed
        samm ar0        ; run=-1                        ->Branch Delayed

NegLevel:
        add #12         ; |Level|-12>0 <=> -Level-12>0 <=> Level+12<0
        bcndd Escape_N,LT
        lamm ar0        ; carrega numero de RUNs        -> Branch Delayed
        samm TREG0      ;                               -> Branch Delayed

        sub #26
        bcndd Escape_N,GT
        mar *,ar3       ;                               -> Branch Delayed
        mpy #6          ; multiplica RUNs por 12 (6*2(devido a PM))
                        ;                               -> Branch Delayed
        pac
        sbb
        sfl
        adds CoefNnL    ; Acc tem o endereco da tabela de COEF - 1
        samm ar3        ; ar3 aponta a tabela (no. de bits do cod. VLC)
        lacc #16
        sub *-
        samm TREG1
        lacc *+,ar2     ; carrega codigo VLC ; ar2 aponta para o buffer VLC
        bcndd Escape_N,EQ
        addt onevlc     ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        sacl *+,ar3     ; escreve codigo VLC
        lacc *,ar2      ; le No. de bits do codigo VLC
        sacl *+,ar1     ; escreve No. de bits do codigo VLC
        retd
        lacc ffffvlc    ;                               ->Branch Delayed
        samm ar0        ; run=-1                        ->Branch Delayed

Escape_N:               ;   '0000011'+last(1)+run(6)+level(8)
        mar *,ar2
        lamm TREG0      ; carrega numero de RUNs
        or esc_cod      ; esc_cod='0000 0001 1000 0000'
        sacl *+,2       ; escreve codigo VLC - [esc_cod+last+run]
        lacc #14
        sacl *+         ; escreve No. de bits do codigo VLC (7+1+6=14)
        lacb            ; carrega Level
        sfl
        sacl *+,7       ; escreve codigo VLC - [level(8)]
        lacl #8         ;
        sacl *+,ar1     ; escreve No. de bits do codigo VLC (8)
        retd
        lacc ffffvlc    ;                               ->Branch Delayed
        samm ar0        ; run=-1                        ->Branch Delayed

*******************************************************************************
```

```
* Funcao VLC (coeficiente DC)
*******************************************************************************
vlc_DC: ldp #Pic_PnI
        lacl Pic_PnI
        bcndd vlcDC_P,NEQ
vlcDC_I:
        lacc *,8,ar2
        sacl *+         ; escreve codigo vlc
        setc sxm
        sub #128
        retcd NEQ
        lacl #8         ;                                   -> Return Delayed
        sacl *+,ar1     ; escreve No. de bits do codigo VLC (8)
                        ;                                   -> Return Delayed
        mar *,ar2
        sbrk #2
        lacl #255
        sacl *
        adrk #2
        retd
        mar *,ar1       ;                                   -> Return Delayed
        sacl *          ;                                   -> Return Delayed

vlcDC_P:
        ;lacc *         ;                                   -> Branch Delayed
        ;setc sxm       ;                                   -> Branch Delayed

        bit *,0         ; regista em TC o bit de sinal
        abs             ; verifica se existe a necessidade de efectuar "clipping"
        sub #127
        bcndd vlcDC_x,LT
        lacl #127       ;                                   -> Branch Delayed
        nop             ;                                   -> Branch Delayed

        xc 1,TC
        neg

        sacl *

vlcDC_x:
;       lacc *          ; Carrega coef. DC no Acc
        bcndd DCNegLevel,LT
        ldp #Lesc_cod   ;                                   -> Branch Delayed
        nop             ;                                   -> Branch Delayed

DCPosLevel:
        sub #3
        bcndd DCEscape_P,GT
        lacc *,ar3      ;                                   -> Branch Delayed
        sfl             ; mult. Level por 2                 -> Branch Delayed

        adds CoefPL     ; Acc tem o endereco da tabela de COEF - 2
        samm ar3        ; ar3 aponta a tabela (codigo VLC corespondente)
        nop             ; -> Pipeline STALL devido a 'samm'
        nop             ; -> Pipeline STALL devido a 'samm'
        lacc *+,ar2     ; carrega codigo VLC ; ar2 aponta para o buffer VLC
        sacl *+,ar3     ; escreve codigo VLC
        retd
        lacc *,ar2      ; le No. de bits do codigo VLC
        sacl *+,ar4     ; escreve No. de bits do codigo VLC

DCEscape_P:             ;   '0000011'+last(1)+run(6)+level(8)
        mar *,ar2
        lacc Lesc_cod   ; Lesc_cod='0000 0001 1100 0000'
        sacl *+,2       ; escreve codigo VLC - [esc_cod+last+run]
        lacc #14
        sacl *+,ar1     ; escreve No. de bits do codigo VLC (7+1+6=14)

        lacc *,ar2
        sfl
        sacl *+,7       ; escreve codigo VLC - [level(8)]
        retd
        lacl #8         ;
        sacl *+,ar4     ; escreve No. de bits do codigo VLC (8)

DCNegLevel:
        add #3          ; |Level|-3>0 <=> -Level-3>0 <=> Level+3<0
        bcndd DCEscape_N,LT
        lacc *,ar3      ;                                   -> Branch Delayed
```

```
        neg             ;                               -> Branch Delayed

        sfl             ; mult. Level por 2
        adds CoefNL     ; Acc tem o endereco da tabela de COEF - 1
        samm ar3        ; ar3 aponta a tabela (no. de bits do cod. VLC)
        lacc #16
        sub *-
        samm TREG1
        lacc *+,ar2     ; carrega codigo VLC ; ar2 aponta para o buffer VLC
        addt onevlc     ;
        sacl *+,ar3     ; escreve codigo VLC
        retd
        lacc *,ar2      ; le No. de bits do codigo VLC
        sacl *+,ar4     ; escreve No. de bits do codigo VLC

DCEscape_N:             ;   '0000011'+last(1)+run(6)+level(8)
        mar *,ar2
        lacc Lesc_cod   ; Lesc_cod='0000 0001 1100 0000'
        sacl *+,2       ; escreve codigo VLC - [esc_cod+last+run]
        lacc #14
        sacl *+,ar1     ; escreve No. de bits do codigo VLC (7+1+6=14)

        lacc *,ar2
        sfl
        sacl *+,7       ; escreve codigo VLC - [level(8)]
        retd
        lacl #8         ;
        sacl *+,ar4     ; escreve No. de bits do codigo VLC (8)

*******************************************************************************
* Funcao PutBits - Concatena e envia os codigos em estruturas de 16 bits para o
*                  buffer de emissao.
*******************************************************************************
PutBits:
        lamm ar2
        lar ar7,#54h
        lar ar3,#13         ; ar3 -> TREG1 (Address=13)
        samm ARCR
        lar ar2,#buf_vlc    ; Buffer VLC [cod-Nbits-cod-Nbits-cod-....]

        ldp #TREG1buf
        mar *,ar2
        clrc sxm
                            ; restauro das variaveis de ambiente Accb e TREG1
        lacc TREG1buf
        samm TREG1
        lacc AccbHbuf,16
        sacb

        lact *+,ar3
        orb
next_w: sacb
        lacc *,16,ar2       ; Preenchimento do numero de "shifts" a efectuar
        sub *+,16,ar3
        bcndd fazout,LT
        sach *,ar2          ;                           -> Branch Delayed
        cmpr 0              ;                           -> Branch Delayed

        bcndd next_w,NTC
        lact *+,ar3         ;                           -> Branch Delayed
        orb                 ;                           -> Branch Delayed

* Saida da rotina: salvaguarda das variaveis de ambiente: Accb e TREG1
        lacb
        sach AccbHbuf
        retd
        lamm TREG1          ;                           -> Return Delayed
        sacl TREG1buf       ;                           -> Return Delayed

fazout  add #32,15          ; converte contador <=0 para >0 somando 16
        mar *,ar3
        sach *,ar7
        lacb                ; Output de 16 bits da trama H.263
        sach *,ar2          ; -> OUT para porto (tirar o sinal +...)
        sacl AccbHbuf
        lacc AccbHbuf,16
        sacb
        bcndd next_w,NTC
        lact *+,ar3         ;                           -> Branch Delayed
```

```
        orb                ;                            -> Branch Delayed

* Saida da rotina: salvaguarda das variaveis de ambiente: Accb e TREG1
*                  (o 'AccbHbuf' ja' foi guardado em cima)
        retd
        lamm TREG1         ;                            -> Return Delayed
        sacl TREG1buf      ;                            -> Return Delayed
```

C.10 FICHEIRO "dma_ini.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "dma_ini.asm"                                                     *
* DESCRIÇÃO: Este ficheiro efectua a inicializacao dos registos que comandam o *
*            funcionamento do bloco responsavel pela transferencia DMA da FPGA2*
*               Registos inicializados:                                       *
*                   RESET     - inicializacao da FPGA;                        *
*                   TARG_OFF  - endereco de destino/offset;                   *
*                   nCOR      - imagem policromatica/monocromatica.           *
*******************************************************************************
*
        .text
*
dma_ini:
        ;++++++++++++++++++
        ; Reset da FPGA2 +
        ;++++++++++++++++++
        zac
        samm RES

        ;++++++++++++++++++++
        ; Modo policromatico +
        ;++++++++++++++++++++
        zac
        samm nCOR

        ;+++++++++++++++++++++++++++++++++++++++++++++++++
        ; inicializacao do registo Target/Offset da FPGA2 +
        ;+++++++++++++++++++++++++++++++++++++++++++++++++
        lacc #2C56h             ; Target=2C00  Offset=8200-2C00=5600
        samm TARG_OFF

        ret
*
*******************************************************************************
* delay - funcao que efectua um ciclo de espera
*******************************************************************************
delay   lacc #07fffh
d_agai  sub #1
        bcnd d_agai,NEQ
        ret
*
```

C.11 FICHEIRO "mem_org.h"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "mem_org.h"                                                       *
* DESCRIÇÃO: Este ficheiro efectua a estruturacao e organizacao da memoria de  *
*            dados utilizada pelo processador                                 *
*                                                                             *
*******************************************************************************
*
        .data
*
```

```
        .include "Old_Pic.h"                    ; mem= 0x2c00 -> 0x7e7f
        .include "TCOEFVLC.h"                   ; mem= 0x7e80 -> 0x81fd
*
tempor  .word 0h                                ; mem= 0x81fe
unused  .word 0h                                ; mem= 0x81ff -> nao e´utilizado....
*
        .include "Init_Pic.h"                   ; mem= 0x8200 -> 0xd47f
*
end_pic .word 0h                                ; mem= 0xd480 -> nao e´utilizado....
*
        .include "vlc_tab.h"
*
buf_vlc .usect "EXT_DATA",1536                  ; 1MB c/ esc_cod: 6*64*4=1536
*
        .data
*
```

C.12 FICHEIRO "vlc_tab.h"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "vlc_tab.h"                                                       *
* DESCRIÇÃO: Este ficheiro contem as estruturas de dados e algumas tabelas de  *
*            VLC para codificar as imagens segundo a norma H.263.             *
*                                                                             *
*******************************************************************************
*
*   Significado das mascaras CBPC dos Macroblocos:
*       - Ao longo do calculo da DCT, o valor guardado do AccB(32 bits) tem
*         o seguinte significado:
*
*      0 1 2 3 4 5 6 7 8 9 a b c d e f   0 1 2 3 4 5 6 7 8 9 a b c d e f
*    |---------------.----------------|----------------.----------------|
*    | x x x x x x x x x x x x x x x x | x x x x x x x x 1 1 1 1 1 1 1 1 |
*    | Bloco tem Coeficientes nao DC   |                |Bloco c/ Coef DC|
*    |---------------------------------|---------------------------------|
*
*       - O valor de CBPC tera' o seguinte significado:
*
*      0  1  2  3  4  5  6  7    8  9  a  b  c  d  e  f
*    |-------------------------|-------------------------|
*    | 0  0  Y1 Y2 Y3 Y4 Cb Cr | 0  0  Y1 Y2 Y3 Y4 Cb Cr |
*    |  1= Bloco tem Coef. DC  |1=Bloco tem coeficientes |
*    |-------------------------|-------------------------|
*
        .data
*
*   CBPC ( 5 Gobs * 11 MacroBlocos = 55 Words )
CBPC .word 0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h
     .word 0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h
     .word 0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h
     .word 0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h
     .word 0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h
*
CBPCMskI                ; tabela de mascaras para as pictures INTRA
        .word 0a020h    ; tem o bit mais significativo a '1'
        .word 09010h
        .word 08808h
        .word 08404h
        .word 08202h
        .word 08101h
*
CBPCMskP                ; tabela de mascaras para as pictures INTER
        .word 2020h     ; tem o bit mais significativo a '0'
        .word 1010h
        .word 0808h
        .word 0404h
        .word 0202h
        .word 0101h
*
Pic_PnI .word 00000h    ; P_PnI=1 -> Picture Inter
```

```
                        ; P_PnI=0 -> Picture Intra
CountINTER .word 0000h  ; Numero de imagens INTER transmitidas desde que se
                        ; transmitiu a ultima imagem INTRA
MaxINTER   .word 0040h  ; Maximo numero de imagens INTER consecutivas (64)
Tem_Ref .word 00000h    ; Temporal Reference (Picture Header)
GOB_Nr  .word 00001h    ; Indica o numero do GOB a ser codificado em
                        ; cada instante
*
*                               CBPC(56)
MCBPC_I .word 08000h, 0001h     ; 00
        .word 02000h, 0003h     ; 00
        .word 04000h, 0003h     ; 00
        .word 06000h, 0003h     ; 00
*
*                               CBPC(56)
MCBPC_P .word 08000h, 0001h     ; 00            MB type 0
        .word 03000h, 0004h     ; 01
        .word 02000h, 0004h     ; 10
        .word 01400h, 0006h     ; 11
*
*                               Differential Value
DQUANT  .word 00000h, 0002h     ; -1
        .word 04000h, 0002h     ; -2
        .word 08000h, 0002h     ;  1
        .word 0c000h, 0002h     ;  2
*
*                               CBPY(1234)  indice CBPY_P=indice CBPY_I XOR 0f000h
CBPY    .word 03000h, 0004h     ; 0000
        .word 02800h, 0005h     ; 0001
        .word 02000h, 0005h     ; 0010
        .word 09000h, 0004h     ; 0011
        .word 01800h, 0005h     ; 0100
        .word 07000h, 0004h     ; 0101
        .word 00800h, 0006h     ; 0110
        .word 0b000h, 0004h     ; 0111
        .word 01000h, 0005h     ; 1000
        .word 00c00h, 0006h     ; 1001
        .word 05000h, 0004h     ; 1010
        .word 0a000h, 0004h     ; 1011
        .word 04000h, 0004h     ; 1100
        .word 08000h, 0004h     ; 1101
        .word 06000h, 0004h     ; 1110
        .word 0c000h, 0002h     ; 1111
*
*
notused .word 0000h     ; so para alinhar a pagina
onevlc  .word 0001h     ; usado para incrementos
fffvlc .word 0ffffh     ; usado para inicializar o no. de RUNs com -1
vlc_ptr .word 0000h     ; ponteiro para a posicao do buf_vlc antes da
*                       ; ultima escrita -> usado em last_vlc
CoefPnL .word 0000h
CoefNnL .word 0000h
CoefPL  .word 0000h
CoefNL  .word 0000h
*
esc_cod  .word 0180h   ; escape cod(*4)='0000 0001 1000 0000'
Lesc_cod .word 01c0h   ; last escape cod(*4)='0000 0001 1100 0000'
*
TREG1buf .word 0000h    ; buffer para guardar o TREG1 entre chamadas a putbits
AccbHbuf .word 0000h    ; buffer para guardar Accb_High
AccbLbuf .word 0000h    ; buffer para guardar Accb_LOW
*
```

D Listagem dos programas que implementam o Descodificador H.263

D.1 FICHEIRO "init.asm"

```
*******************************************************************************
*                          TRABALHO FINAL DE CURSO                            *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "init.asm"                                                        *
* DESCRIÇÃO: Este ficheiro contem o conjunto de procedimentos que constituem a *
*            inicializacao do Processador Digital de Sinal (DSP) TMS320C50.   *
*            Embora nao se utilize qualquer mecanismo suportadoem interrupcoes,*
*            optou-se por definir desde logo a tabela de interrupcoes corres-  *
*            pondente por forma a preencher o espaco de memoria correspondente *
*            com a uma instrucao de salto incondicional para uma funcao 'nula'.*
*            Na rotina de inicializacao propriamente dita e' tambem feita a    *
*            inicializaçao dos registos convenientes por forma a colocar o DSP *
*            no modo de funcionamento desejado.                                *
*******************************************************************************


*******************************************************************************
*    Tabela de Interrupcoes
*******************************************************************************

        .sect "vectors"         ; Endereco 00h da memoria de programa
RESET   B INIT                  ;RESET  (RESET)
        B NONE                  ;INT1   (EXTERNA MASCARAVEL)
        B NONE                  ;INT2   (EXTERNA MASCARAVEL)
        B NONE                  ;INT3   (EXTERNA MASCARAVEL)
        B NONE                  ;TINT   (TIMER)
        B NONE                  ;RINT   (RECEPCAO SERIE)
        B NONE                  ;XINT   (TRANSMISSAO SERIE)
        B NONE                  ;TRNT   (RECEPCAO TDM)
        B NONE                  ;TXNT   (TRANSMISSAO TDM)
        B NONE                  ;INT4   (EXTERNA MASCARAVEL)

        .space (14*16)          ;Zona de memoria reservada

        B NONE                  ;TRAP   (TRAP SOFTWARE)
        B NONE                  ;NMI    (EXTERNA NAO MASC)

        .space (2*16)           ;Zona de memoria reservada


*******************************************************************************
* ROTINA: "INIT"                                                              *
* DESCRICAO: Rotina de Inicializacao do Processador                           *
*            Inicializa:                                                      *
*               Estrutura de interrupcoes (INTM, IMR, IPTR)                   *
*               Controlo de modo (OVM, SXM, PM, AVIS, NDX, TRM)               *
*               Controlo de memoria (RAM, OVLY, CNF)                          *
*               Wait States (PDWSR, IOWSR, CWSR)                              *
*            Apos inicializar as variaveis do DSP procede 'a transferencia do *
*            programa para memoria interna e executa um salto para a rotina   *
*            "START".                                                         *
*******************************************************************************

        .sect   "ini"

INIT:   setc INTM               ; Inactiva as interrupcoes
        ldp #0                  ; pagina 0

        apl #0000h,IMR          ; Desactiva todas as interrupcoes

        setc OVM
        spm 1                   ; deslocamento do registo P de  bit para a esquerda
        apl #0008h,PMST
        opl #3eH,PMST           ; RAM=1, OVLY=0, TRM=1, NDX=1
        clrc CNF

   ; Programacao dos Wait States
```

```
        splk #0ffffH,PDWSR      ; 0WS para a memoria
        splk #00003H,IOWSR      ; 3WS para I/O
        splk #00010H,CWSR       ; BIG=1,  mode=1

   ; Inicializacao da pagina 0

        larp AR1                ; inicializa ARP -> AR1
        lar AR1,#96             ; carrega AR1 com o endereco inicial da pagina 0
        zap
        rpt #31                 ; repete 32 vezes
        sacl *+                 ; inicializacao de todas as posicoes com 'zero'

   ; Inicializacao das paginas 2 a 9

        larp AR1
        lar AR1,#256            ; carrega AR1 com o endereco inicial da pagina 2
        zap
        rpt #(128*8-1)
        sacl *+                 ; limpa todas as posicoes das paginas 2 a 9

   ; Carregamento do programa da RAM externa para a RAM interna

        larp AR1
        lar AR1,#prog_run
        rpt  #(prog_end-prog_start-1)
        blpd #prog_start, *+

        apl #0008h,PMST
        opl #01eH,PMST          ; RAM=1, OVLY=0, TRM=1, NDX=1

        B START

        .text
        .label prog_start
prog_run:
*******************************************************************************
* ROTINA DE INTERRUPCAO (nao utilizada)                                       *
*******************************************************************************
NONE:   rete
```

D.2 FICHEIRO "`main.asm`"

```
*******************************************************************************
*                          TRABALHO FINAL DE CURSO                            *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "main.asm"                                                        *
* DESCRIÇÃO: Este ficehiro ocupa o topo de toda a hierarquia de ficheiros, vis-*
*            to que engloba a rotina principal do programa.                   *
*            Este ficheiro e' executado apos a inicializacao do DSP. Nele, o  *
*            DSP comeca por invocar as varias rotinas que inicializam alguns  *
*            dos blocos do codificador, tais como o bloco da transformadas IDCT*
*            ("tranini") e do codificador de comprimento variavel ("vlc_ini"). *
*            De seguida, e' efectuado o preenchimento dos registos da FPGA3,  *
*            que determinarao a forma como se ira' processar a transferencia  *
*            dos dados da memoria do DSP para as DPRAMs, atraves de um ciclo de*
*            DMA. E´tambem efectuada a transferencia DMA correspondente 'a    *
*            imagem anterior. De seguida, o programa inicia o processo de des- *
*            codificacao da imagem seguinte.                                  *
*******************************************************************************
*
        .title "main.asm"
        .mmregs
        .version 50
*
        .include "mem_org.h"
        .include "init.asm"
        .include "dma_ini.asm"
        .include "transf.asm"
        .include "vlc.asm"
        .include "MBLayer.asm"
        .include "GOBLayer.asm"
        .include "PICLayer.asm"
```

```
*
LIM      .set    PA0
ORIG     .set    PA1
COR      .set    PA2
RES      .set    PA3
PORTO_IN .set    PA4
COUNTER  .set    PA5
OUT2     .set    PA6
OUT3     .set    PA7
*
         .text
*
START:  call traini
        call vlc_ini
        call dma_ini
other:
        ;+++++++++++++++++++++++++++++++++++++++
        ; inicializacao do endereco LIM da FPGA3 +
        ;+++++++++++++++++++++++++++++++++++++++
        lacc #end_pic
        samm LIM
        call delay
*       >>>>>> ciclo de DMA entre C50 -> FPGA3 <<<<<<

        call PIC_Layer
        b other

        idle
        .label prog_end
*
        .end
*
```

D.3 FICHEIRO "PIC_Layer.asm"

```
*******************************************************************************
*                          TRABALHO FINAL DE CURSO                            *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "PIC_Layer.asm"                                                   *
* DESCRIÇÃO: Este modulo faz o tratamento dos dados correspondentes a cada    *
*            'Picture'.                                                       *
*******************************************************************************
*
        .text
*
PIC_Layer:
        lar ar2,#NG0M1B1        ; aponta para o pixel da picture que esta' a
        ldp #Bit_Count          ; ser processado
        zac
        sacl Bit_Count

************
** HEADER **
************
*******************************************************************************
* Existem dois casos possiveis: o PSC aparece alinhado com uma palavra de 16
* bits ou aparece desalinhado com uma palavra de 16 bits mas alinhado com uma
* palavra de 8 bits,conforme se ilustra de seguida:
*
* Caso 1: 00000000 00000000             Caso 2: xxxxxxxx 00000000
*         100000TT TTTTTTPP                     00000000 100000TT
*         PPPPPPPP PPPQQQQQ                     TTTTTTPP PPPPPPPP
*         CE                                    PPPQQQQQ CE
*
* Assim, o programa comeca por testar o Caso1. Ao efectuar o processamento da
* 2a. palavra de 16 bits, caso este verifique que o codigo lido nao corresponde
* ao PSC, este deve verificar se nessa mesma palavra de 16 bits esta' presente
* o inicio do codigo PSC mas alinhado em palavras de 8 bits (Caso 2).
*******************************************************************************
Header:
    ;+++++++++++++++++++++++++++++++++
    ;+ Campo Picture Start Code - PSC +
    ;+++++++++++++++++++++++++++++++++
```

```
        calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bcndd H_1,EQ
        and #0ffh               ;                       <-- Branch Delayed

        bcndd Header,NEQ
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed
    ;+++++++++++++++++++++++++++
    ;+ Campo PSC (14 LSBs) & TR +
    ;+++++++++++++++++++++++++++
        calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

H_2:    bsar 2
        sub #32
        bcndd Header,NEQ
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed
    ;+++++++++++++++++++++++++++
    ;+ Campo TR (6 LSBs) & PTYPE +
    ;+++++++++++++++++++++++++++
        calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        and #0002h              ; Filtra o bit Pic_PnI
        sfr
        sacl Pic_PnI
    ;+++++++++++++++++++++++++++++++++++++++++
    ;+ Campo PTYPE (3 LSBs) & PQUANT & CPM & PEI +
    ;+++++++++++++++++++++++++++++++++++++++++
        calld GetBits
        lacl #10                ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sacl pqndc,3            ; shift left do Acc de 3+16=19 posicoes
        lacc pqndc,16
        lar ar0,#6
        mar *,ar0
PQuant1:bcndd PQuant1,GEQ       ; Quant = 2^(pqnd-1)
        sbrk #1                 ; pqndc = log2(Quant)+1
        sfl

        bd H_GOB
        lamm ar0                ;                       <-- Branch Delay
        sacl pqndc              ;                       <-- Branch Delay

*************************************
    ;+++++++++++++++++++++++++++++++++++++++++
    ;+ Campo PSC (6 LSBs) & TR & PTYPE (2 MSBs)+
    ;+++++++++++++++++++++++++++++++++++++++++
H_1:    calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bcndd H_1,EQ            ; verifica se existe multiplas words a '0000'
        bsar 10                 ;                       <-- Branch Delayed
        sub #32                 ;                       <-- Branch Delayed

        bcndd H_2,NEQ           ; verifica se os 8 LSBs correspondem ao inicio de um
PSC
        lacl Temp_VLC           ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed
    ;+++++++++++++++++++++++++++++++++
    ;+ Campo PTYPE (11 LSBs) & PQUANT+
    ;+++++++++++++++++++++++++++++++++
        calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        and #0200h              ; Filtra o bit Pic_PnI
        bsar 9
        sacl Pic_PnI

        lacc Temp_VLC,11        ; shift left do Acc de 11+16=27 posicoes
```

```
        sacl pqndc
        lacc pqndc,16
        lar ar0,#6
        mar *,ar0
PQuant2:bcndd PQuant2,GEQ       ; Quant = 2^(pqnd-1)
        sbrk #1                 ; pqndc = log2(Quant)+1
        sfl

        lamm ar0
        sacl pqndc
    ;++++++++++++++++++++
    ;+ Campo CPM & PEI +
    ;++++++++++++++++++++
        calld GetBits
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

H_GOB:  ldp #GOB_Nr
        zac                     ; inicializacao de GOB_Nr a '0'
        sacl GOB_Nr             ; pondente ao 1. Macrobloco
***********
** GOB 0 **
***********
GOB0:   call GOB_Layer
***********
** GOB 1 **
***********
GOB1:   call GOB_Layer
***********
** GOB 2 **
***********
GOB2:   call GOB_Layer
***********
** GOB 3 **
***********
GOB3:   call GOB_Layer
***********
** GOB 4 **
***********
GOB4:   call GOB_Layer
***********
** GOB 5 **
***********
GOB5:   call GOB_Layer
***********
** GOB 6 **
***********
GOB6:   call GOB_Layer
***********
** GOB 7 **
***********
GOB7:   call GOB_Layer
***********
** GOB 8 **
***********
GOB8:   call GOB_Layer

************
** TAIL **
************
*******************************************************************************
* Existem dois casos possiveis: o EOS aparece alinhado com uma palavra de 16
* bits ou aparece desalinhado com uma palavra de 16 bits mas alinhado com uma
* palavra de 8 bits,conforme se ilustra de seguida (x-representam 0 a 8 bits de
* ESTUF):
*
* Caso 1: 00000000 00000000             Caso 2: xxxxxxxx 00000000
*         11111100                              00000000 11111100
*
* Assim, o programa comeca por testar o Caso1. Ao efectuar o processamento da
* 2a. palavra de 16 bits, caso este verifique que o codigo lido nao corresponde
* ao EOS, este deve verificar se nessa mesma palavra de 16 bits esta' presente
* o inicio do codigo EOS mas alinhado em palavras de 8 bits (Caso 2).
*******************************************************************************
Tail:
    ;++++++++++++++++++++++++++
    ;+ Campo ESTUF (Stuffing) +
    ;++++++++++++++++++++++++++
        ldp #Bit_Count  ; Bit_Count tem o n. de bits ainda nao lidos do buffer.
```

```
        lacl Bit_Count  ; Dado o GBSC estar alinhado ao byte, o numero de bits
        and #07         ; de stuffing a retirar corresponde a (Bit_Count mod 8)
                        ; pelo que basta colocar o bit 3 a zero para determinar
                        ; quantos bits de stuffing deverao ser lidos.
        ccd GetBits,NEQ
        nop             ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed
*
    ;++++++++++++++++++++++++++++++++
    ;+ Campo End Of Sequence - EOS +
    ;++++++++++++++++++++++++++++++++
        calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bcndd H_3,EQ
        and #0ffh               ;                       <-- Branch Delayed

H_5     bcndd Tail,NEQ
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed
    ;++++++++++++++++++++++++++++++++ ; PSTUF é constituido por 2 bits a '0' por
    ;+ Campo EOS (14 LSBs) & PSTUF + ; forma a alinhara trama H.263 ao byte
    ;++++++++++++++++++++++++++++++++
        calld GetWord
        nop                     ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

H_4:    bsar 2
        sub #63
        retcd EQ
        nop                     ;                       <-- Return Delayed
        nop                     ;                       <-- Return Delayed

        bd Tail
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed

**************************************
    ;++++++++++++++++++++++++++++++++ ; PSTUF é constituido por 2 bits a '0' por
    ;+ Campo EOS (6 LSBs) & PSTUF + ; forma a alinhar a trama H.263 ao byte
    ;++++++++++++++++++++++++++++++++
H_3:    calld GetBits
        lacl #8                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bsar 2
        sub #63
        retcd EQ
        lacc Temp_VLC,8         ;                       <-- Return Delayed
        sacb                    ;                       <-- Return Delayed
  ; verifica se os 8 LSBs correspondem ao inicio de um EOS
        calld GetBits
        lacl #8                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        orb
        bd H_5
        and #0ffh               ;                       <-- Branch Delayed
**************************************
```

D.4 FICHEIRO "GOB_Layer.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "GOB_Layer.asm"                                                  *
* DESCRIÇÃO: Este modulo faz o tratamento dos dados correspondentes a cada GOB.*
*            E' executado apos se fazer o tratamento do cabecalho da "picture" *
*            e executa o modulo de tratamento dos macroblocos.               *
*******************************************************************************

        .text
```

```
*
GOB_Layer:
        ldp #GOB_Nr
        lacl GOB_Nr     ; no caso de se tratar do GOB numero zero, nao
        bcndd MB1,EQ    ; deve ser recebido qualquer header, pelo que
        add #1          ; se deve receber de imediato a informacao corres-
        sacl GOB_Nr     ; pondente ao 1. Macrobloco
************
** HEADER **
************
*******************************************************************************
* Existem dois casos possiveis: o GBSC aparece alinhado com uma palavra de 16
* bits ou aparece desalinhado com uma palavra de 16 bits mas alinhado com uma
* palavra de 8 bits, conforme se ilustra de seguida:
*
* Caso 1: 00000000 00000000             Caso 2: xxxxxxxx 00000000
*         1NNNNNII QQQQQ                        00000000 1NNNNNII
*                                               QQQQQ
*
* Assim, o programa comeca por testar o Caso1. Ao efectuar o processamento da
* 2a. palavra de 16 bits, caso este verifique que o codigo lido nao corresponde
* ao GBSC, este deve verificar se nessa mesma palavra de 16 bits esta' presente
* o inicio do codigo GBSC mas alinhado em palavras de 8 bits (Caso 2).
*******************************************************************************
GHeader:
    ;++++++++++++++++++++++++++
    ;+ Campo GSTUF (Stuffing) +
    ;++++++++++++++++++++++++++
        ldp #Bit_Count  ; Bit_Count tem o n. de bits ainda nao lidos do buffer.
        lacl Bit_Count  ; Dado o GBSC estar alinhado ao byte, o numero de bits
        and #07         ; de stuffing a retirar corresponde a (Bit_Count mod 8)
                        ; pelo que basta colocar o bit 3 a zero para determinar
                        ; quantos bits de stuffing deverao ser lidos.
        ccd GetBits,NEQ
        nop             ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed
*
    ;++++++++++++++++++++++++
    ;+ Campo GBSC (16 MSB) +
    ;++++++++++++++++++++++++
        calld GetWord
        nop             ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed

        bcndd G_1,EQ
        and #0ffh       ;                               <-- Branch Delayed

        bcndd GHeader,NEQ
        nop             ;                               <-- Branch Delayed
        nop             ;                               <-- Branch Delayed
    ;++++++++++++++++++++++++++++++++++
    ;+ Campo GBSC (9 LSBs) & GN & GFID +
    ;++++++++++++++++++++++++++++++++++
        calld GetWord
        nop             ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed

G_2:    bsar 7
        sub #1

        bcndd GQuant,EQ
        nop             ;                               <-- Branch Delay
        nop             ;                               <-- Branch Delay

        bd GHeader
        nop             ;                               <-- Branch Delay
        nop             ;                               <-- Branch Delay


************************************
    ;++++++++++++++++++++++++++++++++++
    ;+ Campo GBSC (1 LSBs) & GN & GFID +
    ;++++++++++++++++++++++++++++++++++
G_1:    calld GetBits
        lacl #8         ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed

        bsar 7
        sub #1
```

```
        bcndd GQuant,EQ
        lacc Temp_VLC,8 ;                               <-- Branch Delay
        sacb            ;                               <-- Branch Delay
  ; verifica se os 8 LSBs correspondem ao inicio de um EOS
        calld GetBits
        lacl #8         ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed

        bd G_2
        orb
        nop             ;                               <-- Branch Delay
****************************************
    ;+++++++++++++++
    ;+ Campo GQUANT +
    ;+++++++++++++++
GQuant:
        calld GetBits
        lacl #5         ;                               <-- Call Delayed
        nop             ;                               <-- Call Delayed

        lacc Temp_VLC,11        ; shift left do Acc de 11+16=27 posicoes
        sacl pqndc
        lacc pqndc,16
        lar ar0,#6
        mar *,ar0
GQuant1:bcndd GQuant1,GEQ       ; Quant = 2^(pqnd-1)
        sbrk #1                 ; pqndc = log2(Quant)+1
        sfl

        lamm ar0
        sacl pqndc

*******************
** MacroBloco MB1 **
*******************
MB1:    call MB_Layer
*******************
** MacroBloco MB2 **
*******************
MB2:    call MB_Layer
*******************
** MacroBloco MB3 **
*******************
MB3:    call MB_Layer
*******************
** MacroBloco MB4 **
*******************
MB4:    call MB_Layer
*******************
** MacroBloco MB5 **
*******************
MB5:    call MB_Layer
*******************
** MacroBloco MB6 **
*******************
MB6:    call MB_Layer
*******************
** MacroBloco MB7 **
*******************
MB7:    call MB_Layer
*******************
** MacroBloco MB8 **
*******************
MB8:    call MB_Layer
*******************
** MacroBloco MB9 **
*******************
MB9:    call MB_Layer
********************
** MacroBloco MB10 **
********************
MB10:   call MB_Layer
********************
** MacroBloco MB11 **
********************
MB11:   call MB_Layer

GOB_end  ret
```

D.5 FICHEIRO "MB_Layer.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "MB_Layer.asm"                                                     *
* DESCRIÇÃO: Este modulo faz o tratamento dos dados correspondentes a cada ma- *
*            crobloco. E' executado apos se fazer a descodificacao dos coefi-  *
*            cientes correspondentes aos cabecalhos do nivel de Picture, e GOB *
*            da imagem corrente, e executa as funcoes correspondentes a: - VLC,*
*            QuantInv, IDCT.                                                   *
*                                                                              *
*            Registos utilizados:     Ar0: No. de runs[vlc]; contador[invqnt]  *
*                                     Ar1: Coeficiente a tratar                *
*                                     Ar2: Pixel a tratar dentro da picture    *
*                                     Ar3:                                     *
*                                     Ar4:                                     *
*                                     Ar5: Mascaras de CBPC                    *
*                                     Ar6: Destino de escrita[idct]            *
*                                     Ar7: Contador de multiplicacoes[idct]    *
*                                     TREG0: [vlc]                             *
*                                     TREG1: [vlc],[invqnt]                    *
*                                     INDEX: [vlc]                             *
*                                     AccB: [vlc]                              *
*******************************************************************************
*
               .text
*
MB_Layer:
************
** HEADER **
************
        lar ar1,#Mblock
        lar ar5,#CBPCMsk
        ldp #Pic_PnI
        lacl Pic_PnI            ; '1'-> INTER ; '0'-> INTRA
        bcndd MB_P,NEQ
        lacc #8000h             ;                       <-- Branch Delayed

;+++++++++++++++
;+ Imagem INTRA +
;+++++++++++++++
MB_I:   sacl CBPCx              ; inicializa a variavel CBPCx
    ;+++++++++++++++
    ;+ Campo MCBPC +
    ;+++++++++++++++
        calld GetBits
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bcndd CBPYi,NEQ
        zac
        sacl CBPC

MCBPCi_A:
        calld GetBits
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sacl CBPC

CBPYi:
    ;++++++++++++++
    ;+ Campo CBPY +
    ;++++++++++++++
        calld GetBits
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sub #3
        bcndd CBPYiA,LT
        lacl #3ch               ;                       <-- Branch Delayed
        or CBPC                 ;                       <-- Branch Delayed

        bd CBPCxI
```

```
        sacl CBPC               ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed

CBPYiA: lacc Temp_VLC,2         ;
        calld GetBits
        sacb                    ;                       <-- Call Delayed
        lacl #2                 ;                       <-- Call Delayed

        orb
        sub #3
        bcndd CBPYiB,LT
        add #tbCBPYa            ;                       <-- Branch Delayed

        samm ar0
        mar *,ar0
        nop
        lacc *
        bd CBPCxI
        or CBPC                 ;                       <-- Branch Delayed
        sacl CBPC               ;                       <-- Branch Delayed

CBPYiB: lacc Temp_VLC,1
        calld GetBits
        sacb                    ;                       <-- Call Delayed
        lacl #1                 ;                       <-- Call Delayed

        orb
        sub #2
        bcndd CBPYiC,LT
        add #tbCBPYb            ;                       <-- Branch Delayed

        samm ar0
        mar *,ar0
        nop
        lacc *
        bd CBPCxI
        or CBPC                 ;                       <-- Branch Delayed
        sacl CBPC               ;                       <-- Branch Delayed

CBPYiC: calld GetBits
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bit Temp_VLC,15
        lacl #18h
        xc 1,TC
        add #0ch
        or CBPC
        sacl CBPC

CBPCxI  lacc #0bf00h            ; inicializacao de CBPCx - Em imagens INTRA,
                                ; todos os blocostem pelo menos o coeficiente DC
        bd BY1
        or CBPC                 ;                       <-- Branch delayed
        sacl CBPCx              ;                       <-- Branch delayed

;+++++++++++++++
;+ Imagem INTER +
;+++++++++++++++
MB_P:
    ;+++++++++++++
    ;+ Campo COD +
    ;+++++++++++++
        calld GetBits
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bcndd MCBPCp,EQ
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed

        mar *,ar6               ;
        retd
        adrk #192               ; soma 6*64=384=192+192 a ar6 por forma a
        adrk #192               ; ficar a apontar para o proximo macrobloco
                                ;                       <-- Return Delayed
    ;+++++++++++++++
    ;+ Campo MCBPC +
    ;+++++++++++++++
```

```
MCBPCp: calld GetBits
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bcndd CBPYp,NEQ
        zac
        sacl CBPC

MCBPCp_A:
        calld GetBits
        lacl #3                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        neg                     ; CBPC= -4.Temp_VLC + 4
        add #4
        sacl CBPC

        sub #3
        ccd GetBits,EQ          ; no caso de esta funcao ser executada, e visto
                                ; que se implementaram apenas os MB INTER tipo
                                ; 0, nao e' necessario ver os bits que dela re-
                                ; sultam visto os varios casos possiveis foram
                                ; ja' identificados.
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

CBPYp:
    ;+++++++++++++
    ;+ Campo CBPY +
    ;+++++++++++++
        calld GetBits
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sub #3
        bcndd CBPCxP,EQ         ; se Acc=Low, entao CBPY(INTER)=0000 e nao e'
                                ; necessario  fazer qqr operacao
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed

CBPYpA: lacc Temp_VLC,2         ;
        calld GetBits
        sacb                    ;                       <-- Call Delayed
        lacl #2                 ;                       <-- Call Delayed

        orb
        sub #3
        bcndd CBPYpB,LT
        add #tbCBPYa            ;                       <-- Branch Delayed

        samm ar0
        mar *,ar0
        nop
        lacc *
        xor #3ch                ; nega o codigo CBPY(Y) - 2 words/ciclos !!!
        bd CBPCxP
        or CBPC                 ;                       <-- Branch Delayed
        sacl CBPC               ;                       <-- Branch Delayed

CBPYpB: lacc Temp_VLC,1
        calld GetBits
        sacb                    ;                       <-- Call Delayed
        lacl #1                 ;                       <-- Call Delayed

        orb
        sub #2
        bcndd CBPYpC,LT
        add #tbCBPYb            ;                       <-- Branch Delayed

        samm ar0
        mar *,ar0
        nop
        lacc *
        xor #3ch                ; nega o codigo CBPY(Y) - 2 words/ciclos !!!
        bd CBPCxP
        or CBPC                 ;                       <-- Branch Delayed
        sacl CBPC               ;                       <-- Branch Delayed

CBPYpC: calld GetBits
```

```
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bit Temp_VLC,15
        lacl #36h
        xc 1,TC
        sub #0ch
        or CBPC
        sacl CBPC

CBPCxP: lacc CBPC,8
        sacl CBPCx

MVD:    calld GetBits           ; visto que nao se implementou a estimacao de
                                ; movimento, nao 'e necessario considerar os
                                ; vectores de movimento bastando apenas retirar
                                ; os 2 bits correspondentes aos vectores de
                                ; movimento nulos.
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

**************
** Bloco Y1 **
**************
BY1:    lar ar5,#CBPCMsk
        mar *,ar5
        lacl CBPCx
        and *,ar1
        bcndd Y1nNULL,NEQ
        nop             ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;+++++++++++++++++
    ;+ Bloco Y1 nulo +
    ;+++++++++++++++++
Y1NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar5                           ; Funcao invQuant nula
        bd BY2                              ; Funcao iDCT nula
        adrk #1
        lacl CBPCx
    ;+++++++++++++++++++++
    ;+ Bloco Y1 nao nulo +
    ;+++++++++++++++++++++
Y1nNULL call vlc
        mar *,ar5
        lacl CBPCx
        and *+,ar1
        and #0ffh
        bcndd Y1cCoef,NEQ
        mar *,ar1       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;++++++++++++++++++++++++++++
    ;+ Bloco Y1 so com coef. DC +
    ;++++++++++++++++++++++++++++
        lacl Pic_PnI     ; Funcao invQuant "fast"
        bcndd Y1_DCP,NEQ ; verifica se coef DC e' Intra ou Inter
        lacl #17         ;                              -> Branch Delayed
        sub pqndc        ;                              -> Branch Delayed

Y1_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y1fIDCT,NEQ
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        lacl #128
        bd Y1fIDCT
        sacl *,3,ar1    ; 128*3=1024                    -> Branch Delayed
        lacc *,13       ;                               -> Branch Delayed

Y1_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y1DCneg,LT
Y1DCpos sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
```

```
        bd Y1fIDCT
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y1DCneg ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y1fIDCT rpt #63         ; limita-se a espalhar o coeficiente DC
        sach *+         ; pelo resto do bloco
        bd BY2
        mar *,ar5
        lacl CBPCx

    ;++++++++++++++++++++++++++++++++++
    ;+ Bloco Y1 com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++++
Y1cCoef ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacl Pic_PnI
        bcndd Y1coefP,NEQ
        lar ar0,#63     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

Y1coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y1next,NEQ
        lar ar0,#62     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y1coefP ;lar ar0,#63
Y1next  bcnd Y1Nzero,NEQ
        mar *+,ar0
        banzd Y1next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y1sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y1Nzero bcndd Y1coefn,LT
Y1coefp sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed

        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y1next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y1sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y1coefn ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y1next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

*       b Y1sIDCT
        sbrk #64

Y1sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar5
        ldp #CBPCx
```

```
        lacl CBPCx

**************
** Bloco Y2 **
**************
BY2:    and *,ar1
        bcndd Y2nNULL,NEQ
        nop             ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;+++++++++++++++++
    ;+ Bloco Y2 nulo +
    ;+++++++++++++++++
Y2NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar5                           ; Funcao invQuant nula
        bd BY3                              ; Funcao iDCT nula
        adrk #1
        lacl CBPCx
    ;+++++++++++++++++++++
    ;+ Bloco Y2 nao nulo +
    ;+++++++++++++++++++++
Y2nNULL call vlc
        mar *,ar5
        lacl CBPCx
        and *+,ar1
        and #0ffh
        bcndd Y2cCoef,NEQ
        mar *,ar1       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;++++++++++++++++++++++++++++
    ;+ Bloco Y2 so com coef. DC +
    ;++++++++++++++++++++++++++++
        lacl Pic_PnI     ; Funcao invQuant "fast"
        bcndd Y2_DCP,NEQ ; verifica se coef DC e' Intra ou Inter
        lacl #17         ;                              -> Branch Delayed
        sub pqndc        ;                              -> Branch Delayed

Y2_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y2fIDCT,NEQ
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        lacl #128
        bd Y2fIDCT
        sacl *,3,ar1    ; 128*3=1024                    -> Branch Delayed
        lacc *,13       ;                               -> Branch Delayed

Y2_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y2DCneg,LT
Y2DCpos sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd Y2fIDCT
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y2DCneg ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y2fIDCT rpt #63         ; limita-se a espalhar o coeficiente DC
        sach *+         ; pelo resto do bloco
        bd BY3
        mar *,ar5
        lacl CBPCx

    ;+++++++++++++++++++++++++++++++++
    ;+ Bloco Y2 com 64 coef. nao nulos +
    ;+++++++++++++++++++++++++++++++++
Y2cCoef ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
```

```
        samm TREG1
        lacl Pic_PnI
        bcndd Y2coefP,NEQ
        lar ar0,#63     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

Y2coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y2next,NEQ
        lar ar0,#62     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y2coefP ;lar ar0,#63
Y2next  bcnd Y2Nzero,NEQ
        mar *+,ar0
        banzd Y2next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y2sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y2Nzero bcndd Y2coefn,LT
Y2coefp sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed

        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y2next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y2sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y2coefn ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y2next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

*       b Y2sIDCT
        sbrk #64

Y2sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar5
        ldp #CBPCx
        lacl CBPCx

**************
** Bloco Y3 **
**************
BY3:    and *,ar1
        bcndd Y3nNULL,NEQ
        nop             ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;+++++++++++++++++
    ;+ Bloco Y3 nulo +
    ;+++++++++++++++++
Y3NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar5                           ; Funcao invQuant nula
        bd BY4                              ; Funcao iDCT nula
        adrk #1
        lacl CBPCx
    ;+++++++++++++++++++++
    ;+ Bloco Y3 nao nulo +
    ;+++++++++++++++++++++
Y3nNULL call vlc
        mar *,ar5
```

```
        lacl CBPCx
        and *+,ar1
        and #0ffh
        bcndd Y3cCoef,NEQ
        mar *,ar1       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;+++++++++++++++++++++++++++
    ;+ Bloco Y3 so com coef. DC +
    ;+++++++++++++++++++++++++++
        lacl Pic_PnI     ; Funcao invQuant "fast"
        bcndd Y3_DCP,NEQ ; verifica se coef DC e' Intra ou Inter
        lacl #17         ;                               -> Branch Delayed
        sub pqndc        ;                               -> Branch Delayed

Y3_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y3fIDCT,NEQ
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        lacl #128
        bd Y3fIDCT
        sacl *,3,ar1    ; 128*3=1024                    -> Branch Delayed
        lacc *,13       ;                               -> Branch Delayed

Y3_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y3DCneg,LT
Y3DCpos sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd Y3fIDCT
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y3DCneg ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y3fIDCT rpt #63         ; limita-se a espalhar o coeficiente DC
        sach *+         ; pelo resto do bloco
        bd BY4
        mar *,ar5
        lacl CBPCx

    ;+++++++++++++++++++++++++++++++++
    ;+ Bloco Y3 com 64 coef. nao nulos +
    ;+++++++++++++++++++++++++++++++++
Y3cCoef ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacl Pic_PnI
        bcndd Y3coefP,NEQ
        lar ar0,#63     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

Y3coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y3next,NEQ
        lar ar0,#62     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y3coefP ;lar ar0,#63
Y3next  bcnd Y3Nzero,NEQ
        mar *+,ar0
        banzd Y3next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
```

```
        nop             ;                               -> Branch Delayed

        bd Y3sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y3Nzero bcndd Y3coefn,LT
Y3coefp sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed

        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y3next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y3sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y3coefn ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y3next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

*       b Y3sIDCT
        sbrk #64

Y3sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar5
        ldp #CBPCx
        lacl CBPCx

**************
** Bloco Y4 **
**************
BY4:    and *,ar1
        bcndd Y4nNULL,NEQ
        nop             ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;+++++++++++++++++
    ;+ Bloco Y4 nulo +
    ;+++++++++++++++++
Y4NULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar5                           ; Funcao invQuant nula
        bd BCb                              ; Funcao iDCT nula
        adrk #1
        lacl CBPCx
    ;+++++++++++++++++++++
    ;+ Bloco Y4 nao nulo +
    ;+++++++++++++++++++++
Y4nNULL call vlc
        mar *,ar5
        lacl CBPCx
        and *+,ar1
        and #0ffh
        bcndd Y4cCoef,NEQ
        mar *,ar1       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;++++++++++++++++++++++++++++
    ;+ Bloco Y4 so com coef. DC +
    ;++++++++++++++++++++++++++++
        lacl Pic_PnI    ; Funcao invQuant "fast"
        bcndd Y4_DCP,NEQ ; verifica se coef DC e' Intra ou Inter
        lacl #17        ;                               -> Branch Delayed
        sub pqndc       ;                               -> Branch Delayed

Y4_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y4fIDCT,NEQ
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        lacl #128
```

```
        bd Y4fIDCT
        sacl *,3,ar1    ; 128*3=1024                    -> Branch Delayed
        lacc *,13       ;                               -> Branch Delayed

Y4_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd Y4DCneg,LT
Y4DCpos sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd Y4fIDCT
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y4DCneg ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
Y4fIDCT rpt #63         ; limita-se a espalhar o coeficiente DC
        sach *+         ; pelo resto do bloco
        bd BCb
        mar *,ar5
        lacl CBPCx

    ;++++++++++++++++++++++++++++++++++
    ;+ Bloco Y4 com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++++
Y4cCoef ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacl Pic_PnI
        bcndd Y4coefP,NEQ
        lar ar0,#63     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

Y4coefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Y4next,NEQ
        lar ar0,#62     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

Y4coefP ;lar ar0,#63
Y4next  bcnd Y4Nzero,NEQ
        mar *+,ar0
        banzd Y4next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y4sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y4Nzero bcndd Y4coefn,LT
Y4coefp sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed

        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y4next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd Y4sIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Y4coefn ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
```

```
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Y4next,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

*       b Y4sIDCT
        sbrk #64

Y4sIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar5
        ldp #CBPCx
        lacl CBPCx

**************
** Bloco Cb **
**************
BCb:    and *,ar1
        bcndd CbnNULL,NEQ
        nop             ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;+++++++++++++++++
    ;+ Bloco Cb nulo +
    ;+++++++++++++++++
CbNULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar5                           ; Funcao invQuant nula
        bd BCr                              ; Funcao iDCT nula
        adrk #1
        lacl CBPCx
    ;+++++++++++++++++++++
    ;+ Bloco Cb nao nulo +
    ;+++++++++++++++++++++
CbnNULL call vlc
        mar *,ar5
        lacl CBPCx
        and *+,ar1
        and #0ffh
        bcndd CbcCoef,NEQ
        mar *,ar1       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;++++++++++++++++++++++++++
    ;+ Bloco Cb so com coef. DC +
    ;++++++++++++++++++++++++++
        lacl Pic_PnI     ; Funcao invQuant "fast"
        bcndd Cb_DCP,NEQ ; verifica se coef DC e' Intra ou Inter
        lacl #17         ;                              -> Branch Delayed
        sub pqndc        ;                              -> Branch Delayed

Cb_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd CbfIDCT,NEQ
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        lacl #128
        bd CbfIDCT
        sacl *,3,ar1    ; 128*3=1024                    -> Branch Delayed
        lacc *,13       ;                               -> Branch Delayed

Cb_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd CbDCneg,LT
CbDCpos sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd CbfIDCT
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

CbDCneg ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
```

```
        lacc *,13
CbfIDCT rpt #63         ; limita-se a espalhar o coeficiente DC
        sach *+         ; pelo resto do bloco
        bd BCr
        mar *,ar5
        lacl CBPCx

    ;++++++++++++++++++++++++++++++++++
    ;+ Bloco Cb com 64 coef. nao nulos +
    ;++++++++++++++++++++++++++++++++++
CbcCoef ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacl Pic_PnI
        bcndd CbcoefP,NEQ
        lar ar0,#63     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

CbcoefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd Cbnext,NEQ
        lar ar0,#62     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

CbcoefP ;lar ar0,#63
Cbnext  bcnd CbNzero,NEQ
        mar *+,ar0
        banzd Cbnext,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd CbsIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

CbNzero bcndd Cbcoefn,LT
Cbcoefp sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed

        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Cbnext,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd CbsIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Cbcoefn ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Cbnext,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

*       b CbsIDCT
        sbrk #64

CbsIDCT call idctslw    ; Funcao iDCT "slow"

        mar *,ar5
        ldp #CBPCx
        lacl CBPCx

**************
** Bloco Cr **
**************
BCr:    and *,ar1
        bcndd CrnNULL,NEQ
        nop             ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
```

```
    ;+++++++++++++++++
    ;+ Bloco Cr nulo +
    ;+++++++++++++++++
CrNULL  adrk #64        ; soma 64 a ar1     ; Funcao VLC nula
        mar *,ar5                           ; Funcao invQuant nula
        bd MB_end                           ; Funcao iDCT nula
        adrk #1
        lacl CBPCx
    ;+++++++++++++++++++++
    ;+ Bloco Cr nao nulo +
    ;+++++++++++++++++++++
CrnNULL call vlc
        mar *,ar5
        lacl CBPCx
        and *+,ar1
        and #0ffh
        bcndd CrcCoef,NEQ
        mar *,ar1       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed
    ;++++++++++++++++++++++++++
    ;+ Bloco Cr so com coef. DC +
    ;++++++++++++++++++++++++++
        lacl Pic_PnI     ; Funcao invQuant "fast"
        bcndd Cr_DCP,NEQ ; verifica se coef DC e' Intra ou Inter
        lacl #17         ;                              -> Branch Delayed
        sub pqndc        ;                              -> Branch Delayed

Cr_DCI  lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *          ; quant. do coeficiente INTRAdc
        sub #2040
        bcndd CrfIDCT,NEQ
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        lacl #128
        bd CrfIDCT
        sacl *,3,ar1    ; 128*3=1024                    -> Branch Delayed
        lacc *,13       ;                               -> Branch Delayed

Cr_DCP  ;lacc #17       ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        ;sub pqndc      ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacc *,16
        bcndd CrDCneg,LT
CrDCpos sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed
        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *,ar1
        bd CrfIDCT
        lacc *,13       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

CrDCneg ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *,ar1
                        ; Funcao iDCT "fast"
        lacc *,13
CrfIDCT rpt #63         ; limita-se a espalhar o coeficiente DC
        sach *+         ; pelo resto do bloco
        bd MB_end
        mar *,ar5
        lacl CBPCx

    ;+++++++++++++++++++++++++++++++++
    ;+ Bloco Cr com 64 coef. nao nulos +
    ;+++++++++++++++++++++++++++++++++
CrcCoef ldp #pqndc      ; Funcao invQuant nao nula
        lacc #17        ; shift=16-QUANT =16-(pqndc-1) =17-pqndc
        sub pqndc       ; passo de quant=2*2^QUANT =2^pqndc
        samm TREG1
        lacl Pic_PnI
        bcndd CrcoefP,NEQ
        lar ar0,#63     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

CrcoefI lacc *,3        ; shift  de 3 bits (x8), correspondente ao passo de
        sacl *+         ; quant. do coeficiente INTRAdc
        sub #2040
```

```
        bcndd Crnext,NEQ
        lar ar0,#62     ;                               -> Branch Delayed
        lacc *,16       ;                               -> Branch Delayed

        lacl #128
        sacl *,3,ar1    ; 128*3=1024
        lacc *,16

CrcoefP ;lar ar0,#63
Crnext  bcnd CrNzero,NEQ
        mar *+,ar0
        banzd Crnext,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd CrsIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

CrNzero bcndd Crcoefn,LT
Crcoefp sfl             ; Acc=2*Level                   -> Branch delayed
        add one,16      ; Acc=2*Level+1                 -> Branch delayed

        satl            ; Acc=Quant*(2*Level+1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Crnext,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

        bd CrsIDCT
        sbrk #64        ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

Crcoefn ;sfl            ; Acc=2*Level                   -> Branch delayed
        sub two,16      ; Acc=2*Level-1  (Correccao de "add one,16")
        satl            ; Acc=Quant*(2*Level-1) ; Quant=pqndc/2
        sacl *+,ar0
        banzd Crnext,*-,ar1
        lacc *,16       ;                               -> Branch Delayed
        nop             ;                               -> Branch Delayed

*       b CrsIDCT
        sbrk #64

CrsIDCT call idctslw    ; Funcao iDCT "slow"

MB_end: ldp #Pic_PnI
        lacl Pic_PnI
        bcndd MB_endI,EQ
        lacc #383       ;                               --> Branch Delayed

MB_endP:
        samm BRCR       ; counter-1= 6*64 - 1
        lar ar1,#Mblock
        mar *,ar2
        rptb end_loop-1
          lacc *,16,ar1
          add *+,16,ar2
          sach *+
end_loop:
        retd
        nop             ;                               <-- Return Delayed
        nop             ;                               <-- Return Delayed

MB_endI:
        mar *,ar2
        rpt #383        ; counter-1= 6*64 - 1
        bldd #Mblock,*+

        retd
        nop             ;                               <-- Return Delayed
        nop             ;                               <-- Return Delayed
```

D.6 FICHEIRO "transf.asm"

```
*******************************************************************************

*                           TRABALHO FINAL DE CURSO                           *

*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "transf.asm"                                                      *
* DESCRIÇÃO: Este ficheiro contem a chamada 'a rotina que efectua a inicializa-*
*            cao da funcao que efectua a transformadas utilizadas na codifica- *
*            cao: DCT.                                                        *
*            Efectuam tambem a definicao de algumas variaveis necessarias ao  *
*            processamento.                                                   *
*                                                                             *
*******************************************************************************
*
        .include "idct.asm"
*
        .text
*
*******************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
*******************************************************************************
traini: ldp #rndoff
        lacc #8000h
        sacl rndoff
        lacl #1
        sacl one
        sacl two,1
*
        call idctini
*
*******************************************************************************
* Definicao de variaveis
*******************************************************************************
Mblock  .usect  "B1",384        ; macrobloco corrente
temp    .usect  "B1",64         ; resultados temporarios das transformadas
rndoff  .usect  "B1",1          ; roundoff factor
src     .usect  "B1",1          ; ponteiro para o inicio do bloco a transformar
dst     .usect  "B1",1          ; endereco do destino da operacao corrente
nextsrc .usect  "B1",1          ; endereco do destino final da idct
*
```

D.7 FICHEIRO "idct.asm"

```
*******************************************************************************
*                           TRABALHO FINAL DE CURSO                           *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "idct.asm"                                                        *
* DESCRIÇÃO: Este ficheiro contem a rotina que executa o calculo da transfor-  *
*            mada inversa de coseno discreta - IDCT. O algoritmo utilizado foi *
*            o baseado na multiplicacao de matrizes da forma:                 *
*               [f] = [C].[F].[C]T                                            *
*            O calculo deste produto é realizado por aplicacao consecutiva de  *
*            duas operacoes da forma:                                         *
*               [Y] = ( [Md].[Mp]T )T                                         *
*            em que [Md] e' a matriz dos pixels armazenada em memoria de dados *
*            e Mp e' a matriz dos coeficientes armazenada em memoria de progra-*
*            ma.                                                              *
*            O calcula do IDCT e' obtido sa seguinte forma:                   *
*               [f] = {( [F].[C]T )T .[C]T}T                                  *
*                   =  [C].[F].[C]T                                           *
*                                                                             *
*******************************************************************************

        .text
*******************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
```

```
********************************************************************************
*
idctini:
        spm 1                   ; Programacao do registo PM para 1 "shift"
        clrc CNF
        lar ar1,#icoeff         ; coeficientes da IDCT
        rpt #(iedata-iidata-1)
        bldd #iidata,*+
        retd
        setc CNF                ; a instrucao "mac" tem um dos operandos arma-
                                ; zenado em memoria de programa
        nop                     ;                              <- Branch delayed

********************************************************************************
* Funcao que executa o calculo da IDCT (versao mais lenta)
********************************************************************************
idctslw:setc sxm
        ldp #src
        sar ar1,src
        sar ar1,dst             ; destino e' igual 'a origem
        adrk #64
        sar ar1,nextsrc         ; ponteiro para o proximo bloco
        lar ar7,#1              ; ar7 : dimension-1
*
*       inicio do ciclo (executado 2 vezes)
*
        lar ar6,#temp           ; endereco de destino (primeira coluna)
*
* primeiro conjunto de coeficientes
*
imult:  lar ar1,src
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
        rptz 7
        mac ct00,*+             ; acc = 0 ,preg= x0 * ct00
        lta *,ar6               ; acumula o ultimo produto e carrega o reg. P
        adds rndoff
        sach *+,ar1             ; guarda resultado parcial
*
* segundo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
```

```
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct10,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
* terceiro conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct20,*+
        lta *,ar6
        adds rndoff
```

```
        sach *+,ar1
*
* quarto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct30,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
* quinto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
```

```
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct40,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
* sexto conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct50,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
* setimo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
```

```
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct60,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
*
* oitavo conjunto de coeficientes
*
        lar ar1,src
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar1
        rptz 7
        mac ct70,*+
        lta *,ar6
        adds rndoff
        sach *+,ar7             ; ar7 := dimensao
*
*       reinicio do ciclo
*
        lar ar1,#temp
        banzd imult,*-,ar1
        sar ar1,src             ;                       <-- Branch Delayed
        lar ar6,dst             ;                       <-- Branch Delayed
*
        retd
        lar ar1,nextsrc         ; por forma a que na proxima chamada a esta
                                ; funcao, ar1 aponte para o proximo bloco
        mar *,ar1
*******************************************************************************
*
*  DEFINICAO E DECLARACAO DA TABELA DE COEFICIENTES
*
```

```
*************************************************************************
        .asect "DCT_COEF",0fe00h
                                ; esta tabela deve ser armazenada em  memoria
                                ; de programa atraves de um comando 'CNFP'
*
        .label iidata           ; coeficientes IDCT
ct00    .word  11585
ct01    .word  16069
ct02    .word  15137
ct03    .word  13623
ct04    .word  11585
ct05    .word   9102
ct06    .word   6270
ct07    .word   3196
ct10    .word  11585            ; 11585 = (1/2) * 2^(-1/2) no formato Q15
ct11    .word  13623
ct12    .word   6270            ; 6269 = (1/2) * sin(Pi/8) no formato Q15
ct13    .word  -3196
ct14    .word -11585
ct15    .word -16069
ct16    .word -15137
ct17    .word  -9102
ct20    .word  11585
ct21    .word   9102
ct22    .word  -6270            ; 15136 = (1/2) * cos(Pi/8) no formato Q15
ct23    .word -16069
ct24    .word -11585
ct25    .word   3196
ct26    .word  15137
ct27    .word  13623
ct30    .word  11585
ct31    .word   3196
ct32    .word -15137
ct33    .word  -9102
ct34    .word  11585
ct35    .word  13623
ct36    .word  -6270
ct37    .word -16069
ct40    .word  11585
ct41    .word  -3196            ; 3196 = (1/2) * sin(Pi/16) no formato Q15
ct42    .word -15137
ct43    .word   9102
ct44    .word  11585
ct45    .word -13623
ct46    .word  -6270
ct47    .word  16069
ct50    .word  11585
ct51    .word  -9102            ; 9102 = (1/2) * sin(3Pi/16) no formato Q15
ct52    .word  -6270
ct53    .word  16069
ct54    .word -11585
ct55    .word  -3196
ct56    .word  15137
ct57    .word -13623
ct60    .word  11585
ct61    .word -13623            ; 13622 = (1/2) * cos(3Pi/16) no formato Q15
ct62    .word   6270
ct63    .word   3196
ct64    .word -11585
ct65    .word  16069
ct66    .word -15137
ct67    .word   9102
ct70    .word  11585
ct71    .word -16069            ; 16069 = (1/2) * cos(Pi/16) no formato Q15
ct72    .word  15137
ct73    .word -13623
ct74    .word  11585
ct75    .word  -9102
ct76    .word   6270
ct77    .word  -3196
        .label iedata           ; fim da tabela de coeficientes
*************************************************************************
*
*  Definicao do espaco para armazenamento desta tabela
*
*************************************************************************
icoeff  .usect "B0",64          ; coeficientes iDCT (Bloco de memoria 'B0')
*
```

D.8 FICHEIRO "vlc.asm"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "vlc.asm"                                                         *
* DESCRIÇÃO: Este ficheiro contem a rotina que executa a descodificacao dos   *
*            coeficientes de comprimento variavel, de acordo com o descrito na *
*            norma H.263.                                                     *
*                                                                             *
*               Variaveis:                                                    *
*                  Acch -> 0                                                  *
*                  Acch -> Numero de RUNs                                     *
*                  Ar1 -> pixel de destino                                    *
*******************************************************************************

        .text
*
*******************************************************************************
* Funcao de inicializacao - chamada apenas uma vez no inicio
*******************************************************************************
vlc_ini:
        lacl #7
        samm INDX
        mar *,ar0
        lar ar0,#F0ptr
        lacc #F0
        sacl *+
        lacc #F1
        sacl *+

        lar ar0,#F0000ptr
        lacc #F0000
        sacl *+
        lacc #F0001
        sacl *+
        lacc #F0010
        sacl *+
        lacc #F0011
        sacl *+
        lacc #F0100
        sacl *+
        lacc #F0101
        sacl *+
        lacc #F0110
        sacl *+
        lacc #F0111
        sacl *+

        lar ar0,#tb_F0000ptr
        lacc #tb_F0000
        sacl *+
        lacc #tb_F0000A
        sacl *+
        lacc #tb_F0000AA
        sacl *+
        lacc #tb_F0000C
        sacl *+
        lacc #tb_F0000CA
        sacl *+
        lacc #tb_F0001
        sacl *+
        lacc #tb_F0001A
        sacl *+
        lacc #tb_F0010
        sacl *+
        lacc #tb_F0011
        sacl *+
        lacc #tb_F0100
        retd
        sacl *+
        nop

*******************************************************************************
* Funcao VLC
```

```
*****************************************************************************
vlc:    lacl Pic_PnI
        bcndd vlcINTER,NEQ
        setc sxm            ;                           <-- Branch Delayed
        ldp #Bit_Count      ;                           <-- Branch Delayed

vlcINTRA:
        mar *,ar1
        rptz #63            ; inicializa o bloco a zero
        sacl *+,ar1

        sbrk #64            ; coloca o Ar0 a apontar para o inicio do bloco
        calld GetBits
        lacl #8             ;                           <-- Call Delayed
        nop                 ;                           <-- Call Delayed

        sacl LEVEL
        lacl CBPCx          ; verifica se o bloco tem coeficientes NAO DC
        and #0ffh
        mar *,ar5
        and *,ar1
        zac                 ; este zac nao influencia a instrucao 'xc'
        xc 1,EQ
        lacl #1             ; LAST=1 se o coef. DC e' o´unico nao nulo

        bd DC_VLC
        sacl LAST           ;                           <-- Branch Delayed
        zac                 ; Runs = 0                  <-- Branch Delayed

vlcINTER:
        mar *,ar1
        rptz #63            ; inicializa o bloco a zero
        sacl *+

        sbrk #64            ; coloca o Ar0 a apontar para o inicio do bloco

        calld GetBits
        lacl #3             ;                           <-- Call Delayed
        mar *,ar0           ;                           <-- Call Delayed

        bsar 2
        add #F0ptr
        samm ar0
        mar *,ar0
        nop
        lacc *,ar1
        cala                ; Termina com as variaveis LEVEL e LAST inicializadas
                            ; e com o valor de RUN no Acc
        sacb
        bit LAST,15
        bcndd DC_VLC,EQ,TC  ; DConly se RUN=0 && Last=1
        lacb                ;                           <-- Branch Delayed
        nop                 ;                           <-- Branch Delayed

        mar *,ar5
        lacc *
        or CBPCx
        sacl CBPCx
        lacb

*******************
DC_VLC:                     ; Escreve o coef.DC lido anteriormente e le o
                            ; proximo coef, caso Last=0
        ccd VLCcoef,EQ
        mar *,ar1           ;   aponto P1               <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P2               <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P3               <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        adrk #8             ;   aponto P4               <-- Call Delayed
        sub #1              ;
```

```
        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P5              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P6              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P7              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P8              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P9              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P10             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        adrk #8             ;   aponto P11             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P12             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P13             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P14             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P15             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P16             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P17             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P18             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P19             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P20             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P21             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        adrk #8             ;   aponto P22             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P23             <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P24             <-- Call Delayed
        sub #1              ;
```

```
        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P25              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P26              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P27              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P28              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P29              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P30              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P31              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P32              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P33              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P34              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P35              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P36              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P37              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P38              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P39              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P40              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P41              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P42              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P43              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        adrk #8             ;   aponto P44              <-- Call Delayed
```

```
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P45              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P46              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P47              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P48              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P49              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P50              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P51              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P52              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P53              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P54              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        adrk #8             ;   aponto P55              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P56              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P57              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P58              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *+,ar1          ;   aponto P59              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P60              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0-,ar1         ;   aponto P61              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        adrk #8             ;   aponto P62              <-- Call Delayed
        sub #1              ;

        ccd VLCcoef,EQ
        mar *0+,ar1         ;   aponto P63              <-- Call Delayed
        sub #1              ;

Last_VLC:                   ; se chegar aqui e' sinal que o coeficiente 64 esta'
```

```
                            ; marcado com LAST = 1. Assim, tem apenas que
                            ; escrever o valor de Level lido anteriormente.
        mar *+,ar1          ;   aponto P64
        lacl LEVEL
        retd
        sacl *+             ; coloca Ar1 a apontar para o inicio do proximo bloco
        sbrk #64            ; coloca Ar1 a apontar para o inicio do bloco corrente

*****************************************************************************
* VLCcoef - funcao que faz a descodificacao do codigo VLC. Retorna o numero de
*           RUNs em AccLow, e o LEVEL na variavel LEVEL. Caso o coeficiente
*           corresponda a um coeficiente marcado com o sinal LAST a funcao e´
*           terminada de imediato.
*           Retorna com ARP=0.
*****************************************************************************
VLCcoef:
        mar *,ar1
        lacl LAST
        bcndd IsLast,NEQ
        lacl LEVEL
        sacl *

        calld GetBits
        lacl #3                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        bsar 2
        add #F0ptr
        samm ar0
        mar *,ar0
        nop
        lacc *
        cala
        retd
        mar *,ar1               ;                       <-- Return Delayed
        nop                     ;                       <-- Return Delayed

IsLast: pop                     ; Por forma a retirar o endereco de PC do 'stack'
                                ; inserido aquando da chamada "ccd VLCcoef,EQ"
        lamm ar1                ; coloca Ar1 a apontar para o inicio do bloco
        and #0ffc0h             ; corrente
        retd
        samm ar1                ;                       <-- Return Delayed
        mar *,ar1               ;                       <-- Return Delayed

******************************
F1:     bit Temp_VLC,14         ; bit(1)=0 -> TC=0  ;  bit(1)=1 -> TC=1
        bcndd F1_1,TC
        zac
        sacl LAST

F1_0:   bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacl #1
        xc 1,TC
        neg
        retd
        sacl LEVEL              ;                       <-- Return Delayed
        zac                     ;                       <-- Return Delayed

F1_1:   bit Temp_VLC,15         ; bit(0)=0 -> TC=0  ;  bit(0)=1 -> TC=1
        bcndd F1_11,TC
        nop                     ;                       <-- Branch Delayed
        nop                     ;                       <-- Branch Delayed

F1_10:  calld GetBits
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sfl                     ; LEVEL= -2*bit(0)+1
        neg
        add #1
        retd
        sacl LEVEL              ;                       <-- Return Delayed
        lacl #1                 ;                       <-- Return Delayed

F1_11:  calld GetBits
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed
```

```
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        sfr
        sacb
        add #1
        xc 1,TC
        neg
        sacl LEVEL
        lacb
        sfl                     ; Runs= -2*bit(1)+2
        retd
        neg                     ;                       <-- Return Delayed
        add #2                  ;                       <-- Return Delayed


*******************************
F0:     lacc Temp_VLC,1
        calld GetBits
        sacb                    ;                       <-- Call Delayed
        lacl #1                 ;                       <-- Call Delayed

        orb
        add #F0000ptr
        samm ar0
        mar *,ar0
        nop
        lacc *
        cala

        ret

F0000:  calld GetBits
        lacl #3                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed
                                ; indices 0..7

        sub #3                  ; indices -3..4
        bcndd F0000_a,LT        ; salta no caso de indices -3,-2,-1 (antigos 0,1,2)
                                ; ficam os indices 0,1,2,3,4 (antigos 3,4,5,6,7)
        sub #1                  ; ficam os indices -1,0,1,2,3(antigos 3,4,5,6,7)
        sfl                     ; multiplicacao por 4
                                ;                       <-- Branch Delay

        bcndd Escape,LT         ; salta no caso de indices -4 (antigo 3)
        sfl                     ; multiplicacao por 4 (cont.)
                                ; ficam os indices 0,4,8,12 (antigos 4,5,6,7)
        sacb                    ;                       <-- Branch Delay

        calld GetBits
        lacl #3                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        sfr                     ; retira o bit de sinal
        orb
        sub #1
        bcndd F0000_b,LT        ; verifica se index=0
        sacb                    ;                       <-- Branch Delayed
        sfl                     ; multiplica por 3      <-- Branch Delayed

        addb
        add tb_F0000ptr         ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop                     ; devido 'a instrucao 'samm'
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

Escape:
        calld GetBits
        lacl #15                ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sacb
```

```
        bit Temp_VLC,8          ; verifica se o valor de Level e' negativo
        lacl #0ffh              ; filtra o campo Level
        andb
        xc 2,TC
        or #0ff00h

        sacl LEVEL

        lacb
        bsar 14                 ; filtra o campo Last
        sacl LAST

        lacb
        bsar 8                  ; filtra o campo Run
        retd
        and #3fh                ; elimina o campo Last

F0000_a:lacl Temp_VLC
        sub #1                  ; index= -1,0,1 (antigos index 0,1,2)
        bcndd F0000_c,GT
        lacc Temp_VLC,4         ; index= -1,0           <-- Branch Delay
        sacb                    ;                       <-- Branch Delay

        calld GetBits
        lacl #4                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        orb
        sfr                     ; retira o bit de sinal
        sub #4
        bcndd F0000_aa,LT
        sacb                    ;                       <-- Branch Delay
        sfl                     ; multiplica por 3      <-- Branch Delay

        addb
        add tb_F0000Aptr        ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0000_aa:
        lacc Temp_VLC           ; recupera o suposto bit de sinal anterior
                                ; index = 4..7
        sub #4                  ; index = 0..3
        sacb
        sfl                     ;multiplica por 3
        addb
        add tb_F0000AAptr       ; AccLow tem o endereco da tabela
        samm ar0
        calld GetBits
        lacl #1                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0000_b:
        calld GetBits           ; le o bit de sinal que falta
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
                                ; referente ao valor lido anteriormente!
        lacl #1                 ;                       <-- Call Delayed
```

```
        zac
        sacl LAST
        lacl #8
        xc 1,NTC                ; calcula o valor absoluto de LEVEL
        add #1

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop
        xc 1,TC
        neg

        retd
        sacl LEVEL              ; falta ainda definir o sinal
        zac

F0000_c:
        calld GetBits           ; le o bit de sinal que falta
        lacl #5                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        sfr                     ; retira o bit de sinal
        sacb
        sub #7
        bcndd F0000_ca,GT
        lacb                    ; multiplica por 3      <-- Branch Delay
        sfl                     ;                       <-- Branch Delay

        addb
        add tb_F0000Cptr        ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop

        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0000_ca:
        lacc Temp_VLC           ; recupera o suposto bit de sinal anterior

        sub #16                 ; index = 0..15
        sacb
        sfl                     ; multiplica por 3
        addb
        add tb_F0000CAptr       ; AccLow tem o endereco da tabela
        samm ar0
        calld GetBits           ; le o bit de sinal que falta
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0001:  calld GetBits
        lacl #5                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        sfr                     ; retira o bit de sinal
        sub #3
        sacb
        bcndd F0001_a,LT
        lacb                    ; multiplica por 3      <-- Branch Delayed
        sfl                     ;                       <-- Branch Delayed

        addb
```

```
        add tb_F0001ptr         ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop                     ; devido 'a instrucao 'samm'
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0001_a:lacl Temp_VLC           ; recupera o suposto bit de sinal anterior
        sacb
        sfl                     ; multiplica por 3
        addb
        add tb_F0001Aptr        ; AccLow tem o endereco da tabela
        calld GetBits
        samm ar0                ;                       <-- Call Delayed
        lacl #1                 ;                       <-- Call Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+


F0010:  calld GetBits
        lacl #4                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        sfr                     ; retira o bit de sinal
        sacb                    ; multiplica por 3
        sfl
        addb
        add tb_F0010ptr         ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop                     ; devido 'a instrucao 'samm'
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0011:  calld GetBits
        lacl #3                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        sfr                     ; retira o bit de sinal
        sacb                    ; multiplica por 3
        sfl
        addb
        add tb_F0011ptr         ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop                     ; devido 'a instrucao 'samm'
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+
```

```
F0100:  calld GetBits
        lacl #3                 ;                       <-- Call Delayed
        mar *,ar0               ;                       <-- Call Delayed

        sfr                     ; retira o bit de sinal
        sacb                    ; multiplica por 3
        sfl
        addb
        add tb_F0100ptr         ; AccLow tem o endereco da tabela
        samm ar0
        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        nop                     ; devido 'a instrucao 'samm'
        lacl *+
        sacl LAST
        lacl *+
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl *+

F0101:  calld GetBits
        lacl #2                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bit Temp_VLC,14         ; bit(1)=0 -> TC=0  ;  bit(1)=1 -> TC=1
        bcndd F0101a,NTC
        sacb                    ; <- Para a funcao F0101a    <-- Branch Delayed
        zac                     ; <- Para ambas as funcoes   <-- Branch Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        sacl LAST
        lacl #1
        xc 1,TC
        neg

        retd
        sacl LEVEL
        lacl #5

F0101a: ;sacb
        ;zac
        calld GetBits           ; leitura do bit de sinal
        sacl LAST               ;                       <-- Call Delayed
        lacl #1                 ;                       <-- Call Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacb
        add #2
        xc 1,TC
        neg

        sacl LEVEL
        lacb
        retd
        neg                     ; Runs+1= -lacb+1
        add #1

F0110:  calld GetBits
        lacl #2                 ;  desnecessario        <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacl #1
        xc 1,TC
        neg

        sacl LEVEL
        lacc #0
        sacl LAST

        lacc Temp_VLC
        sfr                     ; retira o bit de sinal
        retd
        neg
        add #4

F0111:  calld GetBits
```

```
        lacl #1                 ;                       <-- Call Delayed
        nop                     ;                       <-- Call Delayed

        bit Temp_VLC,15         ; s=0 -> TC=0  ;  s=1 -> TC=1
        lacl #1
        sacl LAST
        xc 1,TC
        neg

        retd
        sacl LEVEL              ;                       <-- Return Delayed
        lacl #0                 ;                       <-- Return Delayed
*
*******************************************************************************
* GetBits - funcao que retorna em AccLow o numero de bits da trama H.263
*           requerido em AccLow aquando da sua chamada.
*******************************************************************************
GetBits:
        setc sxm
        ldp #Bit_Count
        samm TREG1              ; TREG1 <- Numero de bits requeridos
        neg
        add Bit_Count
        sacl Bit_Count          ; Bit_Count = Bit_Count - AccLow
        bcndd save_bits,GEQ
        clrc sxm                ;                       <-- Branch Delayed
        lact Buf_VLC            ;                       <-- Branch Delayed

        sach Temp_VLC
waitbits:
        lamm COUNTER
        nop
        sub #1
        bcnd waitbits,LEQ

        lamm PORTO_IN           ; Input de 16 bits da trama H.263
        nop
        sacl Buf_VLC
        setc sxm
        lacl Bit_Count
        add #16
        sacl Bit_Count
        sub #16
        neg                     ; AccLow= abs(Bit_Count)
        samm TREG1
        clrc sxm
        lact Buf_VLC
        sacl Buf_VLC
        bsar 16
        retd
        or Temp_VLC
        sacl Temp_VLC

GetWord:
        setc sxm
        ldp #Bit_Count
        lacl Bit_Count
        sub #16
        sacl Bit_Count          ; Bit_Count = Bit_Count - 16
        bcndd save_bits,GEQ
        clrc sxm                ;                       <-- Branch Delayed
        lacc Buf_VLC,16         ;                       <-- Branch Delayed

        sach Temp_VLC
W_waitbits:
        lamm COUNTER
        nop
        sub #1
        bcnd W_waitbits,LEQ

        lamm PORTO_IN           ; Input de 16 bits da trama H.263
        nop
        sacl Buf_VLC
        setc sxm
        lacl Bit_Count
        add #16
        sacl Bit_Count
        sub #15
        neg                     ; AccLow= abs(Bit_Count)-1
```

```
        samm TREG1
        clrc sxm
        lact Buf_VLC
        sacl Buf_VLC,1
        bsar 15
        retd
        or Temp_VLC
        sacl Temp_VLC

save_bits:
        sacl Buf_VLC
        retd
        sach Temp_VLC           ; sath em 2 ciclos !!!!
        lacl Temp_VLC
```

D.9 FICHEIRO "dma_ini.asm"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "dma_ini.asm"                                                     *
* DESCRIÇÃO: Este ficheiro efectua a inicializacao dos registos que comandam o *
*            funcionamento do bloco responsavel pela transferencia DMA da FPGA3*
*               Registos inicializados:                                       *
*                   RESET - inicializacao da FPGA;                            *
*                   ORIG  - endereco de inicio;                               *
*                   COR   - imagem policromatica/monocromatica.               *
*******************************************************************************
*
        .text
*
dma_ini:
        ;+++++++++++++++++
        ; Reset da FPGA3 +
        ;+++++++++++++++++
        zac
        samm RES

        ;+++++++++++++++++++++++++++++++++++++++
        ; inicializacao do endereco ORIG da FPGA3 +
        ;+++++++++++++++++++++++++++++++++++++++
        lacc #NG0M1B1-40h
        samm ORIG

        ;+++++++++++++++++
        ; Reset da FPGA3 +
        ;+++++++++++++++++
        zac
        samm RES

        ;+++++++++++++++++++++++++++++++++++++++
        ; inicializacao da variavel COR da FPGA3 +
        ;+++++++++++++++++++++++++++++++++++++++
        lacl #1
        samm COR

        ret
*
*******************************************************************************
* delay - funcao que efectua um ciclo de espera
*******************************************************************************
delay   lacc #000fh
d_agai  sub #1
        bcnd d_agai,NEQ
        ret
*
```

D.10 FICHEIRO "mem_org.h"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "mem_org.h"                                                       *
* DESCRIÇÃO: Este ficheiro efectua a estruturacao e organizacao da memoria de  *
*            dados utilizada pelo processador                                 *
*                                                                             *
*******************************************************************************
*
        .data
*
        .include "Picture.h"    ; mem= 0x2c00 -> 0xc080
end_pic     .word 0h            ; mem= 0xc080 -> nao e´utilizado....
*
Bit_Count   .word 0h
Buf_VLC     .word 0h
Temp_VLC    .word 0h
LEVEL       .word 0h
LAST        .word 0h
*
Pic_PnI     .word 0h
pqndc       .word 0h
GOB_Nr      .word 0h
one         .word 1h            ; posicao de memoria inicializada com '1'
two         .word 2h            ; posicao de memoria inicializada com '2'
*
*       - O valor de CBPCx tera' o seguinte significado:
*
*      0  1  2  3  4  5  6  7    8  9  a  b  c  d  e  f
*    |-------------------------|-------------------------|
*    | 0  0  Y1 Y2 Y3 Y4 Cb Cr | 0  0  Y1 Y2 Y3 Y4 Cb Cr |
*    |  1= Bloco tem Coef. DC  |1=Bloco tem coeficientes |
*    |-------------------------|-------------------------|
*
*
*       - Significado do par 'Pic_PnI' - 'CBPC(i)'
*
*         CBPY(i) | Picture | Significado
*         --------|---------|--------------------------------------
*            0    |  INTRA  | Bloco SO' com coeficiente DC
*            1    |  INTRA  | Bloco com coeficientes (nao DC)
*            0    |  INTER  | Bloco com TODOS os coeficientes NULOS
*            1    |  INTER  | Bloco com coeficientes (DC e/ou outros)
*
*       - Correspondencia entre as variaveis 'CBPC' e 'CBPCx'
*
*          Picture | Significado
*          --------|--------------------------------------
*            INTRA  | 1 0 1 1 1 1 1 1 | 0 0 x x x x x x
*            INTRA  | 1 0 1 1 1 1 1 1 | 0 0 x x x x x x
*            INTER  | 1 0 x x x x x x | 0 0 ? ? ? ? ? ?
*            INTER  | 1 0 x x x x x x | 0 0 ? ? ? ? ? ?
*
*       LEGENDA:  'x' - codigo CBPC
*                 '?' - valor inicializado a 'zero'. O valor correcto sera'
*                       depois determinado na funcao "vlc".
*
CBPC        .word 0000h
CBPCx       .word 0000h
*
CBPCMsk     .word 2020h         ; tabela de mascaras CBPC
            .word 1010h
            .word 0808h
            .word 0404h
            .word 0202h
            .word 0101h
*
        .include "TcoefVLC.h"
*
        .data
```

D.11 FICHEIRO "TcoefVLC.h"

```
*******************************************************************************
*                            TRABALHO FINAL DE CURSO                          *
*******************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                        *
*******************************************************************************
* FICHEIRO: "TcoefVLC.h"                                                       *
* DESCRIÇÃO: Ficheiro com as tabelas de coeficientes variaveis TCOEF para os   *
*            varios casos considerados na norma H.263. Ver tabela 13/H.263 da  *
*            norma (pag.21).                                                   *
*                                                                              *
*******************************************************************************
*
        .data
*
*   Tabela de ponteiros para funcoes
*
F0ptr           .word 0h
F1ptr           .word 0h
F0000ptr        .word 0h
F0001ptr        .word 0h
F0010ptr        .word 0h
F0011ptr        .word 0h
F0100ptr        .word 0h
F0101ptr        .word 0h
F0110ptr        .word 0h
F0111ptr        .word 0h
tb_F0000ptr     .word 0h
tb_F0000Aptr    .word 0h
tb_F0000AAptr   .word 0h
tb_F0000Cptr    .word 0h
tb_F0000CAptr   .word 0h
tb_F0001ptr     .word 0h
tb_F0001Aptr    .word 0h
tb_F0010ptr     .word 0h
tb_F0011ptr     .word 0h
tb_F0100ptr     .word 0h

*******************************************************************************
* Tabelas de coeficientes
*******************************************************************************
tb_F0000    .word 0001h, 0001h, 0018h ; "000010001s" - Last=1, Level=1, Run=24
            .word 0001h, 0001h, 0017h ; "000010010s" - Last=1, Level=1, Run=23
            .word 0001h, 0001h, 0016h ; "000010011s" - Last=1, Level=1, Run=22
            .word 0001h, 0001h, 0015h ; "000010100s" - Last=1, Level=1, Run=21
            .word 0001h, 0001h, 0014h ; "000010101s" - Last=1, Level=1, Run=20
            .word 0001h, 0001h, 0013h ; "000010110s" - Last=1, Level=1, Run=19
            .word 0001h, 0001h, 0012h ; "000010111s" - Last=1, Level=1, Run=18
            .word 0001h, 0001h, 0011h ; "000011000s" - Last=1, Level=1, Run=17
            .word 0001h, 0002h, 0000h ; "000011001s" - Last=1, Level=2, Run=0
            .word 0000h, 0001h, 0016h ; "000011010s" - Last=0, Level=1, Run=22
            .word 0000h, 0001h, 0015h ; "000011011s" - Last=0, Level=1, Run=21
            .word 0000h, 0001h, 0014h ; "000011100s" - Last=0, Level=1, Run=20
            .word 0000h, 0001h, 0013h ; "000011101s" - Last=0, Level=1, Run=19
            .word 0000h, 0001h, 0012h ; "000011110s" - Last=0, Level=1, Run=18
            .word 0000h, 0001h, 0011h ; "000011111s" - Last=0, Level=1, Run=17

tb_F0000A   .word 0001h, 0001h, 001ch ; "0000000100s" - Last=1, Level=1, Run=28
            .word 0001h, 0001h, 001bh ; "0000000101s" - Last=1, Level=1, Run=27
            .word 0001h, 0001h, 001ah ; "0000000110s" - Last=1, Level=1, Run=26
            .word 0001h, 0001h, 0019h ; "0000000111s" - Last=1, Level=1, Run=25
            .word 0000h, 0002h, 0009h ; "0000001000s" - Last=0, Level=2, Run=9
            .word 0000h, 0002h, 0008h ; "0000001001s" - Last=0, Level=2, Run=8
            .word 0000h, 0002h, 0007h ; "0000001010s" - Last=0, Level=2, Run=7
            .word 0000h, 0002h, 0006h ; "0000001011s" - Last=0, Level=2, Run=6
            .word 0000h, 0002h, 0005h ; "0000001100s" - Last=0, Level=2, Run=5
            .word 0000h, 0003h, 0003h ; "0000001101s" - Last=0, Level=3, Run=3
            .word 0000h, 0003h, 0002h ; "0000001110s" - Last=0, Level=3, Run=2
            .word 0000h, 0004h, 0001h ; "0000001111s" - Last=0, Level=4, Run=1

tb_F0000AA  .word 0001h, 0002h, 0001h ; "00000000100s" - Last=1, Level=2,  Run=1
            .word 0001h, 0003h, 0000h ; "00000000101s" - Last=1, Level=3,  Run=0
            .word 0000h, 000bh, 0000h ; "00000000110s" - Last=0, Level=11, Run=0
            .word 0000h, 000ah, 0000h ; "00000000111s" - Last=0, Level=10, Run=0

tb_F0000C   .word 0000h, 000ch, 0000h ; "00000100000s" - Last=0, Level=12, Run=0
```

```
            .word 0000h, 0005h, 0001h ; "000001000001s" - Last=0, Level=5,  Run=1
            .word 0000h, 0001h, 0017h ; "000001000010s" - Last=0, Level=1,  Run=23
            .word 0000h, 0001h, 0018h ; "000001000011s" - Last=0, Level=1,  Run=24
            .word 0001h, 0001h, 001dh ; "000001001000s" - Last=1, Level=1,  Run=29
            .word 0001h, 0001h, 001eh ; "000001001001s" - Last=1, Level=1,  Run=30
            .word 0001h, 0001h, 001fh ; "000001001010s" - Last=1, Level=1,  Run=31
            .word 0001h, 0001h, 0020h ; "000001001011s" - Last=1, Level=1,  Run=32

tb_F0000CA  .word 0000h, 0006h, 0001h ; "0000010100000s" - Last=0, Level=6,  Run=1
            .word 0000h, 0004h, 0002h ; "0000010100001s" - Last=0, Level=4,  Run=2
            .word 0000h, 0003h, 0004h ; "0000010100010s" - Last=0, Level=3,  Run=4
            .word 0000h, 0003h, 0005h ; "0000010100011s" - Last=0, Level=3,  Run=5
            .word 0000h, 0003h, 0006h ; "0000010100100s" - Last=0, Level=3,  Run=6
            .word 0000h, 0002h, 000ah ; "0000010100101s" - Last=0, Level=2,  Run=10
            .word 0000h, 0001h, 0019h ; "0000010100110s" - Last=0, Level=1,  Run=25
            .word 0000h, 0001h, 001ah ; "0000010100111s" - Last=0, Level=1,  Run=26
            .word 0001h, 0001h, 0021h ; "0000010110000s" - Last=1, Level=1,  Run=33
            .word 0001h, 0001h, 0022h ; "0000010110001s" - Last=1, Level=1,  Run=34
            .word 0001h, 0001h, 0023h ; "0000010110010s" - Last=1, Level=1,  Run=35
            .word 0001h, 0001h, 0024h ; "0000010110011s" - Last=1, Level=1,  Run=36
            .word 0001h, 0001h, 0025h ; "0000010111100s" - Last=1, Level=1,  Run=37
            .word 0001h, 0001h, 0026h ; "0000010111101s" - Last=1, Level=1,  Run=38
            .word 0001h, 0001h, 0027h ; "0000010111110s" - Last=1, Level=1,  Run=39
            .word 0001h, 0001h, 0028h ; "0000010111111s" - Last=1, Level=1,  Run=40

tb_F0001    .word 0001h, 0001h, 0010h ; "00010011s" - Last=1, Level=1, Run=16
            .word 0001h, 0001h, 000fh ; "00010100s" - Last=1, Level=1, Run=15
            .word 0001h, 0001h, 000eh ; "00010101s" - Last=1, Level=1, Run=14
            .word 0001h, 0001h, 000dh ; "00010110s" - Last=1, Level=1, Run=13
            .word 0001h, 0001h, 000ch ; "00010111s" - Last=1, Level=1, Run=12
            .word 0001h, 0001h, 000bh ; "00011000s" - Last=1, Level=1, Run=11
            .word 0001h, 0001h, 000ah ; "00011001s" - Last=1, Level=1, Run=10
            .word 0001h, 0001h, 0009h ; "00011010s" - Last=1, Level=1, Run=9
            .word 0000h, 0001h, 000eh ; "00011011s" - Last=0, Level=1, Run=14
            .word 0000h, 0001h, 000dh ; "00011100s" - Last=0, Level=1, Run=13
            .word 0000h, 0002h, 0002h ; "00011101s" - Last=0, Level=2, Run=2
            .word 0000h, 0003h, 0001h ; "00011110s" - Last=0, Level=3, Run=1
            .word 0000h, 0005h, 0000h ; "00011111s" - Last=0, Level=5, Run=0

tb_F0001A   .word 0000h, 0001h, 0010h ; "000100000s" - Last=0, Level=1, Run=16
            .word 0000h, 0001h, 000fh ; "000100001s" - Last=0, Level=1, Run=15
            .word 0000h, 0002h, 0004h ; "000100010s" - Last=0, Level=2, Run=4
            .word 0000h, 0002h, 0003h ; "000100011s" - Last=0, Level=2, Run=3
            .word 0000h, 0007h, 0000h ; "000100100s" - Last=0, Level=7, Run=0
            .word 0000h, 0006h, 0000h ; "000100101s" - Last=0, Level=6, Run=0

tb_F0010    .word 0001h, 0001h, 0008h ; "0010000s" - Last=1, Level=1, Run=8
            .word 0001h, 0001h, 0007h ; "0010001s" - Last=1, Level=1, Run=7
            .word 0001h, 0001h, 0006h ; "0010010s" - Last=1, Level=1, Run=6
            .word 0001h, 0001h, 0005h ; "0010011s" - Last=1, Level=1, Run=5
            .word 0000h, 0001h, 000ch ; "0010100s" - Last=0, Level=1, Run=12
            .word 0000h, 0001h, 000bh ; "0010101s" - Last=0, Level=1, Run=11
            .word 0000h, 0001h, 000ah ; "0010110s" - Last=0, Level=1, Run=10
            .word 0000h, 0004h, 0000h ; "0010111s" - Last=0, Level=4, Run=0

tb_F0011    .word 0001h, 0001h, 0004h ; "001100s" - Last=1, Level=1, Run=4
            .word 0001h, 0001h, 0003h ; "001101s" - Last=1, Level=1, Run=3
            .word 0001h, 0001h, 0002h ; "001110s" - Last=1, Level=1, Run=2
            .word 0001h, 0001h, 0001h ; "001111s" - Last=1, Level=1, Run=1

tb_F0100    .word 0000h, 0001h, 0009h ; "010000s" - Last=0, Level=1, Run=9
            .word 0000h, 0001h, 0008h ; "010001s" - Last=0, Level=1, Run=8
            .word 0000h, 0001h, 0007h ; "010010s" - Last=0, Level=1, Run=7
            .word 0000h, 0001h, 0006h ; "010011s" - Last=0, Level=1, Run=6

tbCBPYc     .word 0018h  ; "000010"  - CBPY_i=0110
            .word 0024h  ; "000011"  - CBPY_i=1001
tbCBPYb     .word 0020h  ; "00010"   - CBPY_i=1000
            .word 0010h  ; "00011"   - CBPY_i=0100
            .word 0008h  ; "00100"   - CBPY_i=0010
            .word 0004h  ; "00101"   - CBPY_i=0001
tbCBPYa     .word 0000h  ; "0011"    - CBPY_i=0000
            .word 0030h  ; "0100"    - CBPY_i=1100
            .word 0028h  ; "0101"    - CBPY_i=1010
            .word 0038h  ; "0110"    - CBPY_i=1110
            .word 0014h  ; "0111"    - CBPY_i=0101
            .word 0034h  ; "1000"    - CBPY_i=1101
            .word 000ch  ; "1001"    - CBPY_i=0011
            .word 002ch  ; "1010"    - CBPY_i=1011
```

```
            .word 001ch  ; "1011"     - CBPY_i=0111
            .word 003ch  ; "11"       - CBPY_i=1111
```

E Co-processador de aquisição e formatação de dados

E.1 ESQUEMA GERAL

O esquema geral do co-processador de aquisição e formatação de dados encontra-se apresentado na página seguinte.

Colocar aqui o esquema do co-processador de aquisição

E.2 DESCRIÇÃO DO CIRCUITO IMPLEMENTADO NA FPGA1

E.2.1 Descrição dos portos de entrada e saída

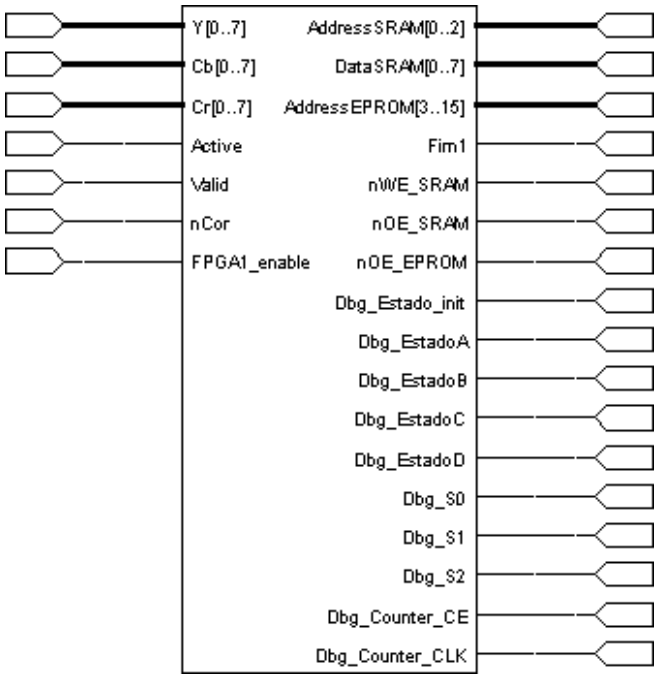


Figura E.1 - Descrição dos portos de entrada e saída da FPGA1.

PORTO	**DESCRIÇÃO**
`Active`	Sinal proveniente do conversor A/D assinalando a presença de uma nova linha
`Valid`	Sinal proveniente do conversor A/D que assinala a presença de um novo *pixel*
`Y`	Componente Y do sinal de vídeo digital
C_B	Componente C_B do sinal de vídeo digital
C_R	Componente C_R do sinal de vídeo digital
`FPGA1_Enable`	Sinal proveniente da FPGA2 que desencadeia o início de uma nova transferência
`nCor`	Entrada que indica se o funcionamento desejado é policromático ou não
`AddressEPROM`	Barramento de endereços para as EPROM
`AddressSRAM`	3 bits menos significativos do contador de 16 bits e que seguem directamente para o barramento de endereços da SRAM
`DataSRAM`	Byte a escrever na SRAM, correspondente a uma das três componentes do sinal de vídeo digital
`Fim1`	Sinal que indica, à FPGA2, a conclusão da transferência de uma nova imagem para a SRAM
`nOE_EPROM`	Sinal que activa a saída das EPROMs durante a fase de transferência, desactivando-as nas restantes ocasiões
`nOE_SRAM`	Sinal que activa as saídas da SRAM durante a transferência da imagem, desactivando-as nas restantes ocasiões
`nWE_EPROM`	Sinal que controla a escrita na SRAM
`Dbg…`	Sinais que correspondem a nós internos da FPGA1. Estão disponíveis apenas para efeitos de teste

E.2.2 Esquema Lógico

O esquema geral do circuito implementado na FPGA1 encontra-se apresentado na página seguinte.

Colocar aqui o esquema da FPGA1

E.2.3 Descrição VHDL dos blocos implementados

E.2.3.1 Ficheiro "`fpga1.vhd`"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "fpga1.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito logico implementado  *
--           pelo coprocessador, efectuando a ligacao de todos os modulos que  *
--           o constituem.                                                     *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on


entity fpga1 is
    port(Y,Cr,Cb: in STD_LOGIC_VECTOR(7 downto 0);
         Active,Valid,nCor,FPGA1_enable: in STD_LOGIC;
         AddressSRAM: out STD_LOGIC_VECTOR(2 downto 0);
         DataSRAM: out STD_LOGIC_VECTOR(7 downto 0);
         AddressEPROM: out STD_LOGIC_VECTOR(15 downto 3);
         Fim1,nWE_SRAM,nOE_SRAM,nOE_EPROM:out STD_LOGIC;

         Dbg_Estado_init,Dbg_EstadoA: out STD_LOGIC;
         Dbg_EstadoB,Dbg_EstadoC,Dbg_EstadoD:out STD_LOGIC;
         Dbg_S0,Dbg_S1,Dbg_S2: out STD_LOGIC;
         Dbg_Counter_CE: out STD_LOGIC;
         Dbg_Counter_CLK: out STD_LOGIC;
         -- readback
         rdbk_trig : in bit;
         rdbk_data : out bit);
end fpga1;

architecture arch1 of fpga1 is
    component Counter
        port(CLK,RST,LOAD: in STD_LOGIC;
             DataIn: in STD_LOGIC_VECTOR(15 downto 0);
             Saida: buffer STD_LOGIC_VECTOR(15 downto 0);
             Modulo: in STD_LOGIC_VECTOR(15 downto 0);
             EOC: out STD_LOGIC);
    end component;

    component Latch8
        port(portoD:in  STD_LOGIC_VECTOR(7 downto 0);
             portoQ:buffer STD_LOGIC_VECTOR(7 downto 0);
             Load,CLR:in STD_LOGIC);
    end component;

    component Maq1
        port(clk,init:in  STD_LOGIC;
             FF0,FF1,FF2,FF3,FF4:buffer  STD_LOGIC);
    end component;

    component MUX4_1B8
        port(in0,in1,in2,in3:in  STD_LOGIC_VECTOR(7 downto 0);
             saida:out  STD_LOGIC_VECTOR(7 downto 0);
             sel:in  STD_LOGIC_VECTOR(1 downto 0));
    end component;

    component Samostra
        port(Active,init,nD:in  STD_LOGIC;
             s0,s1,s2:buffer  STD_LOGIC);
    end component;

    component Xor2B13
        port(entrada: in STD_LOGIC_VECTOR(12 downto 0);
             neg: in STD_LOGIC;
```

```
            saida: out STD_LOGIC_VECTOR(12 downto 0));
    end component;

    component RDBK
        port(trig: in bit;
             data, rip: out bit);
    end component;

    for all: Counter use entity work.Counter(arch1);
    for all: Latch8 use entity work.Latch8(arch1);
    for all: Maq1 use entity work.Maq1(arch1);
    for all: MUX4_1B8 use entity work.MUX4_1B8(arch1);
    for all: Samostra use entity work.Samostra(arch1);
    for all: Xor2B13 use entity work.Xor2B13(arch1);

    signal estado_init: STD_LOGIC;
    signal estadoA: STD_LOGIC;
    signal estadoB: STD_LOGIC;
    signal estadoC: STD_LOGIC;
    signal estadoD,nestadoD: STD_LOGIC;
    signal Maq1_init: STD_LOGIC;
    signal nFPGA1_enable: STD_LOGIC;

    signal mux1out: STD_LOGIC_VECTOR(7 downto 0);
    signal mux1sel: STD_LOGIC_VECTOR(1 downto 0);
    signal s2:STD_LOGIC;
    signal latch1out: STD_LOGIC_VECTOR(7 downto 0);

    signal ce,sfim1,CounterCLK,Cor:STD_LOGIC;
    signal CounterOut:STD_LOGIC_VECTOR(15 downto 0) ;
    signal modulo: STD_LOGIC_VECTOR(15 downto 0);

    signal zeros: STD_LOGIC_VECTOR(15 downto 0);
    signal ones: STD_LOGIC_VECTOR(15 downto 0);

    signal zero: STD_LOGIC;
    signal one: STD_LOGIC;

    signal rdbk_rip: bit;

begin

    xrdbk: RDBK port map (rdbk_trig,rdbk_data,rdbk_rip);

    zeros <="0000000000000000";
    ones <="1111111111111111";

    zero <='0';
    one <='1';

    nFPGA1_enable<= not FPGA1_enable;
    nestadoD <= not estadoD;
    Cor<= not nCor;

    mux1:MUX4_1B8 port map(Cr,Cb,Y,Y,mux1out,mux1sel);
    latch1:Latch8 port map(mux1out,latch1out,estadoC,zero);

    process(FPGA1_enable,latch1out)
    begin
        if FPGA1_enable='1' then
            DataSRAM <= latch1out;
        else
            DataSRAM <= "ZZZZZZZZ";
        end if;
    end process;

    Maq1_init  <= (FPGA1_enable nand Active);
    maq:Maq1 port map (Valid, Maq1_init, estado_init, estadoA, estadoB, estadoC,
estadoD);
    samostra1:Samostra port map(Active, nFPGA1_enable, nestadoD, mux1sel(0),
mux1sel(1), s2);

    process(nCor)
    begin
        if nCor='0' then                  -- policromatico
            modulo<="1001010001111111";   --38015=(38016-1)
        else
            modulo<="0110001011111111";   --25343=(25344-1)
        end if;
```

```
    end process;

    ce<= (mux1sel(1) or (s2 and Cor));
    CounterCLK <= (ce nand estadoD);
    counter1:Counter port map(CounterCLK, nFPGA1_enable, zero, zeros, CounterOut,
modulo, sfim1);
    fim1 <= sfim1;

    selcor:Xor2B13 port map (CounterOut(15 downto 3), Cor, AddressEPROM(15 downto 3));

    process(FPGA1_enable,CounterOut)
    begin
        if FPGA1_enable='1' then
            AddressSRAM(2 downto 0) <=CounterOut(2 downto 0) ;
        else
            AddressSRAM(2 downto 0) <= "ZZZ";
        end if;
    end process;

    nOE_EPROM <= nFPGA1_enable;
    nOE_SRAM  <= FPGA1_enable;
    nWE_SRAM  <= not (FPGA1_enable and estadoD and ce and (not sfim1));

    Dbg_Estado_init <= Estado_init;
    Dbg_EstadoA <= estadoA;
    Dbg_EstadoB <= estadoB;
    Dbg_EstadoC <= estadoC;
    Dbg_EstadoD <= estadoD;
    Dbg_S0 <= mux1sel(0);
    Dbg_S1 <= mux1sel(1);
    Dbg_S2 <= s2;
    Dbg_Counter_CE <= ce;
    Dbg_Counter_CLK <= CounterCLK;
end arch1;
```

E.2.3.2 Ficheiro "`Maq1.vhd`"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "Maq1.vhd"                                                        *
--DESCRIÇÃO: Este ficheiro contem a descricao da maquina de estados Maq1.     *
--*****************************************************************************

library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on


entity Maq1 is
    port(clk,init:in  STD_LOGIC;
         FF0,FF1,FF2,FF3,FF4:buffer  STD_LOGIC);
end Maq1;

architecture arch1 of Maq1 is

        component FFDs
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
        end component;

        component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
        end component;

        for all: FFDr use entity work.FFDr(arch1);
```

```
        for all: FFDs use entity work.FFDs(arch1);

        signal zero: STD_LOGIC;
        signal inFF1: STD_LOGIC;

begin
        zero <= '0';

        inFF1 <= (FF4 or FF0);
        flip0:FFDs port map (zero,FF0,clk,init);
        flip1:FFDr port map (inFF1,FF1,clk,init);
        flip2:FFDr port map (FF1,FF2,clk,init);
        flip3:FFDr port map (FF2,FF3,clk,init);
        flip4:FFDr port map (FF3,FF4,clk,init);

end arch1;
```

E.2.3.3 Ficheiro "Samostra.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "Samostra.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao da maquina de estados Samostra   *
--*****************************************************************************

library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Samostra is
    port(Active,init,nD:in  STD_LOGIC;
         s0,s1,s2:buffer  STD_LOGIC);
end Samostra;

architecture arch1 of Samostra is
    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    for all: FFDr use entity work.FFDr(arch1);

    signal ns0,ns1,ns2:STD_LOGIC ;

begin
    ns0<= not s0;
    ns1<= not s1;
    ns2<= not s2;

    ff5:FFDr port map (ns0,s0,nD,init);
    ff6:FFDr port map (ns1,s1,Active,init);
    ff7:FFDr port map (ns2,s2,s1,init);
end arch1;
```

E.2.3.4 Ficheiro "Mux4_1B8.vhd"

```
--*********************************************************************************
--                          TRABALHO FINAL DE CURSO                                *
--*********************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*********************************************************************************
--FICHEIRO: "Mux4_1B8.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um multiplexer 4:1 de 8 bits. *
--*********************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity MUX4_1B8 is
    port(in0,in1,in2,in3:in  STD_LOGIC_VECTOR(7 downto 0);
         saida:out  STD_LOGIC_VECTOR(7 downto 0);
         sel:in  STD_LOGIC_VECTOR(1 downto 0));
end MUX4_1B8;

architecture arch1 of MUX4_1B8 is
begin
    process(in0,in1,in2,in3,sel)
    begin
        if sel="00" then
            saida<= in0;
        elsif sel="01" then
            saida<= in1;
        elsif sel="10" then
            saida<= in2;
        elsif sel="11" then
            saida<= in3;
        end if;
    end process;
end arch1;
```

E.2.3.5 Ficheiro "Counter.vhd"

```
--*********************************************************************************
--                          TRABALHO FINAL DE CURSO                                *
--*********************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*********************************************************************************
--FICHEIRO: "Counter.vhd"                                                        *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 16 bits com     *
--           RESET assincrono, carregamento sincrono e com sinalizacao de fim   *
--           de contagem.                                                       *
--*********************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter is
    port(CLK,RST,LOAD: in STD_LOGIC;
         DataIn: in STD_LOGIC_VECTOR(15 downto 0);
         Saida: buffer STD_LOGIC_VECTOR(15 downto 0);
         Modulo: in STD_LOGIC_VECTOR(15 downto 0);
         EOC: out STD_LOGIC);
end Counter;

architecture arch1 of Counter is
    component adder16
        port(in1,in2: in STD_LOGIC_VECTOR(15 downto 0);
```

```
            c0:in STD_LOGIC;
            s: out STD_LOGIC_VECTOR(15 downto 0);
            flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder16 use entity work.adder16(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(15 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(15 downto 0);
    signal ONES: STD_LOGIC_VECTOR(15 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES <= "1111111111111111";
    ZEROS <= "0000000000000000";

    Adder: adder16 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,RST,LOAD,Modulo,DataIn,saida,ZEROS)
    begin
        if RST='1' then
            saida<= ZEROS;
            EOC<='0';
        elsif CLK'event and CLK='1' then
            if LOAD='1' then
                saida<= DataIn;
            elsif saida=ONES then
                saida<=ZEROS;
            elsif saida=Modulo then
                EOC<='1';
            else
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

E.2.3.6 Ficheiro "Latch8.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Latch8.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop de 8 bits.      *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Latch8 is
    port(portoD:in  STD_LOGIC_VECTOR(7 downto 0);
         portoQ:buffer STD_LOGIC_VECTOR(7 downto 0);
         Load,CLR:in STD_LOGIC);
end Latch8;

architecture arch1 of Latch8 is
begin
    process(portoD,Load,CLR)
    begin
        if CLR='1' then
            portoQ<="00000000";
        elsif Load'event and Load='1' then
            portoQ<= portoD;
        end if;
    end process;
end arch1;
```

E.2.3.7 Ficheiro "Xor2B13.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "Xor2B13.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao do modulo que executa a funcao   *
--           lógica XOR dos 13 bits em 'entrada' com o bit 'neg'.               *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Xor2B13 is
    port(entrada: in STD_LOGIC_VECTOR(12 downto 0);
         neg: in STD_LOGIC;
         saida: out STD_LOGIC_VECTOR(12 downto 0));
end Xor2B13;

architecture arch1 of Xor2B13 is
begin
    process(entrada,neg)
    begin
        for I in 0 to 12 loop
            saida(I)<=entrada(I) xor neg;
        end loop;
    end process;
end arch1;
```

E.2.3.8 Ficheiro "adder16.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "adder16.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 16 bits.                                                        *
--*******************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder16 is
    port(in1,in2: in STD_LOGIC_VECTOR(15 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(15 downto 0);
         flag_carry: out STD_LOGIC);
end adder16;

architecture arch1 of adder16 is

    signal p,g: STD_LOGIC_VECTOR(15 downto 0);
    signal c: STD_LOGIC_VECTOR(16 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(15 downto 0);
```

```
    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    c(9) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)
and p(8)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8))
or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(5) and p(6) and p(7) and p(8)) or
        (g(6) and p(7) and p(8)) or
        (g(7) and p(8)) or
        g(8));

    c(10) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9))
or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(6) and p(7) and p(8) and p(9)) or
```

```
        (g(7) and p(8) and p(9)) or
        (g(8) and p(9)) or
        g(9));

    c(11) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10))
or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(7) and p(8) and p(9) and p(10)) or
        (g(8) and p(9) and p(10)) or
        (g(9) and p(10)) or
        g(10));

    c(12) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(8) and p(9) and p(10) and p(11)) or
        (g(9) and p(10) and p(11)) or
        (g(10) and p(11)) or
        g(11));

    c(13) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(9) and p(10) and p(11) and p(12)) or
        (g(10) and p(11) and p(12)) or
        (g(11) and p(12)) or
        g(12));

    c(14) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(9) and p(10) and p(11) and p(12) and p(13)) or
```

```
        (g(10) and p(11) and p(12) and p(13)) or
        (g(11) and p(12) and p(13)) or
        (g(12) and p(13)) or
        g(13));

    c(15) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) ) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13) and p(14)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13) and p(14)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(11) and p(12) and p(13) and p(14)) or
        (g(12) and p(13) and p(14)) or
        (g(13) and p(14)) or
        g(14));

    c(16) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and
p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14) and p(15)) or
          (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14) and p(15)) or
          (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14) and p(15)) or
          (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and
p(15)) or
          (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15))
or
          (g(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(12) and p(13) and p(14) and p(15)) or
          (g(13) and p(14) and p(15)) or
          (g(14) and p(15)) or
          g(15));

    flag_carry <= c(16);
end arch1;
```

E.2.3.9 Ficheiro "FFDs.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "FFDs.vhd"                                                        *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop tipo D com SET   *
--           assincrono.                                                      *
--*****************************************************************************

--use work.all;
library xc4000;
```

```
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity FFDs is
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,set:in STD_LOGIC);
end FFDs;

architecture arch1 of FFDs is
begin
    process(D,clk,set)
    begin
        if set='1' then
            Q<='1';
        elsif clk'event and clk='1' then
            Q<= D;
        end if;
    end process;
end arch1;
```

E.2.3.10 Ficheiro "FFDr.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "FFDr.vhd"                                                        *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop tipo D com RESET *
--           assincrono.                                                      *
--*****************************************************************************

--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity FFDr is
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,reset:in STD_LOGIC);
end FFDr;

architecture arch1 of FFDr is
begin
    process(D,clk,reset)
    begin
        if reset='1' then
            Q<='0';
        elsif clk'event and clk='1' then
            Q<= D;
        end if;
    end process;
end arch1;
```

E.3 DESCRIÇÃO DO CIRCUITO IMPLEMENTADO NA FPGA2

E.3.1 Descrição dos portos de entrada e saída

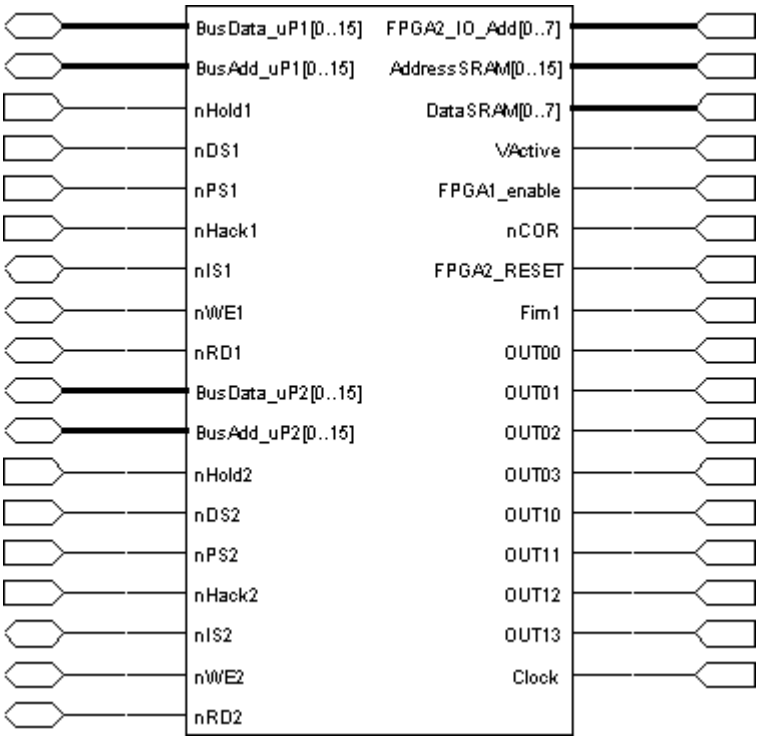


Figura E.2 - Descrição dos portos de entrada e saída da FPGA2.

PORTO	**DESCRIÇÃO**
FPGA2_IO_Addr	Define o byte mais significativo dos endereços de acesso aos registos e daqueles a que correspondem uma saída do descodificador de endereços
Clock	Relógio de 20MHz que comanda o funcionamento desta FPGA
BusData_uP1/2	Barramento de dados do DSP1/2
BusAdd_uP1/2	Barramento de endereços do DSP1/2
nHold1/2 nHoldA1/2	Sinais $\overline{\text{HOLD}}$ e $\overline{\text{HOLDA}}$ do DSP1/2
nIS1/2 nWE1/2 nRD1/2	Sinais $\overline{\text{IS}}$, $\overline{\text{WE}}$ e $\overline{\text{RD}}$ do DSP1/2
VActive	Sinal proveniente do conversor A/D que identifica o início de uma nova imagem
FPGA1_Enable	Sinal que comanda a FPGA1, para efectuar uma transferência de dados
nCor	Indica o modo de funcionamento: Policromático (‘0’) ou Monocromático (‘1’)
FPGA2_RESET	Sinal que provoca a inicialização da FPGA2
Fim1	Sinal proveniente da FPGA1, indicando o final da transferência de dados
AddressSRAM	Barramento de endereços para a SRAM do co-processador
DataSRAM	Barramento de dados proveniente da SRAM do co-processador
OUTij	Sinais resultantes da descodificação de endereços

E.3.2 Esquema Lógico

O esquema geral do circuito implementado na FPGA2 encontra-se apresentado na página seguinte.

Colocar aqui o esquema da FPGA2

E.3.3 Codificação dos estados da máquina `Maq2`

ESTADOS / SINAIS	E0 0x001	E1 0x002	E2 0x004	E3 0x008	E4 0x010	E5 0x020	E6 0x040	E7 0x080	E8 0x100	E9 0x200
FPGA1_Enable	0	0	0	0	0	0	0	0	0	1
Bdata_uP1in	0	0	1	0	0	0	0	0	0	0
Bdata_uP1out	0	0	0	1	0	0	0	0	0	0
Bdata_uP2in	0	0	0	0	0	0	1	0	0	0
Bdata_uP2out	0	0	0	0	0	0	0	1	0	0
LE_Data	0	0	1	0	0	0	1	0	0	0
Reset_Counter1	1	1	0	0	0	0	0	0	1	1
CLK_Counter1	1	1	1	0	1	1	1	0	1	1
Sel_uP	0	0	0	0	1	1	1	1	1	0
Load_Counter2	1	1	0	0	1	1	0	0	1	1
CLK_Counter2	0	1	1	0	0	1	1	0	1	0
Baddress_uP1low	0	0	1	1	0	0	0	0	0	0
Baddress_uP2low	0	0	0	0	0	0	1	1	0	0
Baddress_SRAM	1	1	1	1	1	1	1	1	1	0
nWE1	X	X	1	0	X	X	X	X	X	X
nRD1	X	X	0	1	X	X	X	X	X	X
nPS1	X	X	1	1	X	X	X	X	X	X
nDS1	X	X	0	0	X	X	X	X	X	X
nIS1	X	X	1	1	X	X	X	X	X	X
nWE2	X	X	X	X	X	X	1	0	X	X
nRD2	X	X	X	X	X	X	0	1	X	X
nPS2	X	X	X	X	X	X	1	1	X	X
nDS2	X	X	X	X	X	X	0	0	X	X
nIS2	X	X	X	X	X	X	1	1	X	X
nHold1	0	1	1	1	0	0	0	0	0	0
nHold2	0	0	0	0	0	1	1	1	0	0

E.3.4 Descrição VHDL dos blocos implementados

E.3.4.1 Ficheiro "`fpga2.vhd`"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "fpga2.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito implementado na FPGA2*
--           pelo que contem a interligacao de todos os modulos que constituem *
--           esse mesmo circuito.                                             *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity fpga2 is
    port(
        FPGA2_IO_Add:in STD_LOGIC_VECTOR(7 downto 0);
        Clock:in STD_LOGIC;

        BusData_uP1: inout STD_LOGIC_VECTOR(15 downto 0);
        BusAdd_uP1: inout STD_LOGIC_VECTOR(15 downto 0);
        nHold1,nDS1,nPS1:out STD_LOGIC;
        nHack1:in STD_LOGIC;
        nIS1,nWE1,nRD1:inout STD_LOGIC;

        BusData_uP2: inout STD_LOGIC_VECTOR(15 downto 0);
        BusAdd_uP2: inout STD_LOGIC_VECTOR(15 downto 0);
        nHold2,nDS2,nPS2:out STD_LOGIC;
        nHack2:in STD_LOGIC;
        nIS2,nWE2,nRD2:inout STD_LOGIC;

        VActive: in STD_LOGIC;
        FPGA1_enable: out STD_LOGIC;
        nCOR: out STD_LOGIC;
        FPGA2_RESET: in STD_LOGIC;
        Fim1: in STD_LOGIC;

        AddressSRAM: out STD_LOGIC_VECTOR(15 downto 0);
        DataSRAM: in STD_LOGIC_VECTOR(7 downto 0);

        OUT00,OUT01,OUT02,OUT03: out STD_LOGIC;
        OUT10,OUT11,OUT12,OUT13: out STD_LOGIC;

        -- readback
        rdbk_trig : in bit;
        rdbk_data : out bit);
end fpga2;

architecture arch1 of fpga2 is
    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    component Address_Gen
        port(Start_Address1,Start_Address2:in  STD_LOGIC_VECTOR(7 downto 0);
             OffsetRW1,OffsetRW2:in  STD_LOGIC_VECTOR(7 downto 0);
             Lim1,Lim2:in  STD_LOGIC_VECTOR(15 downto 0);
             Counter2_RST:in  STD_LOGIC;
             Estados: in STD_LOGIC_VECTOR(9 downto 0);
             Fim2:out STD_LOGIC;
             Address_uP1,Address_uP2:out STD_LOGIC_VECTOR(15 downto 0);
             Gen1, Gen2:out STD_LOGIC);
    end component;
```

```
    component Data_Mgmt
        port(DataOut_SRAM: in STD_LOGIC_VECTOR(7 downto 0);
             Address_SRAM: out STD_LOGIC_VECTOR(15 downto 0);
             Data_uP1,Data_uP2: inout STD_LOGIC_VECTOR(15 downto 0);
             Estados:in STD_LOGIC_VECTOR(9 downto 0);
             Data_uP1In,Data_uP2In,Data_uP1Out,Data_uP2Out: out STD_LOGIC);
    end component;

    component Contr_Regs
        port(DataBus:inout  STD_LOGIC_VECTOR(15 downto 0);
             AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
             FPGA2_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
             nISx,nWE,nRD,CLR0,CLR1:in STD_LOGIC;
             TargetAdd,OffsetAdd:out  STD_LOGIC_VECTOR(7 downto 0);
             Lim:out  STD_LOGIC_VECTOR(15 downto 0);
             DReq:out STD_LOGIC;
             ReadReg:out STD_LOGIC);
    end component;

    component Maq2
        port(clk,init:in  STD_LOGIC;
             DReq1,DReq2,nHoldAuP1,nHoldAuP2:in  STD_LOGIC;
             Fim1,Fim2,VActive:in  STD_LOGIC;
             FF0,FF1,FF2,FF3,FF4:buffer  STD_LOGIC;
             FF5,FF6,FF7,FF8,FF9:buffer  STD_LOGIC);
    end component;

    component Ext_Add_Dec
        port(AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
             FPGA2_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
             nISx:in STD_LOGIC;
             OUT0,OUT1,OUT2,OUT3: out STD_LOGIC);
    end component;

    component BufTBid16
        port(T12,T21:in STD_LOGIC;
             entrada_saida1,entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
    end component;

    component BufTBid16v2
        port(T12,T21:in STD_LOGIC;
             entrada1:in STD_LOGIC_VECTOR(15 downto 0);
             saida1:out STD_LOGIC_VECTOR(15 downto 0);
             entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
    end component;

    component RDBK
        port(trig: in bit;
             data, rip: out bit);
    end component;

    for all: FFDr use entity work.FFDr(arch1);

    for all: Address_Gen use entity work.Address_Gen(arch1);
    for all: Data_Mgmt use entity work.Data_Mgmt(arch1);
    for all: Contr_Regs use entity work.Contr_Regs(arch1);
    for all: Maq2 use entity work.Maq2(arch1);
    for all: Ext_Add_Dec use entity work.Ext_Add_Dec(arch1);
    for all: BufTBid16 use entity work.BufTBid16(arch1);
    for all: BufTBid16v2 use entity work.BufTBid16v2(arch1);

    signal SoftRESET, RESET: STD_LOGIC;
    signal EQU23, TCor1, TCor2, CLKCor: STD_LOGIC;
    signal ReadReg1, ReadReg2: STD_LOGIC;
    signal Data_uP1In,Data_uP2In,Data_uP1Out,Data_uP2Out: STD_LOGIC;
    signal TD12_1,TD21_1,TD12_2,TD21_2: STD_LOGIC;
    signal TA12_1,TA21_1,TA12_2,TA21_2: STD_LOGIC;
    signal Gen1, Gen2: STD_LOGIC;

    signal xnCor: STD_LOGIC_VECTOR (15 downto 0);
    signal xnCor1: STD_LOGIC;
    signal nTCor1: STD_LOGIC;
    signal Estados: STD_LOGIC_VECTOR (9 downto 0);
    signal Fim2: STD_LOGIC;

    signal xBusData_uP1, BusAddIn_uP1, BusAddOut_uP1: STD_LOGIC_VECTOR (15 downto 0);
    signal TargetAdd1,OffsetAdd1: STD_LOGIC_VECTOR (7 downto 0);
    signal Lim1: STD_LOGIC_VECTOR (15 downto 0);
    signal DReq1,CLR1: STD_LOGIC;
```

```
    signal Hold1x: STD_LOGIC;
    signal IS1: STD_LOGIC;

    signal xBusData_uP2, BusAddIn_uP2, BusAddOut_uP2: STD_LOGIC_VECTOR (15 downto 0);
    signal TargetAdd2,OffsetAdd2: STD_LOGIC_VECTOR (7 downto 0);
    signal Lim2: STD_LOGIC_VECTOR (15 downto 0);
    signal DReq2,CLR2: STD_LOGIC;
    signal Hold2x: STD_LOGIC;
    signal IS2: STD_LOGIC;

    signal rdbk_rip: bit;

begin

    xrdbk: RDBK port map (rdbk_trig,rdbk_data,rdbk_rip);

    process(FPGA2_IO_Add,BusAddIn_uP1,nIS1,BusAddIn_uP2,nIS2)
    begin
        if (((BusAddIn_uP1(15 downto 8)=FPGA2_IO_Add) and (BusAddIn_uP1(7 downto
1)="0101001") and nIS1='0') or ((BusAddIn_uP2(15 downto 8)=FPGA2_IO_Add) and
(BusAddIn_uP2(7 downto 1)="0101001") and nIS2='0')) then
            EQU23<='1';
        else
            EQU23<='0';
        end if;
    end process;

    CLKCor <= not ((not nIS1) and (not(nWE1)) and EQU23 and (not(BusAddIn_uP1(0))));

    FFCor: FFDr port map(xnCor1,xnCor(0),CLKCor,RESET);

    nCor <= xnCor(0);

    TCor1 <= (not nIS1) and EQU23 and (not(nRD1)) and (not(BusAddIn_uP1(0)));
    TCor2 <= (not nIS2) and EQU23 and (not(nRD2)) and (not(BusAddIn_uP2(0)));

    process(TCor1,xnCor)
    begin
        if TCor1='0' then
            xBusData_uP1<= "ZZZZZZZZZZZZZZZZ";
        else
            xBusData_uP1<= xnCor;
        end if;
    end process;

    nTCor1 <= not TCor1;
    xnCor(15 downto 1)<="000000000000000";

    process(nTCor1,xBusData_uP1)
    begin
        if nTCor1='0' then
            xnCor1 <= 'Z';
        else
            xnCor1 <= xBusData_uP1(0);
        end if;
    end process;

    process(TCor2,xnCor)
    begin
        if TCor2='0' then
            xBusData_uP2<= "ZZZZZZZZZZZZZZZZ";
        else
            xBusData_uP2 <= xnCor;
        end if;
    end process;

    SoftRESET <= (not nIS1) and (not(nWE1)) and EQU23 and BusAddIn_uP1(0);
    RESET<=(SoftReset or FPGA2_RESET);

    Maq2_1: Maq2 port map(Clock, RESET, DReq1, DReq2, nHack1, nHack2, Fim1, Fim2,
VActive,
                          Estados(0),Estados(1),Estados(2),Estados(3),Estados(4),
                          Estados(5),Estados(6),Estados(7),Estados(8),Estados(9));

    CLR1<= (RESET or (estados(3)and Fim2));
    CLR2<= (RESET or (estados(7)and Fim2));

    Contr_Regs_1: Contr_Regs port map(xBusData_uP1, BusAddIn_uP1, FPGA2_IO_Add, nIS1,
nWE1, nRD1, CLR1, RESET, TargetAdd1, OffsetAdd1, Lim1, DReq1, ReadReg1);
```

```
    Contr_Regs_2: Contr_Regs port map(xBusData_uP2, BusAddIn_uP2, FPGA2_IO_Add, nIS2,
nWE2, nRD2, CLR2, RESET, TargetAdd2, OffsetAdd2, Lim2, DReq2, ReadReg2);

    Address_Gen_1: Address_Gen port map(TargetAdd1, TargetAdd2, OffsetAdd1, OffsetAdd2,
Lim1, Lim2, RESET, Estados, Fim2, BusAddOut_uP1, BusAddOut_uP2, Gen1, Gen2);

    Data_Mgmt_1: Data_Mgmt port map(DataSRAM, AddressSRAM, xBusData_uP1, xBusData_uP2,
Estados, Data_uP1In, Data_uP2In, Data_uP1Out, Data_uP2Out);

    TD12_1 <= Data_uP1Out or ReadReg1 or TCor1;
    TD21_1 <= Data_uP1In or (not ReadReg1) or (not TCor1);
    BufData1: BufTBid16 port map (TD12_1, TD21_1, xBusData_uP1, BusData_uP1);

    TD12_2 <= Data_uP2Out or ReadReg2 or TCor2;
    TD21_2 <= Data_uP2In or (not ReadReg2) or (not TCor2);
    BufData2: BufTBid16 port map (TD12_2, TD21_2, xBusData_uP2, BusData_uP2);

    TA12_1 <= Gen1;
    TA21_1 <= not Gen1;
    BufAdd1: BufTBid16v2 port map (TA12_1, TA21_1, BusAddOut_uP1, BusAddIn_uP1,
BusAdd_uP1);

    TA12_2 <= Gen2;
    TA21_2 <= not Gen2;
    BufAdd2: BufTBid16v2 port map (TA12_2, TA21_2, BusAddOut_uP2, BusAddIn_uP2,
BusAdd_uP2);

    process(Estados)
    begin
        if (Estados(2) or Estados(3))='1' then
            nDS1<='0';
            nPS1<='1';
            nIS1<='1';
            nWE1<= not(Estados(3));
            nRD1<= not(Estados(2));
        else
            nDS1<='Z';
            nPS1<='Z';
            nIS1<='Z';
            nWE1<='Z';
            nRD1<='Z';
        end if;
    end process;

    process(Estados)
    begin
        if (Estados(6) or Estados(7))='1' then
            nDS2<='0';
            nPS2<='1';
            nIS2<='1';
            nWE2<= not(Estados(7));
            nRD2<= not(Estados(6));
        else
            nDS2<='Z';
            nPS2<='Z';
            nIS2<='Z';
            nWE2<='Z';
            nRD2<='Z';
        end if;
    end process;

    Hold1x<=(Estados(1) or Estados(2) or Estados(3));
    Hold2x<=(Estados(5) or Estados(6) or Estados(7));

    process(Hold1x,Hold2x)
    begin
        if (Hold1x='1') then
            nHold1<='0';
        else
            nHold1<='Z';
        end if;

        if (Hold2x='1') then
            nHold2<='0';
        else
            nHold2<='Z';
        end if;
    end process;
```

```
    FPGA1_ENABLE <= Estados(9);

    --- DESCODIFICACAO DE ENDERECOS AUXILIARES
    --- FPGA_IO_Add & 0000 01xx
    uP1_Ext_Regs : Ext_Add_Dec port map (BusAddIn_uP1, FPGA2_IO_Add, nIS1, OUT00,
OUT01, OUT02, OUT03);
    uP2_Ext_Regs : Ext_Add_Dec port map (BusAddIn_uP2, FPGA2_IO_Add, nIS2, OUT10,
OUT11, OUT12, OUT13);

end arch1;
```

E.3.4.2 Ficheiro "Contr_Regs.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Contr_Regs.vhd"                                                  *
--DESCRIÇÃO: Este ficheiro contem a descricao de um banco de registos do      *
--           coprocessador.                                                   *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Contr_Regs is
    port(DataBus:inout  STD_LOGIC_VECTOR(15 downto 0);
         AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
         FPGA2_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
         nISx,nWE,nRD,CLR0,CLR1:in STD_LOGIC;
         TargetAdd,OffsetAdd:out  STD_LOGIC_VECTOR(7 downto 0);
         Lim:out  STD_LOGIC_VECTOR(15 downto 0);
         DReq:out STD_LOGIC;
         ReadReg:out STD_LOGIC);
end Contr_Regs;

architecture arch1 of Contr_Regs is
    component Flip_Flop16
        port(portoD:in  STD_LOGIC_VECTOR(15 downto 0);
             portoQ:buffer STD_LOGIC_VECTOR(15 downto 0);
             CLK,CLR:in STD_LOGIC);
    end component;

    for all:Flip_Flop16 use entity work.Flip_Flop16(arch1);

    signal FFout0,FFout1:STD_LOGIC_VECTOR(15 downto 0);
    signal CLK0,CLK1:STD_LOGIC;
    signal T0,T1,EQU:STD_LOGIC;

begin
    process(FPGA2_IO_Add,AddressBus,nISx)
    begin
        if ((AddressBus(15 downto 8)=FPGA2_IO_Add) and nISx='0' and (AddressBus(7
downto 1)="0101000")) then
            EQU<='1';
        else
            EQU<='0';
        end if;
    end process;

    CLK0 <= (not nISx) and (not(nWE)) and EQU and (not(AddressBus(0)));
    CLK1 <= (not nISx) and (not(nWE)) and EQU and AddressBus(0);

    FF0: Flip_Flop16 port map(DataBus,FFout0,CLK0,CLR0);
    FF1: Flip_Flop16 port map(DataBus,FFout1,CLK1,CLR1);

    DReq <= FFout0(0) or FFout0(1) or FFout0(2) or FFout0(3)
            or FFout0(4) or FFout0(5) or FFout0(6) or FFout0(7)
```

```
            or FFout0(8) or FFout0(9) or FFout0(10) or FFout0(11)
            or FFout0(12) or FFout0(13) or FFout0(14) or FFout0(15);

    T0 <= (not nISx) and EQU and (not(nRD)) and (not(AddressBus(0)));
    T1 <= (not nISx) and EQU and (not(nRD)) and AddressBus(0);

    process(T0,FFout0)
    begin
        if T0='0' then
            DataBus <= "ZZZZZZZZZZZZZZZZ";
        else
            DataBus <= FFout0;
        end if;
    end process;

    process(T1,FFout1)
    begin
        if T1='0' then
            DataBus <= "ZZZZZZZZZZZZZZZZ";
        else
            DataBus <= FFout1;
        end if;
    end process;

    TargetAdd <= FFout1(15 downto 8);
    OffsetAdd <= FFout1(7 downto 0);
    Lim <= FFout0(15 downto 0);
    ReadReg <= (T0 or T1);
end arch1;
```

E.3.4.3 Ficheiro "Maq2.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "Maq2.vhd"                                                        *
--DESCRIÇÃO: Este ficheiro contem a descricao da maquina de estados que tem,   *
--           como funcao principal, controlar a transferencia DMA dos dados    *
--           para os dois DSPs.                                                *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Maq2 is
    port(clk, init:in  STD_LOGIC;
         DReq1, DReq2, nHoldAuP1, nHoldAuP2: in  STD_LOGIC;
         Fim1, Fim2, VActive: in  STD_LOGIC;
         FF0, FF1, FF2, FF3, FF4: buffer STD_LOGIC;
         FF5, FF6, FF7, FF8, FF9: buffer STD_LOGIC);
end Maq2;

architecture arch1 of Maq2 is
    component FFDs
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,set:in STD_LOGIC);
    end component;

    component FFDr
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,reset:in STD_LOGIC);
    end component;

    for all: FFDr use entity work.FFDr(arch1);
    for all: FFDs use entity work.FFDs(arch1);
```

```
    signal zero: STD_LOGIC;
    signal inFF0,inFF1,inFF2,inFF3,inFF4: STD_LOGIC;
    signal inFF5,inFF6,inFF7,inFF8,inFF9: STD_LOGIC;

begin
    zero <= '0';

    inFF0 <= (Fim1 and FF9) or (not(DReq1) and FF0);
    inFF1 <= (DReq1 and FF0) or (nHoldAuP1 and FF1);
    inFF2 <= (not(nHoldAuP1) and FF1) or (not(Fim2) and FF3);
    inFF3 <=  FF2;
    inFF4 <= (Fim2 and FF3) or (not(DReq2) and FF4) or ((not nHoldAuP1) and FF4);
    inFF5 <= (DReq2 and FF4) or (nHoldAuP2 and FF5);
    inFF6 <= (not(nHoldAuP2) and FF5) or (not(Fim2) and FF7);
    inFF7 <=  FF6;
    inFF8 <= (Fim2 and FF7) or (VActive and FF8) or ((not nHoldAuP2) and FF8);
    inFF9 <= (not(Fim1) and FF9) or (not(VActive) and FF8);

    flip0:FFDr port map (inFF0,FF0,clk,init);
    flip1:FFDr port map (inFF1,FF1,clk,init);
    flip2:FFDr port map (inFF2,FF2,clk,init);
    flip3:FFDr port map (inFF3,FF3,clk,init);
    flip4:FFDr port map (inFF4,FF4,clk,init);
    flip5:FFDr port map (inFF5,FF5,clk,init);
    flip6:FFDr port map (inFF6,FF6,clk,init);
    flip7:FFDr port map (inFF7,FF7,clk,init);
    flip8:FFDs port map (inFF8,FF8,clk,init);
    flip9:FFDr port map (inFF9,FF9,clk,init);
end arch1;
```

E.3.4.4 Ficheiro "Address_Gen.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Address_Gen.vhd"                                                 *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito que faz parte do    *
--           controlador DMA implementado, gerando os enderecos de leitura e  *
--           escrita de/para as memorias do processador.                       *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Address_Gen is
    port(Start_Address1,Start_Address2:in  STD_LOGIC_VECTOR(7 downto 0);
         OffsetRW1,OffsetRW2:in  STD_LOGIC_VECTOR(7 downto 0);
         Lim1,Lim2:in  STD_LOGIC_VECTOR(15 downto 0);
         Counter2_RST:in  STD_LOGIC;
         Estados: in STD_LOGIC_VECTOR(9 downto 0);
         Fim2:out STD_LOGIC;
         Address_uP1,Address_uP2:out STD_LOGIC_VECTOR(15 downto 0);
         Gen1, Gen2:out STD_LOGIC);
end Address_Gen;

architecture arch1 of Address_Gen is
    component Counter
        port(CLK,RST,LOAD: in STD_LOGIC;
             DataIn: in STD_LOGIC_VECTOR(15 downto 0);
             Saida: buffer STD_LOGIC_VECTOR(15 downto 0);
             Modulo: in STD_LOGIC_VECTOR(15 downto 0);
             EOC: out STD_LOGIC);
    end component;

    component adder8
        port(in1,in2: in STD_LOGIC_VECTOR(7 downto 0);
             c0:in STD_LOGIC;
```

```
            s: out STD_LOGIC_VECTOR(7 downto 0);
            flag_carry: out STD_LOGIC);
    end component;

    component BufTB8
        port(DataIn: in STD_LOGIC_VECTOR(7 downto 0);
            DataOut: out STD_LOGIC_VECTOR(7 downto 0);
            T: in STD_LOGIC);
    end component;

    for all: Counter use entity work.Counter(arch1);
    for all: adder8 use entity work.adder8(arch1);
    for all: BufTB8 use entity work.BufTB8(arch1);

    signal Modulo: STD_LOGIC_VECTOR (15 downto 0);
    signal StartAdd: STD_LOGIC_VECTOR (15 downto 0);
    signal OffsetRW: STD_LOGIC_VECTOR (7 downto 0);

    signal Sel_uP: STD_LOGIC;
    signal Load_Counter2,CLK_Counter2: STD_LOGIC;
    signal Counter2_Out: STD_LOGIC_VECTOR (15 downto 0);

    signal Adder_Cin: STD_LOGIC;
    signal Adder_Cout: STD_LOGIC;
    signal Adder_Out: STD_LOGIC_VECTOR (7 downto 0);

    signal Baddress_uP1_Low,Baddress_uP2_Low: STD_LOGIC;
    signal Baddress_uP1_HighR,Baddress_uP1_HighW: STD_LOGIC;
    signal Baddress_uP2_HighR,Baddress_uP2_HighW: STD_LOGIC;

begin
    StartAdd(7 downto 0) <= "00000000";
    Adder_Cin<='0';
    Sel_uP<= (Estados(4) or Estados(5) or Estados(6) or Estados(7) or Estados(8));

    process(Lim1, Lim2, Sel_uP, Start_Address1, OffsetRW1, Start_Address2, OffsetRW2)
    begin
        if Sel_uP='0' then
            Modulo(15 downto 0) <= Lim1;
            StartAdd(15 downto 8) <= Start_Address1;
            OffsetRW <= OffsetRW1;
        else
            Modulo(15 downto 0) <= Lim2;
            StartAdd(15 downto 8) <= Start_Address2;
            OffsetRW <= OffsetRW2;
        end if;
    end process;

    Load_Counter2<= (Estados(0) or Estados(1) or Estados(4) or Estados(5) or Estados(8)
or Estados(9));
    CLK_Counter2<=  not(Estados(0) or Estados(3) or Estados(4) or Estados(7) or
Estados(9));

    Counter2: Counter port
map(CLK_Counter2,Counter2_RST,Load_Counter2,StartAdd,Counter2_Out,Modulo,Fim2);

    Adder1: adder8 port map(OffsetRW,Counter2_Out(15 downto
8),Adder_Cin,Adder_Out,Adder_Cout);

    Baddress_uP1_Low<= (Estados(2) or Estados(3));
    Baddress_uP2_Low<= (Estados(6) or Estados(7));
    Baddress_uP1_HighR<= Estados(2);
    Baddress_uP1_HighW<= Estados(3);
    Baddress_uP2_HighR<= Estados(6);
    Baddress_uP2_HighW<= Estados(7);

    Gen1<=Baddress_uP1_Low and (Baddress_uP1_HighR or Baddress_uP1_HighW);
    Gen2<=Baddress_uP2_Low and (Baddress_uP2_HighR or Baddress_uP2_HighW);

    BufT1: BufTB8 port map(Counter2_Out(7 downto 0), Address_uP1(7 downto 0),
Baddress_uP1_Low);
    BufT2: BufTB8 port map(Counter2_Out(15 downto 8), Address_uP1(15 downto 8),
Baddress_uP1_HighR);
    BufT3: BufTB8 port map(Adder_Out, Address_uP1(15 downto 8), Baddress_uP1_HighW);

    BufT4: BufTB8 port map(Counter2_Out(7 downto 0), Address_uP2(7 downto 0),
Baddress_uP2_Low);
    BufT5: BufTB8 port map(Counter2_Out(15 downto 8), Address_uP2(15 downto 8),
Baddress_uP2_HighR);
```

```
    BufT6: BufTB8 port map(Adder_Out, Address_uP2(15 downto 8), Baddress_uP2_HighW);

end arch1;
```

E.3.4.5 Ficheiro "Data_Mgmt.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Data_Mgmt.vhd"                                                   *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito que efectua a       *
--           subtraccao dos valores na memoria dos DSPs aos pixels            *
--           correspondentes na SRAM externa.                                 *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Data_Mgmt is
    port(
        DataOut_SRAM: in STD_LOGIC_VECTOR(7 downto 0);
        Address_SRAM: out STD_LOGIC_VECTOR(15 downto 0);
        Data_uP1,Data_uP2: inout STD_LOGIC_VECTOR(15 downto 0);
        Estados:in STD_LOGIC_VECTOR(9 downto 0);
        Data_uP1In,Data_uP2In,Data_uP1Out,Data_uP2Out: out STD_LOGIC);
end Data_Mgmt;

architecture arch1 of Data_Mgmt is
    component Counter
        port(CLK,RST,LOAD: in STD_LOGIC;
             DataIn: in STD_LOGIC_VECTOR(15 downto 0);
             Saida: buffer STD_LOGIC_VECTOR(15 downto 0);
             Modulo: in STD_LOGIC_VECTOR(15 downto 0);
             EOC: out STD_LOGIC);
    end component;

    component adder8
        port(in1,in2: in STD_LOGIC_VECTOR(7 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(7 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    component BufTB8
        port(DataIn: in STD_LOGIC_VECTOR(7 downto 0);
             DataOut: out STD_LOGIC_VECTOR(7 downto 0);
             T: in STD_LOGIC);
    end component;

    component BufTB16
        port(DataIn: in STD_LOGIC_VECTOR(15 downto 0);
             DataOut: out STD_LOGIC_VECTOR(15 downto 0);
             T: in STD_LOGIC);
    end component;

    component Latch9
        port(portoD:in  STD_LOGIC_VECTOR(8 downto 0);
             portoQ:buffer STD_LOGIC_VECTOR(8 downto 0);
             LE,CLR:in STD_LOGIC);
    end component;

    for all: Counter use entity work.Counter(arch1);
    for all: adder8 use entity work.adder8(arch1);
    for all: BufTB8 use entity work.BufTB8(arch1);
    for all: BufTB16 use entity work.BufTB16(arch1);
    for all: Latch9 use entity work.Latch9(arch1);
```

```
    signal Start_Address: STD_LOGIC_VECTOR (15 downto 0);
    signal Modulo: STD_LOGIC_VECTOR (15 downto 0);
    signal Counter1_Load,EOC: STD_LOGIC;
    signal Counter1_Out: STD_LOGIC_VECTOR (15 downto 0);
    signal CLK_Counter1,RST_Counter1: STD_LOGIC;

    signal Adder_Cin: STD_LOGIC;
    signal Adder_Cout: STD_LOGIC;
    signal Adder_Out: STD_LOGIC_VECTOR (8 downto 0);

    signal Data_In,nData_In: STD_LOGIC_VECTOR (7 downto 0);
    signal Data_Out: STD_LOGIC_VECTOR (15 downto 0);

    signal LE_Data: STD_LOGIC;
    signal Latch_CLR: STD_LOGIC;

    signal Baddress_SRAM: STD_LOGIC;

    signal BData_uP1In,BData_uP2In: STD_LOGIC;
    signal BData_uP1Out,BData_uP2Out: STD_LOGIC;

begin

    Modulo <= "1111111111111111";
    Start_Address <= "0000000000000000";
    Counter1_Load <='0';
    Adder_Cin<='1';
    Latch_CLR<='0';

    CLK_Counter1<= not(Estados(3) or Estados(7));
    RST_Counter1<= (Estados(0) or Estados(1) or Estados(8) or Estados(9));
    Counter1: Counter port map(CLK_Counter1, RST_Counter1, Counter1_Load,
Start_Address, Counter1_Out, Modulo, EOC);

    BufT1: BufTB16 port map(Counter1_Out, Address_SRAM, BAddress_SRAM);

    nData_In <= not Data_In;
    Adder1: adder8 port map(DataOut_SRAM, nData_In, Adder_Cin, Adder_Out(7 downto 0),
Adder_Cout);
    Adder_Out(8)<= not Adder_Cout;

    LE_Data<=(Estados(2) or Estados(6));
    Latch: Latch9 port map(Adder_Out,Data_Out(8 downto 0), LE_Data, Latch_CLR);
    Data_Out(9)<=Data_Out(8);
    Data_Out(10)<=Data_Out(8);
    Data_Out(11)<=Data_Out(8);
    Data_Out(12)<=Data_Out(8);
    Data_Out(13)<=Data_Out(8);
    Data_Out(14)<=Data_Out(8);
    Data_Out(15)<=Data_Out(8);

    Baddress_SRAM<= not(Estados(9));

    BData_uP1In<= Estados(2);
    BData_uP2In<= Estados(6);
    BData_uP1Out<= Estados(3);
    BData_uP2Out<= Estados(7);

    BufT2: BufTB16 port map(Data_Out,Data_uP1,BData_uP1Out);
    BufT3: BufTB16 port map(Data_Out,Data_uP2,BData_uP2Out);
    BufT4: BufTB8 port map(Data_uP1(7 downto 0),Data_In,BData_uP1In);
    BufT5: BufTB8 port map(Data_uP2(7 downto 0),Data_In,BData_uP2In);

    Data_uP1In <= BData_uP1In;
    Data_uP2In <= BData_uP2In;
    Data_uP1Out <= BData_uP1Out;
    Data_uP2Out <= BData_uP2Out;
end arch1;
```

E.3.4.6 Ficheiro "Ext_Add_Dec.vhd"

```
********************************************************************************
*                            TRABALHO FINAL DE CURSO                           *
********************************************************************************
* AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
*          Nuno Roma        - N. 39921                                         *
********************************************************************************
* FICHEIRO: "Ext_Add_Dec.vhd"                                                  *
* DESCRIÇÃO: Este ficheiro contem a descricao do descodificador de enderecos   *
*            auxiliares.                                                       *
********************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Ext_Add_Dec is
    port(AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
         FPGA2_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
         nISx:in STD_LOGIC;
         OUT0,OUT1,OUT2,OUT3: out STD_LOGIC);
end Ext_Add_Dec;

architecture arch1 of Ext_Add_Dec is

    signal EQU:STD_LOGIC;

begin
    process(FPGA2_IO_Add,AddressBus,nISx)
    begin
        if ((AddressBus(15 downto 8)=FPGA2_IO_Add) and nISx='0' and (AddressBus(7
downto 2)="010101")) then
            EQU<='1';
        else
            EQU<='0';
        end if;
    end process;

    OUT0 <= EQU and (not AddressBus(1)) and (not AddressBus(0));
    OUT1 <= EQU and (not AddressBus(1)) and (    AddressBus(0));
    OUT2 <= EQU and (    AddressBus(1)) and (not AddressBus(0));
    OUT3 <= EQU and (    AddressBus(1)) and (    AddressBus(0));
end arch1;
```

E.3.4.7 Ficheiro "Counter.vhd"

```
--********************************************************************************
--                            TRABALHO FINAL DE CURSO                           *
--********************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--********************************************************************************
--FICHEIRO: "Counter.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 16 bits com    *
--           sinalizacao de fim de contagem.                                   *
--********************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter is
    port(CLK,RST,LOAD: in STD_LOGIC;
         DataIn: in STD_LOGIC_VECTOR(15 downto 0);
```

```
         Saida: buffer STD_LOGIC_VECTOR(15 downto 0);
         Modulo: in STD_LOGIC_VECTOR(15 downto 0);
         EOC: out STD_LOGIC);
end Counter;

architecture arch1 of Counter is
    component adder16
        port(in1,in2: in STD_LOGIC_VECTOR(15 downto 0);
        c0:in STD_LOGIC;
        s: out STD_LOGIC_VECTOR(15 downto 0);
        flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder16 use entity work.adder16(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(15 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(15 downto 0);
    signal ONES: STD_LOGIC_VECTOR(15 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES <= "1111111111111111";
    ZEROS <= "0000000000000000";
    Adder: adder16 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,RST,LOAD,Modulo,DataIn,saida,saidaM1,ZEROS,ONES)
    begin
        if RST='1' then
            saida<= ZEROS;
            EOC<='0';
        elsif CLK'event and CLK='1' then
            if LOAD='1' then
                saida<= DataIn;
                EOC<='0';
            elsif saida=ONES then
                saida<=ZEROS;
            elsif saida=Modulo then
                EOC<='1';
            else
                saida<= saidaM1;
                if saidaM1=Modulo then
                    EOC<='1';
                end if;
            end if;
        end if;
    end process;
end arch1;
```

E.3.4.8 Ficheiro "adder8.vhd"

```
--*****************************************************************************
--                           TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "adder8.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 8 bits.                                                        *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder8 is
    port(in1,in2: in STD_LOGIC_VECTOR(7 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(7 downto 0);
         flag_carry: out STD_LOGIC);
```

```
end adder8;

architecture arch1 of adder8 is

    signal p,g:STD_LOGIC_VECTOR(7 downto 0);
    signal c: STD_LOGIC_VECTOR(8 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(7 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    flag_carry <= c(8);
end arch1;
```

E.3.4.9 Ficheiro "BufTB8.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "BufTB8.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state com portos*
--           de 8 bits.                                                       *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTB8 is
    port(DataIn: in STD_LOGIC_VECTOR(7 downto 0);
         DataOut: out STD_LOGIC_VECTOR(7 downto 0);
         T: in STD_LOGIC);
end BufTB8;

architecture arch1 of BufTB8 is
begin
    process(DataIn,T)
    begin
        if T='0' then
            DataOut<= "ZZZZZZZZ";
        else
            DataOut<= DataIn;
        end if;
    end process;
end arch1;
```

E.3.4.10 Ficheiro "BufTB16.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "BufTB16.vhd"                                                     *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state com portos*
--           de 16 bits.                                                      *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTB16 is
    port(DataIn: in STD_LOGIC_VECTOR(15 downto 0);
         DataOut: out STD_LOGIC_VECTOR(15 downto 0);
         T: in STD_LOGIC);
end BufTB16;

architecture arch1 of BufTB16 is
begin
    process(DataIn,T)
    begin
        if T='0' then
            DataOut<= "ZZZZZZZZZZZZZZZZ";
        else
```

```
            DataOut<= DataIn;
        end if;
    end process;
end arch1;
```

E.3.4.11 Ficheiro "BufTBid16.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "BufTBid16.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state           *
--           bidireccional com 2 portos de 16 bits                             *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTBid16 is
    port(T12,T21:in STD_LOGIC;
         entrada_saida1,entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
end BufTBid16;

architecture arch1 of BufTBid16 is
begin
    process(T12,T21,entrada_saida1,entrada_saida2)
    begin
        if(T12='0') then
            entrada_saida2<="ZZZZZZZZZZZZZZZZ";
        else
            entrada_saida2<=entrada_saida1;
        end if;

        if(T21='0') then
            entrada_saida1<="ZZZZZZZZZZZZZZZZ";
        else
            entrada_saida1<=entrada_saida2;
        end if;
    end process;
end arch1;
```

E.3.4.12 Ficheiro "BufTBid16v2.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "BufTBid16v2.vhd"                                                  *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state           *
--           bidireccional com 3 portos de 16 bits (1 in/out, 1 in, 1 out).    *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTBid16v2 is
```

```
    port(T12,T21:in STD_LOGIC;
         entrada1:in STD_LOGIC_VECTOR(15 downto 0);
         saida1:out STD_LOGIC_VECTOR(15 downto 0);
         entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
end BufTBid16v2;

architecture arch1 of BufTBid16v2 is
begin
    process(T12,T21,entrada1,entrada_saida2)
    begin
        if(T12='0') then
            entrada_saida2<="ZZZZZZZZZZZZZZZZ";
        else
            entrada_saida2<=entrada1;
        end if;

        if(T21='0') then
            saida1<="ZZZZZZZZZZZZZZZZ";
        else
            saida1<=entrada_saida2;
        end if;
    end process;
end arch1;
```

E.3.4.13 Ficheiro "Latch9.vhd"

```
--*********************************************************************************
--                          TRABALHO FINAL DE CURSO                                *
--*********************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*********************************************************************************
--FICHEIRO: "Latch9.vhd"                                                         *
--DESCRIÇÃO: Este ficheiro contem a descricao de um latch de 9 bits com entrada*
--           de RESET.                                                           *
--*********************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Latch9 is
    port(portoD: in  STD_LOGIC_VECTOR(8 downto 0);
         portoQ: buffer STD_LOGIC_VECTOR(8 downto 0);
         LE, CLR: in STD_LOGIC);
end Latch9;

architecture arch1 of Latch9 is
begin
    process(LE,CLR,portoD)
    begin
        if CLR='1' then
            portoQ<="000000000";
        elsif LE='1' then
            portoQ<= portoD;
        end if;
    end process;
end arch1;
```

E.3.4.14 Ficheiro "Flip_Flop16.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "Flip_Flop16.vhd"                                                   *
--DESCRIÇÃO: Este ficheiro contem a descricao de um registo de 16 bits formado *
--           por 16 Flip_Flops tipo D.                                          *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Flip_Flop16 is
    port(portoD:in  STD_LOGIC_VECTOR(15 downto 0);
         portoQ:buffer STD_LOGIC_VECTOR(15 downto 0);
         CLK,CLR:in STD_LOGIC);
end Flip_Flop16;

architecture arch1 of Flip_Flop16 is
begin
    process(CLK,CLR,portoD)
    begin
        if CLR='1' then
            portoQ<="0000000000000000";
        elsif CLK'event and CLK='1' then
            portoQ<= portoD;
        end if;
    end process;
end arch1;
```

E.3.4.15 Ficheiro "FFDs.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "FFDs.vhd"                                                          *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop tipo D com SET   *
--           assincrono.                                                        *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity FFDs is
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,set:in STD_LOGIC);
end FFDs;

architecture arch1 of FFDs is
begin
    process(D,clk,set)
    begin
        if set='1' then
            Q<='1';
        elsif clk'event and clk='1' then
```

```
            Q<= D;
        end if;
    end process;
end arch1;
```

E.3.4.16 Ficheiro "FFDr.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "FFDr.vhd"                                                        *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop tipo D com RESET *
--           assincrono.                                                      *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity FFDr is
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,reset:in STD_LOGIC);
end FFDr;

architecture arch1 of FFDr is
begin
    process(D,clk,reset)
    begin
        if reset='1' then
            Q<='0';
        elsif clk'event and clk='1' then
            Q<= D;
        end if;
    end process;
end arch1;
```

E.3.4.17 Ficheiro "adder16.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "adder16.vhd"                                                     *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 16 bits.                                                      *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder16 is
    port(in1,in2: in STD_LOGIC_VECTOR(15 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(15 downto 0);
         flag_carry: out STD_LOGIC);
end adder16;
```

```
architecture arch1 of adder16 is

    signal p,g: STD_LOGIC_VECTOR(15 downto 0);
    signal c: STD_LOGIC_VECTOR(16 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(15 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    c(9) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)
and p(8)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8))
or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(5) and p(6) and p(7) and p(8)) or
        (g(6) and p(7) and p(8)) or
        (g(7) and p(8)) or
        g(8));
```

```
    c(10) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9))
or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(6) and p(7) and p(8) and p(9)) or
        (g(7) and p(8) and p(9)) or
        (g(8) and p(9)) or
        g(9));

    c(11) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10))
or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(7) and p(8) and p(9) and p(10)) or
        (g(8) and p(9) and p(10)) or
        (g(9) and p(10)) or
        g(10));

    c(12) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(8) and p(9) and p(10) and p(11)) or
        (g(9) and p(10) and p(11)) or
        (g(10) and p(11)) or
        g(11));

    c(13) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(9) and p(10) and p(11) and p(12)) or
        (g(10) and p(11) and p(12)) or
        (g(11) and p(12)) or
        g(12));

    c(14) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13)) or
```

```
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(10) and p(11) and p(12) and p(13)) or
        (g(11) and p(12) and p(13)) or
        (g(12) and p(13)) or
        g(13));

    c(15) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) ) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13) and p(14)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13) and p(14)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(11) and p(12) and p(13) and p(14)) or
        (g(12) and p(13) and p(14)) or
        (g(13) and p(14)) or
        g(14));

    c(16) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and
p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14) and p(15)) or
          (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14) and p(15)) or
          (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14) and p(15)) or
          (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and
p(15)) or
          (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15))
or
          (g(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(12) and p(13) and p(14) and p(15)) or
          (g(13) and p(14) and p(15)) or
          (g(14) and p(15)) or
          g(15));

    flag_carry <= c(16);
end arch1;
```

F Co-processador de formatação e visualização de dados

F.1 ESQUEMA GERAL

O esquema geral do co-processador de formatação e visualização de dados encontra-se apresentado na página seguinte.

Colocar aqui o esquema do co-processador de visualização

F.2 DESCRIÇÃO DO CIRCUITO IMPLEMENTADO NA FPGA3

F.2.1 Descrição dos portos de entrada e saída

F.2.1.1 Descrição dos portos de entrada e saída do circuito de transferência por DMA

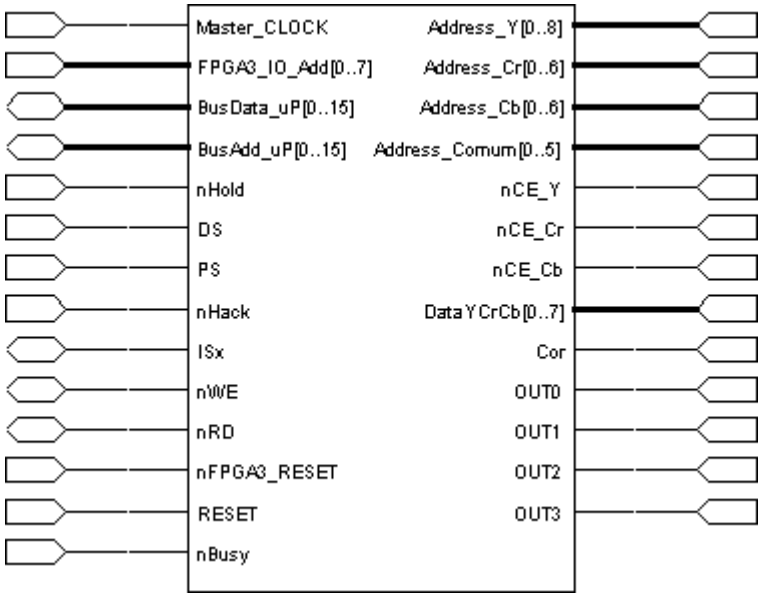


Figura F.1 - Descrição dos portos de entrada e saída do `Bloco_uP`.

PORTO	**DESCRIÇÃO**
`FPGA3_IO_Add`	8 bits mais significativos dos 16 que constituem os endereços dos registos implementados na FPGA3
`Master_Clock`	Relógio de 20 MHz
`BusData_uP`	Barramento de dados do DSP
`BusAdd_uP`	Barramento de endereços do DSP
`nHold` `nHack`	Sinais $\overline{\text{HOLD}}$ e $\overline{\text{HOLDA}}$ do DSP

`DS`
`PS`
`ISx`
	Sinais $\overline{\text{DS}}$, $\overline{\text{PS}}$ e $\overline{\text{IS}}$ do DSP, que identificam qual das áreas de memória é acedida (dados, programa ou I/O, respectivamente)

`nWE` `nRD`	Sinais $\overline{\text{WE}}$ e $\overline{\text{RD}}$ do DSP, correspondente aos sinais de controlo de escrita e leitura, respectivamente, da memória do DSP
`nFPGA3_RESET`	Sinal de inicialização da FPGA3
`RESET`	Sinal de inicialização activado pelo processador, através da escrita para o registo '`Reset`'
`nBusy`	Sinal $\overline{\text{BUSY}}$ das DPRAMs, assinalando contenção no acesso aos portos de memória. Este sinal é constituído pelo produto lógico dos sinais de $\overline{\text{BUSY}}$ das 3 DPRAMs
`Address_Y`	9 bits mais significativos dos endereços da DPRAM onde se armazenam as amostras Y
`Address_Cb`	7 bits mais significativos dos endereços da DPRAM onde se armazenam as amostras C_B

PORTO	DESCRIÇÃO
Address_Cr	7 bits mais significativos dos endereços da DPRAM onde se armazenam as amostras C_R
Address_Comum	6 bits de endereço menos significativos que são enviados para as 3 DPRAMs
nCE_Y nCE_Cb nCE_Cr	Sinais que controlam a escrita nas DPRAMs Y, C_B e C_R respectivamente, através da entrada $\overline{CE}$ dessas memórias
DataYCrCb	Saída de dados para as DPRAMs. São, na realidade, os 8 bits menos significativos do barramento de dados do DSP
Cor	Sinal que reflecte o valor programado no bit menos significativo do registo 'Cor' ('1' – policromático, '0' – monocromático) e que é utilizado na activação ou não das portas XOR junto dos registos
Out0 Out1 Out2 Out3	Resultado da descodificação de endereços (ver secção 3.2.3.4).

F.2.1.2 Descrição dos portos de entrada e saída do circuito de transferência da imagem para o conversor D/A

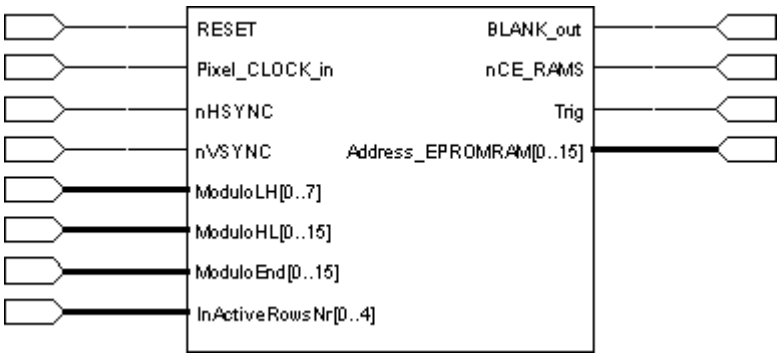


Figura F.2 - Descrição dos portos de entrada e saída do Bloco_BT.

PORTO	DESCRIÇÃO
Reset	Sinal de inicialização
Pixel_Clock_IN	Relógio de *pixel*, com uma frequência 4 vezes inferior à do relógio de *pixel* utilizado no conversor D/A
nHSYNC nVSYNC	Sinais de controlo provenientes do conversor D/A e que assinalam os retornos horizontal e vertical, respectivamente
BLANK_Out	Sinal que indica a presença de vídeo activo e que é enviado para o conversor, encontrando-se sincronizado com $\overline{HSYNC}$
ModuloLH ModuloHL ModuloEND	Valores que permitem alterar a configuração do sinal BLANK, aumentando ou reduzindo a duração do período de vídeo activo. Não estão disponíveis no exterior da FPGA, sendo forçados a valores fixos no seu interior
InActiveRowsNr	Número de linhas correspondentes ao retorno vertical, em que não é enviado sinal de vídeo para o conversor

PORTO	DESCRIÇÃO
Address_EPROMRAM	Barramento de endereços, em que os 13 bits mais significativos se destinam a endereçar as EPROMs e os restantes 3 bits são ligados directamente aos 3 bits menos significativos do barramento de endereços das DPRAMs
nCE_RAMs	Sinal de controlo de leitura nas DPRAMs (ciclo de leitura controlado por $\overline{CE}$)
Trig	Sinal de comando dos registos presentes nas saídas das DPRAMs

F.2.1.3 Descrição dos portos de entrada e saída do bloco PLL_Counters.

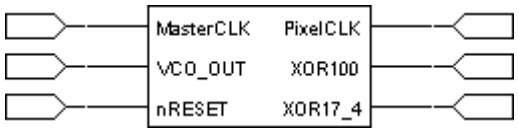
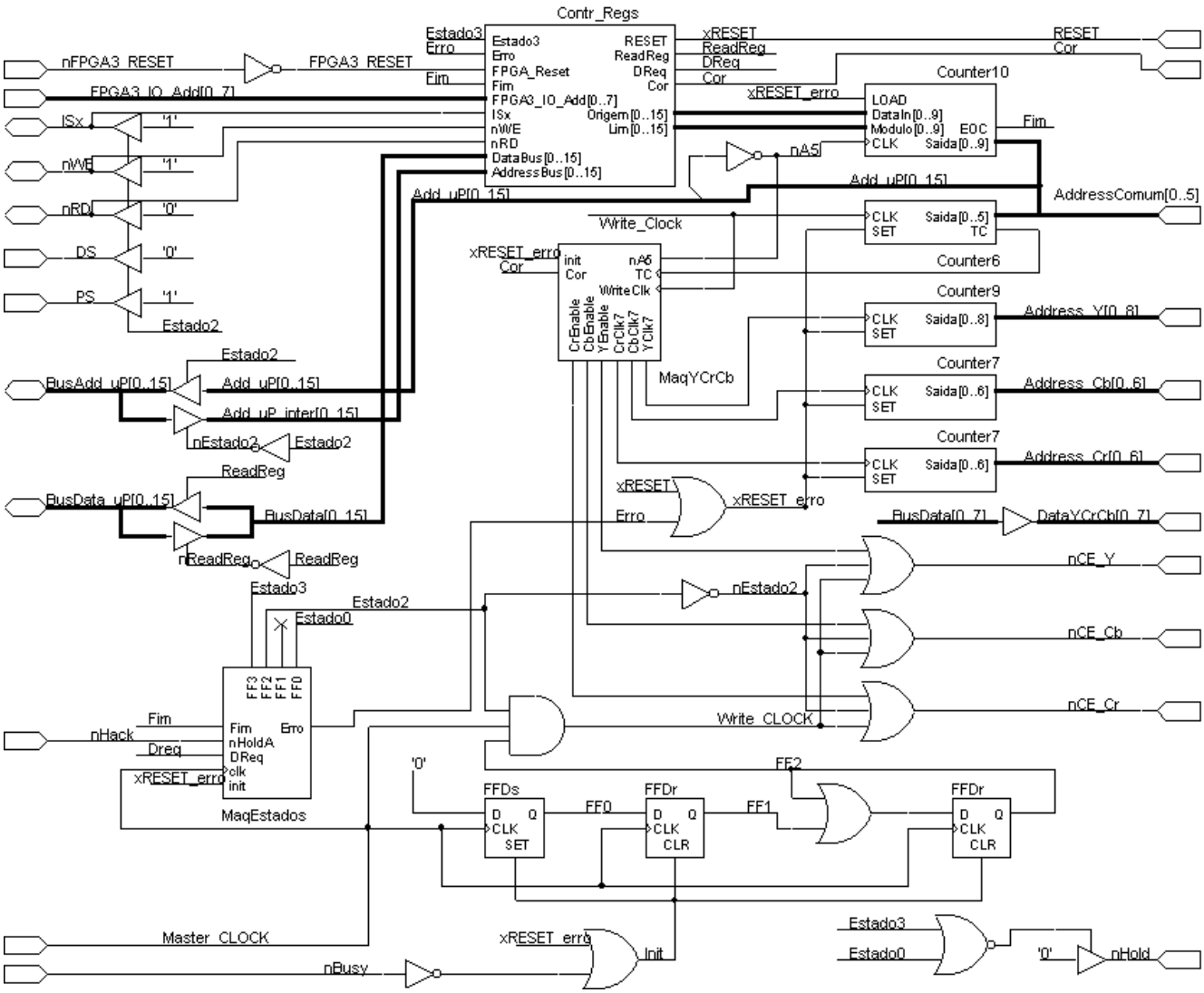


Figura F.3 - Descrição dos portos de entrada e saída do bloco PLL_Counters.

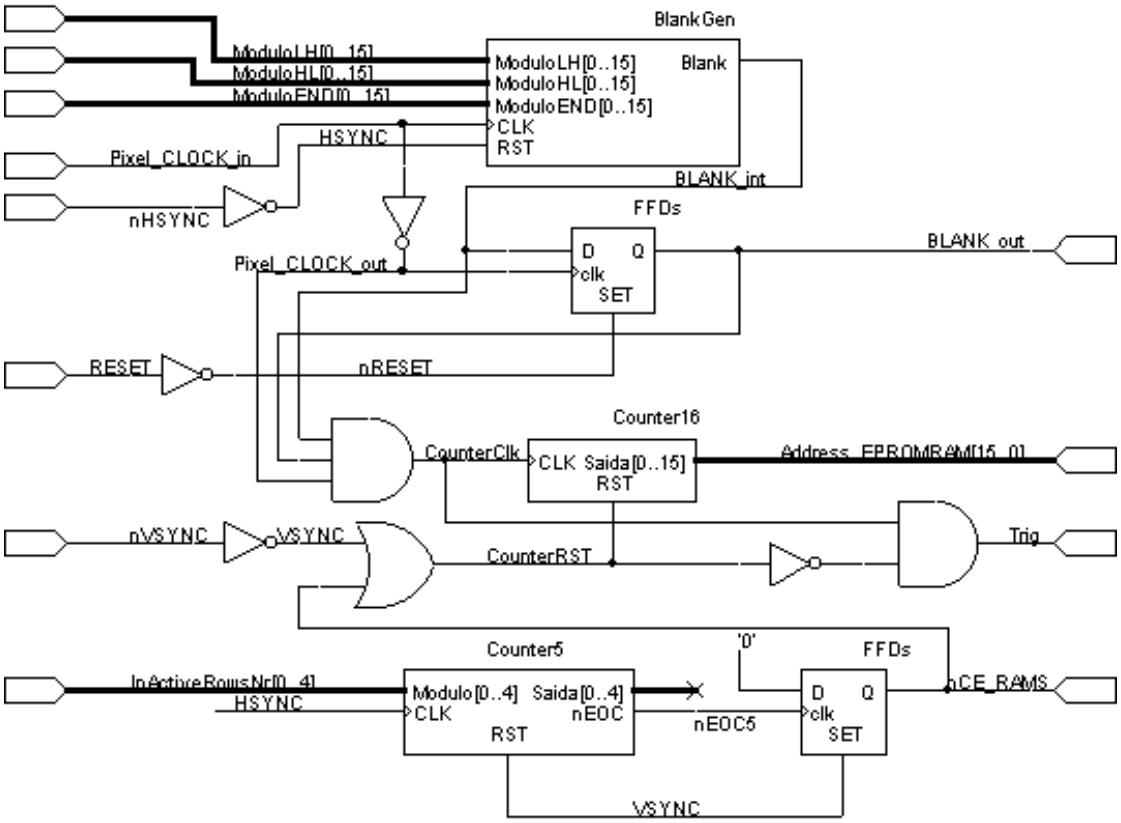
PORTO	DESCRIÇÃO
MasterClk	Relógio de referência de 20 MHz
VCO_Out	Saída do sintetizador de frequência e que corresponde à frequência de 13,6 MHz
PixelClk	Relógio de pixel com uma frequência de 3,4 MHz que não se encontra disponível no exterior da FPGA3
XOR100	Sinal correspondente à divisão do relógio de 20 MHz por 100 e que constitui uma das entradas do detector de fase
XOR17_4	Sinal correspondente à divisão por 68 do sinal à saída do sintetizador de frequência e que constitui uma das entradas do detector de fase
nRESET	Sinal de inicialização

F.2.2 Esquemas Lógicos

F.2.2.1 Esquema lógico do circuito de transferência por DMA



F.2.2.2 Esquema lógico do circuito de transferência da imagem para o conversor D/A



F.2.3 Descrição VHDL dos blocos implementados

F.2.3.1 Ficheiro "`fpga3.vhd`"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "fpga3.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito implementado na FPGA3*
--           pelo que contem a interligacao de todos os modulos que constituem *
--           esse mesmo circuito.                                             *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on


entity fpga3 is
    port(
        -- Geral
        FPGA3_IO_Add:in STD_LOGIC_VECTOR(7 downto 0);
        Master_Clock:in STD_LOGIC;
        nFPGA3_RESET: in STD_LOGIC;
        OUT0,OUT1,OUT2,OUT3: out STD_LOGIC;

         -- uProcessador
        BusData_uP: inout STD_LOGIC_VECTOR(15 downto 0);
        BusAdd_uP: inout STD_LOGIC_VECTOR(15 downto 0);
        nHold,DS,PS:out STD_LOGIC;
        nHack:in STD_LOGIC;
        ISx,nWE,nRD:inout STD_LOGIC;
        Address_Y: buffer STD_LOGIC_VECTOR(8 downto 0);
        Address_Cr: buffer STD_LOGIC_VECTOR(6 downto 0);
        Address_Cb: buffer STD_LOGIC_VECTOR(6 downto 0);
        Address_Comum: buffer STD_LOGIC_VECTOR(5 downto 0);
        DataYCrCb: out STD_LOGIC_VECTOR(7 downto 0);
        nBusy: in STD_LOGIC;
        nCE_Y,nCE_Cr,nCE_Cb:out STD_LOGIC;

        -- BrookTree
        nHSYNC,nVSYNC: in STD_LOGIC;
        BLANK: buffer STD_LOGIC;
        COR: out STD_LOGIC;
        Address_EPROMRAM: buffer STD_LOGIC_VECTOR(15 downto 0);
        nCE_RAMS:out STD_LOGIC;
        Trig:out STD_LOGIC;

        -- sinais de/para sintetizador
        Clock_BT : in STD_LOGIC;
        XORin_ref, XORin_div: out STD_LOGIC;
        Clock_BTout: out STD_LOGIC;

        -- readback
        rdbk_trig : in bit;
        rdbk_data : out bit);
end fpga3;


architecture arch1 of fpga3 is

    component Bloco_uP
        port(FPGA3_IO_Add:in STD_LOGIC_VECTOR(7 downto 0);
             Master_CLOCK:in STD_LOGIC;

             BusData_uP: inout STD_LOGIC_VECTOR(15 downto 0);
             BusAdd_uP: inout STD_LOGIC_VECTOR(15 downto 0);
             nHold,DS,PS:out STD_LOGIC;
             nHack:in STD_LOGIC;
```

```
            ISx,nWE,nRD:inout STD_LOGIC;

            nFPGA3_RESET: in STD_LOGIC;    -- Hardware RESET
            RESET: out STD_LOGIC;         -- Global RESET

            nBusy: in STD_LOGIC;
            Address_Y: buffer STD_LOGIC_VECTOR(8 downto 0);
            Address_Cr: buffer STD_LOGIC_VECTOR(6 downto 0);
            Address_Cb: buffer STD_LOGIC_VECTOR(6 downto 0);
            Address_Comum: buffer STD_LOGIC_VECTOR(5 downto 0);

            nCE_Y,nCE_Cr,nCE_Cb:out STD_LOGIC;
            DataYCrCb: out STD_LOGIC_VECTOR(7 downto 0);
            Cor: out STD_LOGIC;
            OUT0,OUT1,OUT2,OUT3: out STD_LOGIC);
    end component;

    component Bloco_BT
        port(RESET: in STD_LOGIC;         -- Global RESET
            Pixel_CLOCK_in: in STD_LOGIC;
            nHSYNC,nVSYNC: in STD_LOGIC;
            BLANK_out: buffer STD_LOGIC;
            ModuloLH,ModuloHL,ModuloEnd: in STD_LOGIC_VECTOR(7 downto 0);
            InActiveRowsNr: in STD_LOGIC_VECTOR(4 downto 0);
            Address_EPROMRAM: buffer STD_LOGIC_VECTOR(15 downto 0);
            nCE_RAMS:out STD_LOGIC;
            Trig:out STD_LOGIC);
    end component;

    component RDBK
        port(trig: in bit;
            data, rip: out bit);
    end component;

    component PLL_counters
        port(MasterCLK, VCO_OUT: in STD_LOGIC;
            PixelCLK, XOR100, XOR17_4: out STD_LOGIC;
            nRESET: in STD_LOGIC);
    end component;

    for all:Bloco_uP use entity work.Bloco_uP(arch1);
    for all:Bloco_BT use entity work.Bloco_BT(arch1);
    for all:PLL_counters use entity work.PLL_counters(arch1);

    signal RESET, PixelClk: STD_LOGIC;
    signal ModuloLH,ModuloHL,ModuloEND: STD_LOGIC_VECTOR(7 downto 0);
    signal InActiveRowsNr: STD_LOGIC_VECTOR(4 downto 0);
    signal rdbk_rip: bit;

begin

    ModuloLH <= "00100100";
    ModuloHL <= "11010100";
    ModuloEnd <= "11011010";
    InActiveRowsNr <= "10110";

    Block_uP: Bloco_uP port map(FPGA3_IO_Add, Master_Clock, BusData_uP, BusAdd_uP,
nHold, DS, PS, nHack, ISx, nWE, nRD, nFPGA3_RESET, RESET, nBusy, Address_Y, Address_Cr,
Address_Cb, Address_Comum, nCE_Y, nCE_Cr, nCE_Cb, DataYCrCb, Cor, OUT0, OUT1, OUT2,
OUT3);
    Block_BT: Bloco_BT port map(nFPGA3_RESET, PixelClk, nHSYNC, nVSYNC, BLANK,
ModuloLH, ModuloHL, ModuloEnd, InActiveRowsNr, Address_EPROMRAM, nCE_RAMS, Trig);

    xrdbk: RDBK port map (rdbk_trig, rdbk_data, rdbk_rip);

    PLL: PLL_counters port map (Master_Clock, Clock_BT, PixelClk, XORin_ref, XORin_div,
nFPGA3_RESET);

    Clock_BTout <= Clock_BT;
end arch1;
```

F.2.3.2 Ficheiro "Bloco_uP.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "Bloco_uP.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito responsavel pela fase*
--           de transferencia DMA, constituindo um dos blocos principais do    *
--           coprocessador                                                      *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Bloco_uP is
    port(FPGA3_IO_Add:in STD_LOGIC_VECTOR(7 downto 0);
         Master_CLOCK:in STD_LOGIC;

         BusData_uP: inout STD_LOGIC_VECTOR(15 downto 0);
         BusAdd_uP: inout STD_LOGIC_VECTOR(15 downto 0);
         nHold,DS,PS:out STD_LOGIC;
         nHack:in STD_LOGIC;
         ISx,nWE,nRD:inout STD_LOGIC;

         nFPGA3_RESET: in STD_LOGIC;    -- Hardware RESET
         RESET: out STD_LOGIC;         -- Global RESET

         nBusy: in STD_LOGIC;
         Address_Y: buffer STD_LOGIC_VECTOR(8 downto 0);
         Address_Cr: buffer STD_LOGIC_VECTOR(6 downto 0);
         Address_Cb: buffer STD_LOGIC_VECTOR(6 downto 0);
         Address_Comum: buffer STD_LOGIC_VECTOR(5 downto 0);

         nCE_Y,nCE_Cr,nCE_Cb:out STD_LOGIC;
         DataYCrCb: out STD_LOGIC_VECTOR(7 downto 0);
         Cor: out STD_LOGIC;
         OUT0,OUT1,OUT2,OUT3: out STD_LOGIC);
end Bloco_uP;

architecture arch1 of Bloco_uP is
    component MaqEstados
        port(clk,init:in  STD_LOGIC;
             DReq,nHoldA:in  STD_LOGIC;
             Fim:in  STD_LOGIC;
             FF0,FF1,FF2,FF3:buffer  STD_LOGIC;
             Erro: out STD_LOGIC);
    end component;

    component Contr_Regs
        port(DataBus:inout  STD_LOGIC_VECTOR(15 downto 0);
             AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
             FPGA3_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
             ISx,nWE,nRD:in STD_LOGIC;
             Origem:out  STD_LOGIC_VECTOR(15 downto 0);
             Lim:out  STD_LOGIC_VECTOR(15 downto 0);
             Cor:out STD_LOGIC;
             DReq:out STD_LOGIC;
             ReadReg:out STD_LOGIC;
             FPGA_Reset:in STD_LOGIC;
             RESET:out STD_LOGIC;
             Erro: in STD_LOGIC;
             Fim,Estado3:in STD_LOGIC);
    end component;

    component Counter10
        port(CLK,LOAD: in STD_LOGIC;
             DataIn: in STD_LOGIC_VECTOR(9 downto 0);
             Saida: buffer STD_LOGIC_VECTOR(9 downto 0);
             Modulo: in STD_LOGIC_VECTOR(9 downto 0);
```

```
            EOC: out STD_LOGIC);
    end component;

    component Counter9
        port(CLK,SET: in STD_LOGIC;
             Saida: buffer STD_LOGIC_VECTOR(8 downto 0));
    end component;

    component Counter7
        port(CLK,SET: in STD_LOGIC;
             Saida: buffer STD_LOGIC_VECTOR(6 downto 0));
    end component;

    component Counter6
        port(CLK,SET: in STD_LOGIC;
             Saida: buffer STD_LOGIC_VECTOR(5 downto 0);
             TC: out STD_LOGIC);
    end component;

    component MaqYCrCb
        port(TC,WriteClk,init:in  STD_LOGIC;
             Cor,nA5:in  STD_LOGIC;
             YClk7,CrClk7,CbClk7:out  STD_LOGIC;
             YEnable,CrEnable,CbEnable:out  STD_LOGIC);
    end component;

    component BufTB16
        port(DataIn: in STD_LOGIC_VECTOR(15 downto 0);
             DataOut: out STD_LOGIC_VECTOR(15 downto 0);
             T: in STD_LOGIC);
    end component;

    component BufTBid
        port(T12,T21:in STD_LOGIC;
             entrada_saida1,entrada_saida2: inout STD_LOGIC);
    end component;

    component BufTBid16
        port(T12,T21:in STD_LOGIC;
             entrada_saida1,entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
    end component;

    component BufTBid16v2
        port(T12,T21:in STD_LOGIC;
             entrada1:in STD_LOGIC_VECTOR(15 downto 0);
             saida1:out STD_LOGIC_VECTOR(15 downto 0);
             entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
    end component;

    component FFDs
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
    end component;

    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    component Ext_Add_Dec
        port(AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
             FPGA3_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
             ISx:in STD_LOGIC;
             OUT0,OUT1,OUT2,OUT3: out STD_LOGIC);
    end component;

    for all: MaqEstados use entity work.MaqEstados(arch1);
    for all: Contr_Regs use entity work.Contr_Regs(arch1);
    for all: Counter10 use entity work.Counter10(arch1);
    for all: Counter9 use entity work.Counter9(arch1);
    for all: Counter7 use entity work.Counter7(arch1);
    for all: Counter6 use entity work.Counter6(arch1);
    for all: MaqYCrCb use entity work.MaqYCrCb(arch1);
    for all: BufTB16 use entity work.BufTB16(arch1);
    for all: BufTBid use entity work.BufTBid(arch1);
    for all: BufTBid16 use entity work.BufTBid16(arch1);
    for all: BufTBid16v2 use entity work.BufTBid16v2(arch1);
```

```
    for all: FFDs use entity work.FFDs(arch1);
    for all: FFDr use entity work.FFDr(arch1);
    for all:Ext_Add_Dec use entity work.Ext_Add_Dec(arch1);

    signal xRESET, FPGA3_RESET: STD_LOGIC;
    signal Write_CLOCK,nWrite_CLOCK,Write_CLOCK_fast: STD_LOGIC;
    signal Dreq,ReadReg,Fim,xCor: STD_LOGIC;
    signal Estado0,Estado1,Estado2,Estado3,nEstado2: STD_LOGIC;
    signal Origem,Lim: STD_LOGIC_VECTOR(15 downto 0);
    signal BusData: STD_LOGIC_VECTOR(15 downto 0);
    signal nReadReg: STD_LOGIC;

    signal Add_uP,Add_uP_inter,Add_uP_inter2: STD_LOGIC_VECTOR(15 downto 0);
    signal ClkCounter10,nA5: STD_LOGIC;
    signal zIS,zWE,zRD: STD_LOGIC;
    signal TC:STD_LOGIC;
    signal YClk7,CrClk7,CbClk7:STD_LOGIC;
    signal YEnable,CrEnable,CbEnable:STD_LOGIC;

    signal ZERO:STD_LOGIC;
    signal FF0,FF1,FF2:STD_LOGIC;
    signal FF0in,FF1in,FF2in:STD_LOGIC;
    signal Init:STD_LOGIC;

    signal Erro,xRESET_erro: STD_LOGIC;

    signal err0in, err0, err1in, err1: STD_LOGIC;

begin
    ZERO <= '0';

    RESET <= xRESET;
    FPGA3_RESET <= not nFPGA3_RESET;
    FF0in <= ZERO;
    FF1in <= FF0;
    FF2in <= FF1 or FF2;

    Init <= (xReset_erro or not(nBusy) or err1);
    -- maquina de estados para paragem de escrita nas DPRAM ate' o sinal de Busy
    -- ser desactivado
    FF_0:FFDs port map(FF0in,FF0,Master_Clock,Init);
    FF_1:FFDr port map(FF1in,FF1,Master_Clock,Init);
    FF_2:FFDr port map(FF2in,FF2,Master_Clock,Init);

    FFerr0: FFDr port map (err0in,err0,Master_Clock,xRESET);
    FFerr1: FFDr port map (err1in,err1,Master_Clock,xRESET);
    err0in <= not (FF0 or FF1 or FF2);
    err1in <= (err0 and err0in);

    Write_CLOCK_fast <= (Master_Clock and FF2 and Estado2);
    nWrite_CLOCK <= not Write_CLOCK;
    div_WC: FFDs port map (nWrite_CLOCK,Write_CLOCK,Write_CLOCK_fast,xRESET_erro);

    nReadReg <= not ReadReg;

    BUFTBidBusData:BufTBid16 port map (nReadReg,ReadReg,BusData_uP,BusData);
    BUFTBidIS:BufTBid port map (nEstado2,Estado2,ISx,zIS);
    BUFTBidWE:BufTBid port map (nEstado2,Estado2,nWE,zWE);
    BUFTBidRD:BufTBid port map (nEstado2,Estado2,nRD,zRD);

    DataYCrCb(7 downto 0) <= BusData_uP(7 downto 0);

    xRESET_erro <= xRESET or Erro;

    Main_Mach:MaqEstados port map (Master_Clock, xRESET_erro, Dreq, nHack, Fim,
Estado0, Estado1, Estado2, Estado3, Erro);
    Ctr_Regs: Contr_Regs port map (BusData, Add_uP_inter, FPGA3_IO_Add, zIS, zWE, zRD,
Origem, Lim, xCor, DReq, ReadReg, FPGA3_RESET, xRESET, Erro, Fim, Estado3);
    Count10:Counter10 port map (ClkCounter10, xRESET_erro, Origem(15 downto 6),
Add_uP(15 downto 6), Lim(15 downto 6), Fim);
    ClkCounter10 <= ((xRESET_erro and Master_Clock) or (not(xRESET_erro)and nA5));

    Count9Y:Counter9 port map (YClk7,xRESET_erro,Address_Y);
    nCE_Y <= (Write_CLOCK or not(YEnable)) or nEstado2 ;

    Count7Cr:Counter7 port map (CrClk7,xRESET_erro,Address_Cr);
    nCE_Cr <= (Write_CLOCK or not(CrEnable)) or nEstado2 ;

    Count7Cb:Counter7 port map (CbClk7,xRESET_erro,Address_Cb);
```

```
    nCE_Cb <= (Write_CLOCK or not(CbEnable)) or nEstado2 ;

    Count6:Counter6 port map (Write_CLOCK,xRESET_erro,Add_uP(5 downto 0),TC);
    MchYCrCb:MaqYCrCb port map
(TC,Write_CLOCK,xRESET_erro,xCor,nA5,YClk7,CrClk7,CbClk7,YEnable,CrEnable,CbEnable);
    nA5 <= not(Add_uP(5));

    Address_Comum <= Add_uP(5 downto 0);

    nEstado2 <= not(Estado2);
    BUFT:BufTBid16v2 port map (Estado2,nEstado2,Add_uP_inter2,Add_uP_inter,BusAdd_uP);
    BUFT2:BufTB16 port map (Add_uP,Add_uP_inter2,Estado2);

    process(Estado0,Estado3)
    begin
        if ( not (Estado3='1' or Estado0='1')) then
            nHold <= '0';
        else
            nHold <= 'Z';
        end if;
    end process;

    process(Estado2)
    begin
        if (Estado2='1') then
            DS <= '0';
            PS <= '1';
            zIS <= '1';
            zWE <= '1';
            zRD <= '0';
        else
            DS <= 'Z';
            PS <= 'Z';
            zIS <= 'Z';
            zWE <= 'Z';
            zRD <= 'Z';
        end if;
    end process;

    Cor <= xCor;
    Descodif_Ender: Ext_Add_Dec port
map(Add_uP_inter,FPGA3_IO_Add,zIS,OUT0,OUT1,OUT2,OUT3);

end arch1;
```

F.2.3.3 Ficheiro "Bloco_BT.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Bloco_BT.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito responsavel pela fase*
--           de transferencia da imagem nas 3 DPRAM para o conversor D/A.     *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Bloco_BT is
    port(RESET: in STD_LOGIC;         -- Global RESET

         Pixel_CLOCK_in: in STD_LOGIC;
         nHSYNC,nVSYNC: in STD_LOGIC;
         BLANK_out: buffer STD_LOGIC;
         ModuloLH,ModuloHL,ModuloEnd: in STD_LOGIC_VECTOR(7 downto 0);
         InActiveRowsNr: in STD_LOGIC_VECTOR(4 downto 0);

         Address_EPROMRAM: buffer STD_LOGIC_VECTOR(15 downto 0);
```

```
         nCE_RAMS:out STD_LOGIC;
         Trig:out STD_LOGIC);
end Bloco_BT;


architecture arch1 of Bloco_BT is
    component BlankGen
        port(CLK,RST: in STD_LOGIC;
             ModuloLH,ModuloHL,ModuloEnd: in STD_LOGIC_VECTOR(7 downto 0);
             Blank: out STD_LOGIC);
    end component;

    component Counter16
        port(CLK,RST: in STD_LOGIC;
             Saida: buffer STD_LOGIC_VECTOR(15 downto 0));
    end component;

    component Counter5
        port(CLK,RST: in STD_LOGIC;
             Saida: buffer STD_LOGIC_VECTOR(4 downto 0);
             Modulo: in STD_LOGIC_VECTOR(4 downto 0);
             nEOC: out STD_LOGIC);
    end component;

    component FFDs
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
    end component;

    for all: BlankGen use entity work.BlankGen(arch1);
    for all: Counter16 use entity work.Counter16(arch1);
    for all: Counter5 use entity work.Counter5(arch1);
    for all: FFDs use entity work.FFDs(arch1);

    signal ZERO: STD_LOGIC;
    signal Pixel_CLOCK_out: STD_LOGIC;
    signal CounterClk: STD_LOGIC;
    signal CounterRST: STD_LOGIC;
    signal nEOC5: STD_LOGIC;
    signal Counter5out: STD_LOGIC_VECTOR(4 downto 0);
    signal BLANK_int: STD_LOGIC;
    signal VSync,HSync: STD_LOGIC;
    signal ActiveRow: STD_LOGIC;
    signal nRESET: STD_LOGIC;

begin
    ZERO <= '0';

    VSync <= not nVSYNC;
    HSync <= not nHSYNC;

    Pixel_CLOCK_out <= not(Pixel_CLOCK_in);  -- atrasado de 180 graus.

    BLK_GEN: BlankGen port
map(Pixel_CLOCK_in,HSync,ModuloLH,ModuloHL,ModuloEnd,BLANK_int);

    nRESET <= not RESET;
    FF1: FFDs port map(BLANK_int,BLANK_out,Pixel_CLOCK_out,nRESET);

    Count16: Counter16 port map(CounterClk,CounterRST,Address_EPROMRAM);
    CounterClk <= (Pixel_CLOCK_out and BLANK_int and BLANK_out);
    CounterRST <= (VSync or ActiveRow);

    Count5: Counter5 port map(HSync,VSync,Counter5out,InActiveRowsNr,nEOC5);

    FF2: FFDs port map(ZERO,ActiveRow,nEOC5,VSync);

    nCE_RAMS <= ActiveRow;

    Trig <= (CounterClk and not(ActiveRow));
end arch1;
```

F.2.3.4 Ficheiro "PLL_counters.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "PLL_counters.vhd"                                                *
--DESCRIÇÃO: Este ficheiro contem a descricao da interligacao entre os diversos*
--           contadores que fazem parte do sintetizador de frequencia.        *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on


entity PLL_counters is
    port(MasterCLK, VCO_OUT : in STD_LOGIC;
         PixelCLK, XOR100, XOR17_4: out STD_LOGIC;
         nRESET: in STD_LOGIC);
end PLL_counters;

architecture arch1 of PLL_counters is

    component Div17
        port(CLK: in STD_LOGIC;
             CLKout: out STD_LOGIC;
             RESET: in STD_LOGIC);
    end component;

    component Div100
        port(CLK: in STD_LOGIC;
                  CLKout: out STD_LOGIC;
             RESET: in STD_LOGIC);
    end component;

    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    FOR all: Div17 use entity work.Div17(arch1);
    FOR all: Div100 use entity work.Div100(arch1);
    FOR all: FFDr use entity work.FFDr(arch1);

    signal ZERO,pos17, RESET: STD_LOGIC;
    signal Qpre17, Qpos17, Qpixel, nQpre17, nQpos17, nQpixel: STD_LOGIC;

begin
    ZERO <= '0';

    nQpre17 <= not Qpre17;
    nQpos17 <= not Qpos17;
    nQpixel <= not Qpixel;

    RESET <= not nRESET;

    div2pre17: FFDr port map(nQpre17, Qpre17, VCO_OUT, RESET);
    div2pos17: FFDr port map(nQpos17, Qpos17, pos17, RESET);

    XOR17_4 <= Qpos17;

    div_17: Div17 port map(Qpre17, pos17,RESET);
    div2pixclk: FFDr port map(nQpixel, Qpixel, Qpre17, RESET);

    PixelCLK <= Qpixel;

    div_100: Div100 port map (MasterCLK, XOR100,RESET);
end arch1;
```

F.2.3.5 Ficheiro "Ext_Add_Dec.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Ext_Add_Dec.vhd"                                                 *
--DESCRIÇÃO: Este ficheiro contem a descricao do descodificador de enderecos  *
--           auxiliares.                                                      *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Ext_Add_Dec is
    port(AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
         FPGA3_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
         ISx:in STD_LOGIC;
         OUT0,OUT1,OUT2,OUT3: out STD_LOGIC);
end Ext_Add_Dec;

architecture arch1 of Ext_Add_Dec is

    signal EQU:STD_LOGIC;

begin
    process(FPGA3_IO_Add,AddressBus,ISx)
    begin
        if ((AddressBus(15 downto 8)=FPGA3_IO_Add)and ISx='0' and (AddressBus(7 downto
2)="010101")) then
            EQU<='1';
        else
            EQU<='0';
        end if;
    end process;

    OUT0 <= EQU and (not AddressBus(1)) and (not AddressBus(0));
    OUT1 <= EQU and (not AddressBus(1)) and (    AddressBus(0));
    OUT2 <= EQU and (    AddressBus(1)) and (not AddressBus(0));
    OUT3 <= EQU and (    AddressBus(1)) and (    AddressBus(0));

end arch1;
```

F.2.3.6 Ficheiro "BlankGen.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "BlankGen.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao do circuito responsavel pela    *
--           geracao do sinal BLANK que sera' enviado ao conversor D/A.       *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BlankGen is
    port(CLK,RST: in STD_LOGIC;
         ModuloLH,ModuloHL,ModuloEND: in STD_LOGIC_VECTOR(7 downto 0);
         Blank: out STD_LOGIC);
```

```
end BlankGen;

architecture arch1 of BlankGen is
    component adder8
        port(in1,in2: in STD_LOGIC_VECTOR(7 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(7 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder8 use entity work.adder8(arch1);

    signal CarryOut: STD_LOGIC;
    signal Saida,SaidaM1: STD_LOGIC_VECTOR(7 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(7 downto 0);
    signal ONES: STD_LOGIC_VECTOR(7 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "11111111";
    ZEROS <= "00000000";

    Adder: adder8 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,RST,ModuloLH,ModuloHL,ModuloEnd,saida,saidaM1,ZEROS,ONES)
    begin
        if RST='1' then
            saida<= ZEROS;
            Blank<='0';
        elsif CLK'event and CLK='1' then
            if saida=ONES then
                saida<=ZEROS;
                Blank<='0';
            elsif saida=ModuloLH then
                Blank<='1';
                saida<=SaidaM1;
            elsif saida=ModuloHL then
                saida<=SaidaM1;
                Blank<='0';
            elsif saida=ModuloEnd then
                saida<=SaidaM1;
                Blank<='0';
            else
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.7 Ficheiro "MaqEstados.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                            *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "MaqEstados.vhd"                                                   *
--DESCRIÇÃO: Este ficheiro contem a descricao da maquina de estados que tem,   *
--           como funcao principal, controlar a transferencia de dados por DMA *
--           entre o DSP e as 3 DPRAMs.                                        *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity MaqEstados is
    port(clk,init:in  STD_LOGIC;
         DReq,nHoldA:in  STD_LOGIC;
         Fim:in  STD_LOGIC;
         FF0,FF1,FF2,FF3:buffer  STD_LOGIC;
         Erro: out STD_LOGIC);
```

```
end MaqEstados;

architecture arch1 of MaqEstados is

    component FFDs
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
    end component;

    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    for all: FFDr use entity work.FFDr(arch1);
    for all: FFDs use entity work.FFDs(arch1);

    signal zero: STD_LOGIC;
    signal inFF0,inFF1,inFF2,inFF3: STD_LOGIC;
    signal sinal_erro, xErro, inFFerr2, FFerr1: STD_LOGIC;

begin
    zero <= '0';

    inFF0 <= ((not(DReq) and FF0) or (nHoldA and FF3));
    inFF1 <= (DReq and FF0) or (nHoldA and FF1);
    inFF2 <= (not(nHoldA) and FF1) or (not(Fim) and FF2);
    inFF3 <= ((Fim and FF2) or (not(nHoldA) and FF3));

    flip0:FFDs port map (inFF0,FF0,clk,init);
    flip1:FFDr port map (inFF1,FF1,clk,init);
    flip2:FFDr port map (inFF2,FF2,clk,init);
    flip3:FFDr port map (inFF3,FF3,clk,init);

--  Deteccao de estado '0000' durante mais do que 1 ciclo de clk
    sinal_erro <= not (FF0 or FF1 or FF2 or FF3);
    fliperr1: FFDr port map (sinal_erro, FFerr1, clk, init);
    inFFerr2 <= FFerr1 and sinal_erro;
    fliperr2: FFDr port map (inFFerr2, xErro, clk, init);

    Erro <= xErro;
end arch1;
```

F.2.3.8 Ficheiro "MaqYCrCb.vhd"

```
--*****************************************************************************
--                           TRABALHO FINAL DE CURSO                          *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "MaqYCrCb.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao da maquina de estados que tem,   *
--           como funcao principal, gerar os sinais de controlo de escrita para*
--           as 3 DPRAMs.                                                     *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity MaqYCrCb is
    port(TC,WriteClk,init:in  STD_LOGIC;
         Cor,nA5:in  STD_LOGIC;
         YClk7,CrClk7,CbClk7:out  STD_LOGIC;
         YEnable,CrEnable,CbEnable:out  STD_LOGIC);
end MaqYCrCb;

architecture arch1 of MaqYCrCb is

    component FFDs
```

```
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
    end component;

    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    for all: FFDr use entity work.FFDr(arch1);
    for all: FFDs use entity work.FFDs(arch1);

    signal zero: STD_LOGIC;
    signal inFFA,inFFB,inFFC,inFFD,inFFE,inFFF: STD_LOGIC;
    signal A,B,C,D,E,F: STD_LOGIC;
    signal ABCD: STD_LOGIC;
    signal YEx,CrEx,CbEx: STD_LOGIC;

begin
    zero <= '0';

    inFFA <= ((Cor and F) or (not(Cor) and D));
    inFFB <= A;
    inFFC <= B;
    inFFD <= C;
    inFFE <= (Cor and D);
    inFFF <= E;

    flipA:FFDs port map (inFFA,A,TC,init);
    flipB:FFDr port map (inFFB,B,TC,init);
    flipC:FFDr port map (inFFC,C,TC,init);
    flipD:FFDr port map (inFFD,D,TC,init);
    flipE:FFDr port map (inFFE,E,TC,init);
    flipF:FFDr port map (inFFF,F,TC,init);

    ABCD <= (A or B or C or D);
    YClk7 <= (ABCD and nA5);
    CbClk7 <= (E and nA5);
    CrClk7 <= (F and nA5);

    flipYE: FFDr port map (ABCD,YEx,WriteClk,init);
    flipCbE:FFDr port map (E,CbEx,WriteClk,init);
    flipCrE:FFDr port map (F,CrEx,WriteClk,init);

    YEnable <= YEx;
    CbEnable <= CbEx;
    CrEnable <= CrEx;

end arch1;
```

F.2.3.9 Ficheiro "Contr_Regs.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Contr_Regs.vhd"                                                  *
--DESCRIÇÃO: Este ficheiro contem a descricao do banco de registos do coproces-*
--          sador.                                                           *
--*****************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Contr_Regs is
    port(DataBus:inout  STD_LOGIC_VECTOR(15 downto 0);
         AddressBus:in  STD_LOGIC_VECTOR(15 downto 0);
```

```
         FPGA3_IO_Add:in  STD_LOGIC_VECTOR(7 downto 0);
         ISx,nWE,nRD:in STD_LOGIC;
         Origem:out  STD_LOGIC_VECTOR(15 downto 0);
         Lim:out  STD_LOGIC_VECTOR(15 downto 0);
         Cor:out STD_LOGIC;
         DReq:out STD_LOGIC;
         ReadReg:out STD_LOGIC;
         FPGA_Reset:in STD_LOGIC;
         RESET:out STD_LOGIC;
         Erro: in STD_LOGIC;
         Fim,Estado3:in STD_LOGIC);
end Contr_Regs;

architecture arch1 of Contr_Regs is
    component Flip_Flop16
        port(portoD:in  STD_LOGIC_VECTOR(15 downto 0);
             portoQ:buffer STD_LOGIC_VECTOR(15 downto 0);
             CLK,CLR:in STD_LOGIC);
    end component;

    component FFDs
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
    end component;

    for all:Flip_Flop16 use entity work.Flip_Flop16(arch1);
    for all: FFDs use entity work.FFDs(arch1);

    signal FFout0,FFout1:STD_LOGIC_VECTOR(15 downto 0);
    signal CLK0,CLK1:STD_LOGIC;
    signal T0,T1,EQU:STD_LOGIC;
    signal Soft_Reset,xReset:STD_LOGIC;
    signal CLR0:STD_LOGIC;
    signal xcor,TCor,ClkCor:STD_LOGIC;
    signal ZERO:STD_LOGIC;

begin
    ZERO <= '0';

    process(FPGA3_IO_Add,AddressBus,ISx)
    begin
        if ((AddressBus(15 downto 8)=FPGA3_IO_Add) and ISx='0' and (AddressBus(7 downto
2)="010100")) then
            EQU<='1';
        else
            EQU<='0';
        end if;
    end process;

    CLK0 <= not (not(nWE) and EQU and (not(AddressBus(1))) and (not(AddressBus(0))));
    CLK1 <= not (not(nWE) and EQU and (not(AddressBus(1))) and (   (AddressBus(0))));

    T0 <= (not(nRD) and EQU and (not(AddressBus(1))) and (not(AddressBus(0))));
    T1 <= (not(nRD) and EQU and (not(AddressBus(1))) and (   (AddressBus(0))));
    ReadReg <= (not(nRD) and EQU and (not(AddressBus(1))));

    FF0: Flip_Flop16 port map(DataBus,FFout0,CLK0,CLR0);
    FF1: Flip_Flop16 port map(DataBus,FFout1,CLK1,ZERO);

    Origem <= FFout1;
    Lim <= FFout0;
    DReq <= FFout0(0) or FFout0(1) or FFout0(2) or FFout0(3) or FFout0(4) or FFout0(5)
or FFout0(6) or FFout0(7) or FFout0(8) or FFout0(9) or FFout0(10) or FFout0(11) or
FFout0(12) or FFout0(13) or FFout0(14) or FFout0(15);

    process(T0,FFout0)
    begin
        if T0='0' then
            DataBus <= "ZZZZZZZZZZZZZZZZ";
        else
            DataBus <= FFout0;
        end if;
    end process;

    process(T1,FFout1)
    begin
        if T1='0' then
            DataBus <= "ZZZZZZZZZZZZZZZZ";
```

```
        else
            DataBus <= FFout1;
        end if;
    end process;

    Soft_Reset <= (EQU and AddressBus(1) and AddressBus(0));
    xReset <= (FPGA_Reset or Soft_Reset);
    RESET <= xReset;
    CLR0 <= (xReset or (Fim and Estado3) or Erro);

    FFCor: FFDs port map(DataBus(0),xcor,ClkCor,xReset);
    ClkCor <= not (EQU and AddressBus(1) and not(AddressBus(0)) and not(nWE));
    TCor <= (EQU and AddressBus(1) and not(AddressBus(0)) and not(nRD));

    process(TCor,xcor)
    begin
        if TCor='0' then
            DataBus <= "ZZZZZZZZZZZZZZZZ";
        else
            DataBus <= "000000000000000" & xcor;
        end if;
    end process;

    Cor <= xcor;
end arch1;
```

F.2.3.10 Ficheiro "Div17.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Div17.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao do divisor de frequencia por 17  *
--           que faz parte do conjunto de contadores pertencentes ao sintetiza-*
--           dor de frequencia                                                 *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Div17 is
    port(CLK: in STD_LOGIC;
         CLKout: out STD_LOGIC;
         RESET: in STD_LOGIC);
end Div17;

architecture arch1 of Div17 is
    component adder5
        port(in1,in2: in STD_LOGIC_VECTOR(4 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(4 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder5 use entity work.adder5(arch1);

    signal CarryOut: STD_LOGIC:='0';
    signal Saida: STD_LOGIC_VECTOR(4 downto 0):="00000";
    signal SaidaM1: STD_LOGIC_VECTOR(4 downto 0):="00001";
    signal ZEROS: STD_LOGIC_VECTOR(4 downto 0);
    signal MAXm1: STD_LOGIC_VECTOR(4 downto 0);
    signal HALFWAYm1: STD_LOGIC_VECTOR(4 downto 0);
    signal ONE,ZERO: STD_LOGIC;

begin
    ONE <='1';
    ZERO <= '0';
    MAXm1 <= "10000";
```

```
    ZEROS <= "00000";
    HALFWAYm1 <= "01000";

    Adder: adder5 port map(Saida, ZEROS, ONE, SaidaM1, CarryOut);
    process(CLK, saida, saidaM1, MAXm1, HALFWAYm1, RESET, ZEROS, ZERO, ONE)
    begin
        if RESET='1' then
            saida<=ZEROS;
            CLKout<=ZERO;
        elsif CLK'event and CLK='1' then
            if saida=MAXm1 then
                saida<=ZEROS;
                CLKout<=ZERO;
            elsif saida=HALFWAYm1 then
                CLKout<=ONE;
                saida<= saidaM1;
            else
                saida <= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.11 Ficheiro "`Div100.vhd`"

```
--*****************************************************************************
--                           TRABALHO FINAL DE CURSO                            *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "Div100.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao do divisor de frequencia por 100 *
--           que faz parte do conjunto de contadores pertencentes ao sintetiza-*
--           dor de frequencia                                                 *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Div100 is
    port(CLK: in STD_LOGIC;
         CLKout: out STD_LOGIC;
         RESET: in STD_LOGIC);
end Div100;

architecture arch1 of Div100 is
    component adder7
        port(in1,in2: in STD_LOGIC_VECTOR(6 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(6 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder7 use entity work.adder7(arch1);

    signal CarryOut: STD_LOGIC:='0';
    signal Saida: STD_LOGIC_VECTOR(6 downto 0):="0000000";
    signal SaidaM1: STD_LOGIC_VECTOR(6 downto 0):="0000001";
    signal ZEROS: STD_LOGIC_VECTOR(6 downto 0);
    signal MAXm1: STD_LOGIC_VECTOR(6 downto 0);
    signal HALFWAYm1: STD_LOGIC_VECTOR(6 downto 0);
    signal ONE,ZERO: STD_LOGIC;

begin
    ONE <='1';
    ZERO <= '0';
    MAXm1 <= "1100011";
    ZEROS <= "0000000";
    HALFWAYm1 <= "0110001";
```

```
    Adder: adder7 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,saida,saidaM1,MAXm1,HALFWAYm1,RESET,ZEROS,ZERO,ONE)
        begin
            if RESET='1' then
                saida<=ZEROS;
                CLKout<=ZERO;
            elsif CLK'event and CLK='1' then
                if saida=MAXm1 then
                    saida<=ZEROS;
                    CLKout<=ZERO;
                elsif saida=HALFWAYm1 then
                    CLKout<=ONE;
                    saida<= saidaM1;
                else
                    saida <= saidaM1;
                end if;
            end if;
        end process;
end arch1;
```

F.2.3.12 Ficheiro "Counter10.vhd"

```
--*********************************************************************************
--                          TRABALHO FINAL DE CURSO                                *
--*********************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*********************************************************************************
--FICHEIRO: "Counter10.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 10 bits com     *
--           LOAD assincrono e sinalizacao de fim de contagem.                   *
--*********************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter10 is
    port(CLK,LOAD: in STD_LOGIC;
         DataIn: in STD_LOGIC_VECTOR(9 downto 0);
         Saida: buffer STD_LOGIC_VECTOR(9 downto 0);
         Modulo: in STD_LOGIC_VECTOR(9 downto 0);
         EOC: out STD_LOGIC);
end Counter10;

architecture arch1 of Counter10 is
    component adder10
        port(in1,in2: in STD_LOGIC_VECTOR(9 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(9 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    component FFDr
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,reset:in STD_LOGIC);
    end component;

    component FFDs
        port(D:in  STD_LOGIC;
             Q:buffer  STD_LOGIC;
             clk,set:in STD_LOGIC);
    end component;

    FOR all: adder10 use entity work.adder10(arch1);
    FOR all: FFDr use entity work.FFDr(arch1);
    FOR all: FFDs use entity work.FFDs(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(9 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(9 downto 0);
```

```
    signal ONES: STD_LOGIC_VECTOR(9 downto 0);
    signal ONE: STD_LOGIC;
    signal FFrout, FFsout,Sel_Saida: STD_LOGIC_VECTOR(9 downto 0);
    signal Set, Reset: STD_LOGIC_VECTOR(9 downto 0);
    signal xEOC,xxEOC: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "1111111111";
    ZEROS <= "0000000000";
    Adder: adder10 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);

    FFr0: FFDr port map (SaidaM1(0),FFrout(0),CLK,Reset(0));
    FFr1: FFDr port map (SaidaM1(1),FFrout(1),CLK,Reset(1));
    FFr2: FFDr port map (SaidaM1(2),FFrout(2),CLK,Reset(2));
    FFr3: FFDr port map (SaidaM1(3),FFrout(3),CLK,Reset(3));
    FFr4: FFDr port map (SaidaM1(4),FFrout(4),CLK,Reset(4));
    FFr5: FFDr port map (SaidaM1(5),FFrout(5),CLK,Reset(5));
    FFr6: FFDr port map (SaidaM1(6),FFrout(6),CLK,Reset(6));
    FFr7: FFDr port map (SaidaM1(7),FFrout(7),CLK,Reset(7));
    FFr8: FFDr port map (SaidaM1(8),FFrout(8),CLK,Reset(8));
    FFr9: FFDr port map (SaidaM1(9),FFrout(9),CLK,Reset(9));
    FFs0: FFDs port map (SaidaM1(0),FFsout(0),CLK,Set(0));
    FFs1: FFDs port map (SaidaM1(1),FFsout(1),CLK,Set(1));
    FFs2: FFDs port map (SaidaM1(2),FFsout(2),CLK,Set(2));
    FFs3: FFDs port map (SaidaM1(3),FFsout(3),CLK,Set(3));
    FFs4: FFDs port map (SaidaM1(4),FFsout(4),CLK,Set(4));
    FFs5: FFDs port map (SaidaM1(5),FFsout(5),CLK,Set(5));
    FFs6: FFDs port map (SaidaM1(6),FFsout(6),CLK,Set(6));
    FFs7: FFDs port map (SaidaM1(7),FFsout(7),CLK,Set(7));
    FFs8: FFDs port map (SaidaM1(8),FFsout(8),CLK,Set(8));
    FFs9: FFDs port map (SaidaM1(9),FFsout(9),CLK,Set(9));

    process(Load,DataIn)
    begin
        for I in 0 to 9 loop
            Set(I)<= DataIn(I) and Load;
            Reset(I)<= (not DataIn(I)) and Load;
            Sel_Saida(I)<=DataIn(I);
        end loop;
    end process;

    process(Sel_Saida,FFrout,FFsout)
    begin
        for I in 0 to 9 loop
            if Sel_Saida(I)='0' then
                Saida(I)<=FFrout(I);
            else
                Saida(I)<=FFsout(I);
            end if;
        end loop;
    end process;

    FFEOC: FFDr port map (ONE, xxEOC, xEOC, LOAD);
    EOC <= xxEOC;
    process(Saida,Modulo)
    begin
        if (Saida=Modulo) then
            xEOC <='1';
        else
            xEOC <='0';
        end if;
    end process;
end arch1;
```

F.2.3.13 Ficheiro "Counter16.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Counter16.vhd"                                                   *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 16 bits com    *
--           RESET assincrono.                                                *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter16 is
    port(CLK,RST: in STD_LOGIC;
         Saida: buffer STD_LOGIC_VECTOR(15 downto 0));
end Counter16;

architecture arch1 of Counter16 is
    component adder16
        port(in1,in2: in STD_LOGIC_VECTOR(15 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(15 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder16 use entity work.adder16(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(15 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(15 downto 0);
    signal ONES: STD_LOGIC_VECTOR(15 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES <= "1111111111111111";
    ZEROS <= "0000000000000000";

    Adder: adder16 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,RST,saida,saidaM1,ZEROS,ONES)
    begin
        if RST='1' then
            saida<= ZEROS;
        elsif CLK'event and CLK='1' then
            if saida=ONES then
                saida<=ZEROS;
            else
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.14 Ficheiro "Counter5.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Counter5.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 9 bits com    *
--           RESET assincrono e sinalizacao de fim de contagem.               *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter5 is
    port(CLK,RST: in STD_LOGIC;
         Saida: buffer STD_LOGIC_VECTOR(4 downto 0);
         Modulo: in STD_LOGIC_VECTOR(4 downto 0);
         nEOC: out STD_LOGIC);
end Counter5;

architecture arch1 of Counter5 is
    component adder5
        port(in1,in2: in STD_LOGIC_VECTOR(4 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(4 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder5 use entity work.adder5(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(4 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(4 downto 0);
    signal ONES: STD_LOGIC_VECTOR(4 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "11111";
    ZEROS <= "00000";

    Adder: adder5 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,RST,Modulo,saida,saidaM1,ZEROS,ONES)
        begin
            if RST='1' then
                saida<= ZEROS;
                nEOC<='1';
            elsif CLK'event and CLK='1' then
                if saida=ONES then
                    saida<=ZEROS;
                elsif saida=Modulo then
                    saida<=ZEROS;
                    nEOC<='1';
                else
                    if saidaM1=Modulo then
                    nEOC<='0';
                end if;
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.15 Ficheiro "Counter6.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Counter6.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 6 bits com SET *
--           assincrono e com sinalizacao de fim de contagem.                 *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter6 is
    port(CLK,SET: in STD_LOGIC;
         Saida: buffer STD_LOGIC_VECTOR(5 downto 0);
         TC: out STD_LOGIC);
end Counter6;

architecture arch1 of Counter6 is
    component adder6
        port(in1,in2: in STD_LOGIC_VECTOR(5 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(5 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder6 use entity work.adder6(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(5 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(5 downto 0);
    signal ONES: STD_LOGIC_VECTOR(5 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "111111";
    ZEROS <= "000000";

    Adder: adder6 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,SET,saida,saidaM1,ZEROS,ONES)
    begin
        if SET='1' then
            saida<= ONES;
            TC<='0';
        elsif CLK'event and CLK='1' then
            if saida=ONES then
                saida<=ZEROS;
                TC<='0';
            else
                if saidaM1=ONES then
                    TC<='1';
                end if;
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.16 Ficheiro "Counter7.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Counter7.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 7 bits com SET *
--           assincrono.                                                      *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter7 is
    port(CLK,SET: in STD_LOGIC;
         Saida: buffer STD_LOGIC_VECTOR(6 downto 0));
end Counter7;

architecture arch1 of Counter7 is
    component adder7
        port(in1,in2: in STD_LOGIC_VECTOR(6 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(6 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder7 use entity work.adder7(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(6 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(6 downto 0);
    signal ONES: STD_LOGIC_VECTOR(6 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "1111111";
    ZEROS <= "0000000";

    Adder: adder7 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,SET,saida,saidaM1,ZEROS,ONES)
    begin
        if SET='1' then
            saida<= ONES;
        elsif CLK'event and CLK='1' then
            if saida=ONES then
                saida<=ZEROS;
            else
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.17 Ficheiro "Counter8.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "Counter8.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 8 bits com RST *
--           assincrono com sinalizacao de fim de contagem.                     *
--*******************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter8 is
    port(CLK,RST: in STD_LOGIC;
         Saida: buffer STD_LOGIC_VECTOR(7 downto 0);
         Modulo: in STD_LOGIC_VECTOR(7 downto 0);
         EOC: out STD_LOGIC);
end Counter8;

architecture arch1 of Counter8 is
    component adder8
        port(in1,in2: in STD_LOGIC_VECTOR(7 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(7 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder8 use entity work.adder8(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(7 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(7 downto 0);
    signal ONES: STD_LOGIC_VECTOR(7 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "11111111";
    ZEROS <= "00000000";

    Adder: adder8 port map(Saida,ZEROS,ONE,SaidaM1,CarryOut);
    process(CLK,RST,Modulo,saida,saidaM1,ZEROS,ONES)
    begin
        if RST='1' then
            saida<= ZEROS;
            EOC<='0';
        elsif CLK'event and CLK='1' then
            if saida=ONES then
                saida<=ZEROS;
            elsif saida=Modulo then
                saida<=ZEROS;
                EOC<='0';
            else
                if saidaM1=Modulo then
                    EOC<='1';
                end if;
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.18 Ficheiro "Counter9.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "Counter9.vhd"                                                    *
--DESCRIÇÃO: Este ficheiro contem a descricao de um contador de 9 bits com SET *
--           assincrono.                                                      *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Counter9 is
    port(CLK,SET: in STD_LOGIC;
         Saida: buffer STD_LOGIC_VECTOR(8 downto 0));
end Counter9;

architecture arch1 of Counter9 is
    component adder9
        port(in1,in2: in STD_LOGIC_VECTOR(8 downto 0);
             c0:in STD_LOGIC;
             s: out STD_LOGIC_VECTOR(8 downto 0);
             flag_carry: out STD_LOGIC);
    end component;

    FOR all: adder9 use entity work.adder9(arch1);

    signal CarryOut: STD_LOGIC;
    signal SaidaM1: STD_LOGIC_VECTOR(8 downto 0);
    signal ZEROS: STD_LOGIC_VECTOR(8 downto 0);
    signal ONES: STD_LOGIC_VECTOR(8 downto 0);
    signal ONE: STD_LOGIC;

begin
    ONE <='1';
    ONES  <= "111111111";
    ZEROS <= "000000000";

    Adder: adder9 port map(Saida, ZEROS, ONE, SaidaM1, CarryOut);
    process(CLK, SET, saida, saidaM1, ZEROS, ONES)
    begin
        if SET='1' then
            saida<= ONES;
        elsif CLK'event and CLK='1' then
            if saida=ONES then
                saida<=ZEROS;
            else
                saida<= saidaM1;
            end if;
        end if;
    end process;
end arch1;
```

F.2.3.19 Ficheiro "BufTB16.vhd"

```
--*******************************************************************************
--                           TRABALHO FINAL DE CURSO                           *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "BufTB16.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state com portos*
--           de 16 bits.                                                       *
--*******************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTB16 is
    port(DataIn: in STD_LOGIC_VECTOR(15 downto 0);
         DataOut: out STD_LOGIC_VECTOR(15 downto 0);
         T: in STD_LOGIC);
end BufTB16;

architecture arch1 of BufTB16 is
begin
    process(DataIn,T)
    begin
        if T='0' then
            DataOut<= "ZZZZZZZZZZZZZZZZ";
        else
            DataOut<= DataIn;
        end if;
    end process;
end arch1;
```

F.2.3.20 Ficheiro "BufTBid.vhd"

```
--*******************************************************************************
--                           TRABALHO FINAL DE CURSO                           *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "BufTBid.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state            *
--           bidireccional com 2 portos.                                       *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTBid is
    port(T12,T21:in STD_LOGIC;
         entrada_saida1,entrada_saida2: inout STD_LOGIC);
end BufTBid;

architecture arch1 of BufTBid is
begin
    process(T12,T21,entrada_saida1,entrada_saida2)
    begin
        if(T12='0') then
            entrada_saida2<='Z';
        else
            entrada_saida2<=entrada_saida1;
        end if;
```

```
        if(T21='0') then
            entrada_saida1<='Z';
        else
            entrada_saida1<=entrada_saida2;
        end if;
    end process;
end arch1;
```

F.2.3.21 Ficheiro "BufTBid16.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                              *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*******************************************************************************
--FICHEIRO: "BufTBid16.vhd"                                                     *
--DESCRIÇÃO: Este ficheiro contem a descricao de um buffer tri-state            *
--           bidireccional com 2 portos de 16 bits                              *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity BufTBid16 is
    port(T12,T21:in STD_LOGIC;
         entrada_saida1,entrada_saida2: inout STD_LOGIC_VECTOR(15 downto 0));
end BufTBid16;

architecture arch1 of BufTBid16 is
begin
    process(T12,T21,entrada_saida1,entrada_saida2)
    begin
        if(T12='0') then
            entrada_saida2<="ZZZZZZZZZZZZZZZZ";
        else
            entrada_saida2<=entrada_saida1;
        end if;

        if(T21='0') then
            entrada_saida1<="ZZZZZZZZZZZZZZZZ";
        else
            entrada_saida1<=entrada_saida2;
        end if;
    end process;
end arch1;
```

F.2.3.22 Ficheiro "FFDr.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                              *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*******************************************************************************
--FICHEIRO: "FFDr.vhd"                                                          *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop tipo D com RESET *
--           assincrono.                                                        *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on
```

```
entity FFDr is
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,reset:in STD_LOGIC);
end FFDr;

architecture arch1 of FFDr is

begin
    process(D,clk,reset)
    begin
        if reset='1' then
            Q<='0';
        elsif clk'event and clk='1' then
            Q<= D;
        end if;
    end process;
end arch1;
```

F.2.3.23 Ficheiro "FFDs.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                              *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*******************************************************************************
--FICHEIRO: "FFDs.vhd"                                                          *
--DESCRIÇÃO: Este ficheiro contem a descricao de um Flip_Flop tipo D com SET   *
--           assincrono.                                                        *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity FFDs is
    port(D:in  STD_LOGIC;
         Q:buffer  STD_LOGIC;
         clk,set:in STD_LOGIC);
end FFDs;

architecture arch1 of FFDs is
begin
    process(D,clk,set)
    begin
        if set='1' then
            Q<='1';
        elsif clk'event and clk='1' then
            Q<= D;
        end if;
    end process;
end arch1;
```

F.2.3.24 Ficheiro "Flip_Flop16.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                              *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                          *
--*******************************************************************************
--FICHEIRO: "Flip_Flop16.vhd"                                                   *
--DESCRIÇÃO: Este ficheiro contem a descricao de um registo de 16 bits formado *
--           por 16 Flip_Flops tipo D.                                          *
--*******************************************************************************
--use work.all;
library xc4000;
--use x4000.all;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity Flip_Flop16 is
    port(portoD:in  STD_LOGIC_VECTOR(15 downto 0);
         portoQ:buffer STD_LOGIC_VECTOR(15 downto 0);
         CLK,CLR:in STD_LOGIC);
end Flip_Flop16;

architecture arch1 of Flip_Flop16 is
begin
    process(CLK,CLR,portoD)
    begin
        if CLR='1' then
            portoQ<="0000000000000000";
        elsif CLK'event and CLK='1' then
            portoQ<= portoD;
        end if;
    end process;
end arch1;
```

F.2.3.25 Ficheiro "adder10.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*****************************************************************************
--FICHEIRO: "adder10.vhd"                                                     *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 10 bits.                                                      *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder10 is
    port(in1,in2: in STD_LOGIC_VECTOR(9 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(9 downto 0);
         flag_carry: out STD_LOGIC);
end adder10;

architecture arch1 of adder10 is

    signal p,g: STD_LOGIC_VECTOR(9 downto 0);
    signal c: STD_LOGIC_VECTOR(10 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(9 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));
```

```
    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    c(9) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)
and p(8)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8))
or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(5) and p(6) and p(7) and p(8)) or
        (g(6) and p(7) and p(8)) or
        (g(7) and p(8)) or
        g(8));

    c(10) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9))
or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(6) and p(7) and p(8) and p(9)) or
        (g(7) and p(8) and p(9)) or
        (g(8) and p(9)) or
        g(9));

    flag_carry <= c(10);
end arch1;
```

F.2.3.26 Ficheiro "adder16.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "adder16.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 16 bits.                                                       *
--*******************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder16 is
    port(in1,in2: in STD_LOGIC_VECTOR(15 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(15 downto 0);
         flag_carry: out STD_LOGIC);
end adder16;

architecture arch1 of adder16 is

    signal p,g: STD_LOGIC_VECTOR(15 downto 0);
    signal c: STD_LOGIC_VECTOR(16 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(15 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
```

```
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    c(9) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)
and p(8)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8))
or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(5) and p(6) and p(7) and p(8)) or
        (g(6) and p(7) and p(8)) or
        (g(7) and p(8)) or
        g(8));

    c(10) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9))
or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(6) and p(7) and p(8) and p(9)) or
        (g(7) and p(8) and p(9)) or
        (g(8) and p(9)) or
        g(9));

    c(11) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10))
or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(7) and p(8) and p(9) and p(10)) or
        (g(8) and p(9) and p(10)) or
        (g(9) and p(10)) or
        g(10));

    c(12) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(8) and p(9) and p(10) and p(11)) or
        (g(9) and p(10) and p(11)) or
        (g(10) and p(11)) or
        g(11));
```

```
    c(13) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(9) and p(10) and p(11) and p(12)) or
        (g(10) and p(11) and p(12)) or
        (g(11) and p(12)) or
        g(12));

    c(14) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(10) and p(11) and p(12) and p(13)) or
        (g(11) and p(12) and p(13)) or
        (g(12) and p(13)) or
        g(13));

    c(15) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) ) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13) and p(14)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13) and p(14)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(11) and p(12) and p(13) and p(14)) or
        (g(12) and p(13) and p(14)) or
        (g(13) and p(14)) or
        g(14));

    c(16) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and
p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
```

```
          (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14) and p(15)) or
          (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14) and p(15)) or
          (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14) and p(15)) or
          (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and
p(15)) or
          (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15))
or
          (g(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(12) and p(13) and p(14) and p(15)) or
          (g(13) and p(14) and p(15)) or
          (g(14) and p(15)) or
          g(15));

    flag_carry <= c(16);
end arch1;
```

F.2.3.27 Ficheiro "adder18.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "adder18.vhd"                                                     *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 18 bits.                                                      *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder18 is
    port(in1,in2: in STD_LOGIC_VECTOR(17 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(17 downto 0);
         flag_carry: out STD_LOGIC);
end adder18;

architecture arch1 of adder18 is


    signal p,g: STD_LOGIC_VECTOR(17 downto 0);
    signal c: STD_LOGIC_VECTOR(18 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(17 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
```

```
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    c(9) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)
and p(8)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8))
or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(5) and p(6) and p(7) and p(8)) or
        (g(6) and p(7) and p(8)) or
        (g(7) and p(8)) or
        g(8));

    c(10) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9))
or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(5) and p(6) and p(7) and p(8) and p(9)) or
        (g(6) and p(7) and p(8) and p(9)) or
        (g(7) and p(8) and p(9)) or
        (g(8) and p(9)) or
        g(9));

    c(11) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10))
or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10)) or
```

```
        (g(6) and p(7) and p(8) and p(9) and p(10)) or
        (g(7) and p(8) and p(9) and p(10)) or
        (g(8) and p(9) and p(10)) or
        (g(9) and p(10)) or
        g(10));

    c(12) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(7) and p(8) and p(9) and p(10) and p(11)) or
        (g(8) and p(9) and p(10) and p(11)) or
        (g(9) and p(10) and p(11)) or
        (g(10) and p(11)) or
        g(11));

    c(13) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(8) and p(9) and p(10) and p(11) and p(12)) or
        (g(9) and p(10) and p(11) and p(12)) or
        (g(10) and p(11) and p(12)) or
        (g(11) and p(12)) or
        g(12));

    c(14) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13)) or
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(9) and p(10) and p(11) and p(12) and p(13)) or
        (g(10) and p(11) and p(12) and p(13)) or
        (g(11) and p(12) and p(13)) or
        (g(12) and p(13)) or
        g(13));

    c(15) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) ) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10)
and p(11) and p(12) and p(13) and p(14)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11)
and p(12) and p(13) and p(14)) or
```

```
        (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14)) or
        (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14)) or
        (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14)) or
        (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(9) and p(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(10) and p(11) and p(12) and p(13) and p(14)) or
        (g(11) and p(12) and p(13) and p(14)) or
        (g(12) and p(13) and p(14)) or
        (g(13) and p(14)) or
        g(14));

    c(16) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and
p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14) and p(15)) or
          (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14) and p(15)) or
          (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14) and p(15)) or
          (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and
p(15)) or
          (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15))
or
          (g(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(10) and p(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(11) and p(12) and p(13) and p(14) and p(15)) or
          (g(12) and p(13) and p(14) and p(15)) or
          (g(13) and p(14) and p(15)) or
          (g(14) and p(15)) or
          g(15));

    c(17) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and
p(16)) or
          (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and
p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11) and p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14) and p(15) and p(16)) or
          (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14) and p(15) and p(16)) or
          (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and
p(15) and p(16)) or
          (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)
and p(16)) or
          (g(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16))
or
          (g(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(11) and p(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(12) and p(13) and p(14) and p(15) and p(16)) or
          (g(13) and p(14) and p(15) and p(16)) or
          (g(14) and p(15) and p(16)) or
          (g(15) and p(16)) or
                  g(16));

    c(18) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and
p(16) and p(17)) or
```

```
          (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)
and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16) and
p(17)) or
          (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9)
and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and
p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(3) and p(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and
p(11) and p(12) and p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(4) and p(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and
p(12) and p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(5) and p(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and
p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(6) and p(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and
p(14) and p(15) and p(16) and p(17)) or
          (g(7) and p(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and
p(15) and p(16) and p(17)) or
          (g(8) and p(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15)
and p(16) and p(17)) or
          (g(9) and p(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16)
and p(17)) or
          (g(10) and p(11) and p(12) and p(13) and p(14) and p(15) and p(16) and p(17))
or
          (g(11) and p(12) and p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(12) and p(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(13) and p(14) and p(15) and p(16) and p(17)) or
          (g(14) and p(15) and p(16) and p(17)) or
          (g(15) and p(16) and p(17)) or
                  (g(16) and p(17)) or
                  g(17));

    flag_carry <= c(18);
end arch1;
```

F.2.3.28 Ficheiro "adder5.vhd"

```
--*******************************************************************************
--                           TRABALHO FINAL DE CURSO                            *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "adder5.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 5 bits.                                                        *
--*******************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder5 is
    port(in1,in2: in STD_LOGIC_VECTOR(4 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(4 downto 0);
         flag_carry: out STD_LOGIC);
end adder5;

architecture arch1 of adder5 is

    signal p,g:STD_LOGIC_VECTOR(4 downto 0);
    signal c: STD_LOGIC_VECTOR(5 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;
    S <=(in1 xor in2) xor c(4 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
```

```
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    flag_carry <= c(5);
end arch1;
```

F.2.3.29 Ficheiro "adder6.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "adder6.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 6 bits.                                                       *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder6 is
    port(in1,in2: in STD_LOGIC_VECTOR(5 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(5 downto 0);
         flag_carry: out STD_LOGIC);
end adder6;

architecture arch1 of adder6 is

    signal p,g:STD_LOGIC_VECTOR(5 downto 0);
    signal c: STD_LOGIC_VECTOR(6 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(5 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));
```

```
    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    flag_carry <= c(6);
end arch1;
```

F.2.3.30 Ficheiro "adder7.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "adder7.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 7 bits.                                                       *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder7 is
    port(in1,in2: in STD_LOGIC_VECTOR(6 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(6 downto 0);
         flag_carry: out STD_LOGIC);
end adder7;

architecture arch1 of adder7 is

    signal p,g:STD_LOGIC_VECTOR(6 downto 0);
    signal c: STD_LOGIC_VECTOR(7 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(6 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));
```

```
    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));


    flag_carry <= c(7);
end arch1;
```

F.2.3.31 Ficheiro "adder8.vhd"

```
--*******************************************************************************
--                          TRABALHO FINAL DE CURSO                             *
--*******************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                         *
--*******************************************************************************
--FICHEIRO: "adder8.vhd"                                                       *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 8 bits.                                                        *
--*******************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on

entity adder8 is
    port(in1,in2: in STD_LOGIC_VECTOR(7 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(7 downto 0);
         flag_carry: out STD_LOGIC);
end adder8;

architecture arch1 of adder8 is

    signal p,g:STD_LOGIC_VECTOR(7 downto 0);
    signal c: STD_LOGIC_VECTOR(8 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(7 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));
```

```
    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));


    flag_carry <= c(8);
end arch1;
```

F.2.3.32 Ficheiro "adder9.vhd"

```
--*****************************************************************************
--                          TRABALHO FINAL DE CURSO                           *
--*****************************************************************************
--AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
--         Nuno Roma        - N. 39921                                        *
--*****************************************************************************
--FICHEIRO: "adder9.vhd"                                                      *
--DESCRIÇÃO: Este ficheiro contem a descricao de um somador 'carry look ahead' *
--           de 9 bits.                                                       *
--*****************************************************************************
library xc4000;
--use x4000.components.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
-- synopsys translate_off
use IEEE.GS_TYPES.sdt_values_t;
-- synopsys translate_on
```

```
entity adder9 is
    port(in1,in2: in STD_LOGIC_VECTOR(8 downto 0);
         c0:in STD_LOGIC;
         s: out STD_LOGIC_VECTOR(8 downto 0);
         flag_carry: out STD_LOGIC);
end adder9;

architecture arch1 of adder9 is

    signal p,g: STD_LOGIC_VECTOR(8 downto 0);
    signal c: STD_LOGIC_VECTOR(9 downto 0);

begin
    p <=in1 or in2;
    g <=in1 and in2;

    S <=(in1 xor in2) xor c(8 downto 0);

    c(0) <=c0;

    c(1) <= ((c0 and p(0)) or g(0));

    c(2) <= ((c0 and p(0) and p(1))
        or (g(0) and p(1))
        or g(1));

    c(3) <= ((c0 and p(0) and p(1) and p(2))
        or (g(0) and p(1) and p(2))
        or (g(1) and p(2))
        or g(2));

    c(4) <= ((c0 and p(0) and p(1) and p(2) and p(3))
        or (g(0) and p(1) and p(2) and p(3))
        or (g(1) and p(2) and p(3))
        or (g(2) and p(3))
        or g(3));

    c(5) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4))
        or (g(0) and p(1) and p(2) and p(3) and p(4))
        or (g(1) and p(2) and p(3) and p(4))
        or (g(2) and p(3) and p(4))
        or (g(3) and p(4))
        or g(4));

    c(6) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(1) and p(2) and p(3) and p(4) and p(5)) or
        (g(2) and p(3) and p(4) and p(5)) or
        (g(3) and p(4) and p(5)) or
        (g(4) and p(5)) or
        g(5));

    c(7) <= ((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(2) and p(3) and p(4) and p(5) and p(6)) or
        (g(3) and p(4) and p(5) and p(6)) or
        (g(4) and p(5) and p(6)) or
        (g(5) and p(6)) or
        g(6));

    c(8) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and
p(7)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(3) and p(4) and p(5) and p(6) and p(7)) or
        (g(4) and p(5) and p(6) and p(7)) or
        (g(5) and p(6) and p(7)) or
        (g(6) and p(7)) or
        g(7));

    c(9) <=((c0 and p(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7)
and p(8)) or
        (g(0) and p(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8))
or
        (g(1) and p(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(2) and p(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(3) and p(4) and p(5) and p(6) and p(7) and p(8)) or
```

```
        (g(4) and p(5) and p(6) and p(7) and p(8)) or
        (g(5) and p(6) and p(7) and p(8)) or
        (g(6) and p(7) and p(8)) or
        (g(7) and p(8)) or
        g(8));

    flag_carry <= c(9);
end arch1;
```

G Circuito de controlo de *buffer* e geração de trama

O esquema geral do circuito de controlo e geração de trama encontra-se apresentado na página seguinte.

Colocar aqui o esquema do buffer do emissor.

H Circuito de controlo de *buffer* e recuperação de trama

O esquema geral do circuito de controlo e recuperação de trama encontra-se apresentado na página seguinte.

Colocar aqui o esquema do buffer do receptor.

I Listagem do código de programação dos microcontroladores utilizados

I.1 MICROCONTROLADOR PIC16F84 DO MÓDULO DE AQUISIÇÃO

	Valor	Obs.
Registo 0	00h	
Registo 1	---	Reservado.
Registo 2	00h	
Registo 3	A0h	Formato de saída: 24 bits YC_RC_B.
Registo 4	00h	Modo de operação normal, com controlo automático de ganho (CAG).
Registo 5	74h	Entrada de vídeo composto, PAL, Cor.
Registo 6	00h	Entrada de relógio no pino XTAL1-IN.
Registo 7	00h	Saídas de *pixel* e controlo activadas.
Registo 8	00h	Ajuste de brilho.
Registo 9	80h	Ajuste de contraste.
Registo 10	80h	Ajuste de saturação.
Registo 11	00h	Ajuste de matiz.
Registo 12	C0h	Byte menos significativo de HCLOCK.
Registo 13	03h	Byte mais significativo de HCLOCK.
Registo 14	51h	Byte menos significativo de HDELAY.
Registo 15	00h	Byte mais significativo de HDELAY.
Registo 16	C0h	Byte menos significativo de ACTIVE_PIXELS.
Registo 17	02h	Byte mais significativo de ACTIVE_PIXELS.
Registo 18	23h	Byte menos significativo de VDELAY.
Registo 19	00h	Byte mais significativo de VDELAY.
Registo 20	40h	Byte menos significativo de ACTIVE_LINES.
Registo 21	02h	Byte mais significativo de ACTIVE_LINES.
Registo 22	B2h	Frequência de sub-portadora P0.
Registo 23	EAh	Frequência de sub-portadora P1.
Registo 24	12h	Frequência de sub-portadora P2.
Registo 25	6Dh	Atraso do CAG.
Registo 26	64h	Atraso do *Burst*.
Registo 27	A3h	Byte menos significativo de SampleRateConversion.
Registo 28	1Bh	Byte mais significativo de SampleRateConversion.
Registo 29	30h	Polaridade dos sinais VALID e ACTIVE: *High*.

Tabela I.1 - Registos de programação do conversor A/D Bt812.

```
;*******************************************************************************
;                            TRABALHO FINAL DE CURSO                           *
;*******************************************************************************
; AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
;          Nuno Roma        - N. 39921                                         *
;*******************************************************************************
; FICHEIRO: "bt812.asm"                                                        *
; DESCRIÇÃO: Programa para Inicializacao do Descodificador de Video BT812      *
;            atraves do PIC 16F84                                              *
;*******************************************************************************

; ************************************************************
; Definicao das constantes a programar o BT812
; ************************************************************
reg2    EQU     H'00'   ; Status
reg3    EQU     H'A0'   ; Output Format Definition
reg4    EQU     H'00'   ; Operation Mode Select -->> H'40' com geração interna de
barras
reg5    EQU     H'74'   ; Input Format Definition (COLOR_ON)
reg6    EQU     H'00'   ; Clock Definition
reg7    EQU     H'00'   ; Video Timing Definition
reg8    EQU     H'00'   ; Brightness Adjust
reg9    EQU     H'80'   ; Contrast Adjust
reg10   EQU     H'80'   ; Saturation Adjust
reg11   EQU     H'00'   ; Hue Adjust
reg12   EQU     H'C0'   ; HClock Low
reg13   EQU     H'03'   ; HClock High
reg14   EQU     H'51'   ; HDelay Low
reg15   EQU     H'00'   ; HDelay High
reg16   EQU     H'C0'   ; Active_Pixels Low
reg17   EQU     H'02'   ; Active_Pixels High
reg18   EQU     H'23'   ; VDelay Low
reg19   EQU     H'00'   ; VDelay High
reg20   EQU     H'40'   ; Active Lines Low
reg21   EQU     H'02'   ; Active Lines High
reg22   EQU     H'B2'   ; P0 (Subcarrier Frequency)
reg23   EQU     H'EA'   ; P1 (Subcarrier Frequency)
reg24   EQU     H'12'   ; P2 (Subcarrier Frequency)
reg25   EQU     H'6D'   ; AGC Delay
reg26   EQU     H'64'   ; Burst Delay
reg27   EQU     H'A3'   ; Sample Rate Conversion Low
reg28   EQU     H'1B'   ; Sample Rate Conversion High
reg29   EQU     H'30'   ; Polarity

PORTA   EQU     H'0005'
PORTB   EQU     H'0006'
TRISA   EQU     H'0005'
TRISB   EQU     H'0006'
STATUS  EQU     H'0003'
RP0     EQU     H'0005'

        LIST    p=16F84
    __config    H'3FFA'

        org 0
        goto inicio
        org 0x20

; ************************************************************
; Definicao de Macros para escrita nos pinos de saida
; ************************************************************
; Correspondencia entre pinos do PIC16F84 e pinos do BT812:
; RB<0-7> <--> D<0-7>
; RA0     <--> WR*
; RA1     <--> RS1
; RA2     <--> RD*
; RA3     <--> Color
; RA4     <--> YCrCb

    variable D = 0 , WR_ = 1 , RS1 = 0 , RD_ = 1
    variable Inicio = 0, Fim = 0, endereco = 0

; Define PORT_A como porto de saida, excepto RA3/Color e RA4/YCrCb
PA_SAIDA    macro
    BCF     STATUS,RP0  ; selecciona Banco 0
    CLRF    PORTA       ; inicializa PORTA
    BSF     STATUS,RP0  ; selecciona Banco 1
    MOVLW   0x18        ; coloca os bits RA<2:0> como saidas
```

```
    MOVWF   TRISA       ; e os bit RA3 e RA4 como entradas
    BCF     STATUS,RP0  ; selecciona Banco 0
    endm

; Define PORT_B como porto de saida
PB_SAIDA    macro
    BCF     STATUS,RP0  ; selecciona Banco 0
    CLRF    PORTB       ; inicializa PORTA
    BSF     STATUS,RP0  ; selecciona Banco 1
    CLRF    TRISB       ; coloca os bits RB<7:0> como saidas
    BCF     STATUS,RP0  ; selecciona Banco 0
    endm

; Define PORT_B como porto de entrada
PB_ENTRADA  macro
    BCF     STATUS,RP0  ; selecciona Banco 0
    CLRF    PORTB       ; inicializa PORTA
    BSF     STATUS,RP0  ; selecciona Banco 1
    MOVLW   0xFF
    MOVWF   TRISB       ; coloca os bits RB<7:0> como saidas
    BCF     STATUS,RP0  ; selecciona Banco 0
    endm

; Escreve Dados no Porto B, que deve estar definido como saida
DADOS macro
    MOVLW   D
    MOVWF   PORTB
    endm

; Escreve dados de controlo no Porto A, que deve estar definido como saida
CONTROLO macro
    local   RA
RA = RD_ * 4 + RS1 * 2 + WR_
    MOVLW   RA
    MOVWF   PORTA
    endm

; Aplica dados no Porto B e efectua um ciclo de escrita com 0--1 no sinal /WR
ESCREVE macro
WR_ = 0
    CONTROLO
    DADOS
WR_ = 1
    CONTROLO
    endm

; Define endereco inicial para acesso aos registos
ENDERECO macro
RS1 = 0
WR_ = 1
    CONTROLO
    ESCREVE
RS1 = 1
    CONTROLO
    endm

; Realiza ciclos de escrita entre as posicoes Inicio e Fim
ESCREVE_BLOCO macro
D = Inicio
    ENDERECO
endereco = Inicio
    while endereco <= Fim
D = reg#v(endereco)
    ESCREVE
endereco +=1
    endw
    endm

; Efectua um ciclo de leitura, com 0--1 no sinal /RD
LE macro
RD_ = 0
    CONTROLO
RD_ = 1
    CONTROLO
    endm

; Realiza ciclos de leitura entre as posicoes Inicio e Fim
LE_BLOCO macro
D = Inicio
```

```
    ENDERECO
    PB_ENTRADA
endereco = Inicio
    while endereco <= Fim
    LE
endereco +=1
    endw
    endm

; ************************************************************
; Inicio do Programa
; ************************************************************
inicio
; [...]
    PA_SAIDA    ; define Porto A como saida para controlo MPU,
                ; excepto RA3/Color como entrada
    PB_SAIDA    ; define Porto B como saida para escrita nos registos

; Escrita de dados nos 29 registos de controlo do BT812
Inicio = 2
Fim = D'29'
    ESCREVE_BLOCO

; Leitura de dados nos 29 registos de controlo do BT812
leitura
Inicio = D'00'
Fim = D'29'
    LE_BLOCO
    goto    fim

; Fim do programa, com entrada em modo de espera (Standby Mode)
fim
    PB_SAIDA
D = D'00'
    DADOS
WR_ = 1
RS1 = 0
RD_ = 1
    CONTROLO
    SLEEP
    END
```

I.2 MICROCONTROLADOR PIC16F84 DO MÓDULO DE VISUALIZAÇÃO

	Valor	Obs.
Registo 0	90h	24-bit YCrCb.
Registo 1	0Ch	
Registo 2	F0h	Operação normal.
Registo 3	80h	Modo 3 de temporização, formato PAL, cor, limitação dos valores de vídeo activada.
Registo 4	28h	
Registo 5	--	Reservado.
Registo 6	9Ah	Byte menos significativo de P1.
Registo 7	02h	Byte mais significativo de P1.
Registo 8	CDh	Byte menos significativo de P2.
Registo 9	05h	Byte mais significativo de P2.
Registo 10	00h	Ajuste da fase da sub-portadora de cor (Byte menos significativo).
Registo 11	00h	Ajuste da fase da sub-portadora de cor (Byte mais significativo).
Registo 12	68h	Byte menos significativo de HCOUNT.
Registo 13	03h	Byte mais significativo de HCOUNT.

Tabela I.2 - Registos de programação do conversor D/A Bt858.

```
;*******************************************************************************
;                            TRABALHO FINAL DE CURSO                            *
;*******************************************************************************
; AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
;          Nuno Roma        - N. 39921                                          *
;*******************************************************************************
; FICHEIRO: "bt858.asm"                                                        *
; DESCRIÇÃO: Programa para Inicializacao do Codificador de Video BT858         *
;            atraves do PIC 16F84                                              *
;*******************************************************************************

; ************************************************************
; Definicao das constantes a programar o BT858
; ************************************************************

#define aanr
    ifdef   aanr
reg0    EQU     H'90'   ; 24-bit YCrCb
reg1    EQU     H'0C'
reg2    EQU     H'F0'   ; Normal operation, normal video
reg3    EQU     H'80'   ; PAL Timing Mode 3
reg4    EQU     H'28'   ; Clock_Out output disabled, Control output enabled, FIELD
output
reg5    EQU     H'00'   ; reserved
reg6    EQU     H'9B'   ; P1 low register
reg7    EQU     H'02'   ; P1 high register
reg8    EQU     H'D0'   ; P2 low register
reg9    EQU     H'07'   ; P2 high register
reg10   EQU     H'00'   ; Fsc phase adjust low register
reg11   EQU     H'00'   ; Fsc phase adjust high register
reg12   EQU     H'66'   ; HCOUNT low register
reg13   EQU     H'03'   ; HCOUNT high register
    endif

    ifdef   mira
reg0    EQU     H'00'   ; 24-bit RGB
reg1    EQU     H'0C'
reg2    EQU     H'F0'   ; Normal operation, normal video
reg3    EQU     H'84'   ; PAL Timing Mode 3 ; Generate Color Bars
reg4    EQU     H'28'   ; Clock_Out output disabled, Control output enabled, FIELD
output
```

```
reg5    EQU     H'00'   ; reserved
reg6    EQU     H'9B'   ; P1 low register
reg7    EQU     H'02'   ; P1 high register
reg8    EQU     H'D5'   ; P2 low register
reg9    EQU     H'07'   ; P2 high register
reg10   EQU     H'00'   ; Fsc phase adjust low register
reg11   EQU     H'00'   ; Fsc phase adjust high register
reg12   EQU     H'68'   ; HCOUNT low register
reg13   EQU     H'03'   ; HCOUNT high register
    endif

PORTA   EQU H'0005'
PORTB   EQU H'0006'
TRISA   EQU H'0005'
TRISB   EQU H'0006'
STATUS  EQU H'0003'
RP0     EQU H'0005'

    LIST    p=16F84
    __config    H'3FFA'

    org 0
    goto inicio
    org 0x20


; ************************************************************
; Definicao de Macros
; ************************************************************
; Correspondencia entre pinos do PIC16F84 e pinos do BT858:
; RB<0-7> <--> D<0-7>
; RA0     <--> WR*
; RA1     <--> RS1
; RA2     <--> RD*
; RA3     <--> Color
; RA4     <--> YCrCb

    variable D = 0 , WR_ = 1 , RS1 = 0 , RD_ = 1
    variable Inicio = 0, Fim = 0, endereco = 0
    variable Test_Color = 0, Test_YCrCb = 0, Test_HCOUNT = 0

; Define PORT_A como porto de saida, excepto RA3/Color e RA4/YCrCb
PA_SAIDA macro
    BCF     STATUS,RP0  ; selecciona Banco 0
    CLRF    PORTA       ; inicializa PORTA
    BSF     STATUS,RP0  ; selecciona Banco 1
    MOVLW   0x18        ; coloca os bits RA<2:0> como saidas
    MOVWF   TRISA       ; e os bit RA3 e RA4 como entradas
    BCF     STATUS,RP0  ; selecciona Banco 0
    endm

; Define PORT_B como porto de saida
PB_SAIDA macro
    BCF     STATUS,RP0  ; selecciona Banco 0
    CLRF    PORTB       ; inicializa PORTA
    BSF     STATUS,RP0  ; selecciona Banco 1
    CLRF    TRISB       ; coloca os bits RB<7:0> como saidas
    BCF     STATUS,RP0  ; selecciona Banco 0
    endm

; Define PORT_B como porto de entrada
PB_ENTRADA macro
    BCF     STATUS,RP0  ; selecciona Banco 0
    CLRF    PORTB       ; inicializa PORTA
    BSF     STATUS,RP0  ; selecciona Banco 1
    MOVLW   0xFF
    MOVWF   TRISB       ; coloca os bits RB<7:0> como saidas
    BCF     STATUS,RP0  ; selecciona Banco 0
    endm

; Escreve Dados no Porto B, que deve estar definido como saida
DADOS macro
    MOVLW   D
    MOVWF   PORTB
    endm

; Escreve dados de controlo no Porto A, que deve estar definido como saida,
; excepto RA3/Color para activacao da cor e RA4/YCrCb para escolher YCrCb ou RGB
CONTROLO    macro
```

```
    local   RA
RA = RD_ * 4 + RS1 * 2 + WR_
    MOVLW   RA
    MOVWF   PORTA
    endm

; Aplica dados no Porto B e efectua um ciclo de escrita com 0--1 no sinal /WR
ESCREVE macro
WR_ = 0
    CONTROLO
    DADOS
WR_ = 1
    CONTROLO
    endm

; Define endereco inicial para acesso aos registos
ENDERECO macro
RS1 = 0
WR_ = 1
    CONTROLO
    ESCREVE
RS1 = 1
    CONTROLO
    endm

; Realiza ciclos de escrita entre as posicoes Inicio e Fim
ESCREVE_BLOCO   macro
D   = Inicio
    ENDERECO
endereco = Inicio
    while endereco <= Fim
D   = reg#v(endereco)
Test_Color  = 0
Test_YCrCb  = 0
Test_HCOUNT = 0
    if  endereco == D'00'
Test_YCrCb = 1
    endif
    if  endereco == D'03'
Test_Color = 1
    endif
    if  endereco == D'12'
Test_HCOUNT = 1
    endif
    ESCREVE
endereco +=1
    endw
    endm

; Efectua um ciclo de leitura, com 0--1 no sinal /RD
LE macro
RD_ = 0
    CONTROLO
RD_ = 1
    CONTROLO
    endm

; Realiza ciclos de leitura entre as posicoes Inicio e Fim
LE_BLOCO macro
D   = Inicio
    ENDERECO
    PB_ENTRADA
endereco = Inicio
    while endereco <= Fim
    LE
endereco +=1
    endw
    endm

; ************************************************************
; Inicio do Programa
; ************************************************************
inicio
; [...]
    PA_SAIDA    ; define Porto A como saida para controlo MPU,
                ; excepto RA3/Color como entrada
    PB_SAIDA    ; define Porto B como saida para escrita nos registos

; Escrita de dados nos 13 registos de controlo do BT858
```

```
Inicio = 0
Fim = D'13'
    ESCREVE_BLOCO

; Segunda Escrita de dados nos 13 registos de controlo do BT858
Inicio = D'07'
Fim = D'13'
    ESCREVE_BLOCO

; Leitura de dados nos 13 registos de controlo do BT858
leitura
Inicio = D'00'
Fim = D'13'
    LE_BLOCO
    goto fim

; Fim do programa, com entrada em modo de espera (Standby Mode)
fim
    PB_SAIDA
D   = D'00'
    DADOS
WR_ = 1
RS1 = 0
RD_ = 1
    CONTROLO
    SLEEP
    END
```

I.3 MICROCONTROLADOR PIC16F74A DO MÓDULO DE CONTROLO DO *BUFFER* DE EMISSÃO

```
;*******************************************************************************
;                            TRABALHO FINAL DE CURSO                            *
;*******************************************************************************
; AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
;          Nuno Roma        - N. 39921                                         *
;*******************************************************************************
; FICHEIRO: "pic_em.asm"                                                       *
; DESCRIÇÃO: Programa para mirocontrolador na placa de emissao, que controla   *
;            FIFOs de emissão                                                  *
;*******************************************************************************
    list    p=PIC16C74A
    include <p16C74a.inc>

#define nDELAYED
;-------------------------------------------------
;   - PORTA -
; SINAIS
#define CLK_W       .1  ; saida
#define CLK_RA      .2  ; saida
#define CLK_RB      .4  ; saida
#define RESET_RA    .8  ; saida
#define RESET_RB    .16 ; saida
#define G_RESET     .32 ; saida
;
; CONFIGURACAO
#define CFGTRISA    0x00
;-------------------------------------------------
;   - PORTB -
; SINAIS
#define nRESET_FF_RA    .1  ; saida
#define nRESET_FF_WA    .2  ; saida
#define nRESET_FF_RB    .4  ; saida
#define nRESET_FF_WB    .8  ; saida

#define COUNTER_RA      .4  ; entrada (interrupt)
#define COUNTER_WA      .5  ; entrada (interrupt)
#define COUNTER_RB      .6  ; entrada (interrupt)
#define COUNTER_WB      .7  ; entrada (interrupt)

;
; CONFIGURACAO
#define CFGTRISB    0xf0
#define CFGINTCON   0x08    ; enable RB chg intr
                            ; disable RB0 intr
;-------------------------------------------------
;   - PORTC -
; SINAIS
; (constitui o porto de saida do contador UP/DOWN
;  juntamente com o barramento de dados de Q1900)
;
; CONFIGURACAO
#define CFGTRISC    0x00
;-------------------------------------------------
;   - PORTD -
; SINAIS
; (constitui o porto de saida para programacao do Q1900)
#define nWR .32
#define nRD .64
#define nCS .128

; CONFIGURACAO
#define CFGTRISD    0x00
;-------------------------------------------------
;   - PORTE -
; SINAIS
#define SEL_HIGH    .1  ; saida ('0': LOW '1':HIGH)
#define SEL_REG     .2  ; saida ('0': REG_A '1':REG_B)
#define CLK_REG     .4  ; saida
;
; CONFIGURACAO
#define CFGTRISE    0x00    ; mode: general purpose I/O
;
;-------------------------------------------------
CFGADCON0   EQU   0x00  ; so' bit 0 tem significado: desliga A/D
CFGADCON1   EQU 0x0f    ; so' bits 1 e 2 tem significado: colocam
```

```
                        ; portos em I/O digital
CFGOPTION   EQU 0x80    ; disable pull-ups PORTB
CFGPIE1     EQU 0x00
CFGPIE2     EQU 0x00
CFGT1CON    EQU 0x00    ; Timer1 OFF
CFGT2CON    EQU 0x00    ; Timer2 OFF
CFGCCP1CON  EQU 0x00    ; PWM1 OFF
CFGCCP2CON  EQU 0x00    ; PWM2 OFF
CFGSSPCON   EQU 0x00    ; Serial Port OFF
CFGRCSTA    EQU 0X00
CFGTXSTA    EQU 0X00
;-----------------------------------------------------
; Registos para Viterbi CODEC (Q1900) de acordo com
; os 18 passos de programcao indicados no datasheet
; para operacao '3/4 Parallel Viterbi' (pg. 1-39 no
; referido datasheet)

;enderecos de registos a alterar em cada um dos 18 passos
VIT_ADD1    EQU 0x15
VIT_ADD2    EQU 0x16
VIT_ADD3    EQU 0x02
VIT_ADD4    EQU 0x03
VIT_ADD5    EQU 0x04
VIT_ADD6    EQU 0x08
VIT_ADD7    EQU 0x09
VIT_ADD8    EQU 0x0a
VIT_ADD9    EQU 0x0b
VIT_ADD10   EQU 0x0c
VIT_ADD11   EQU 0x17
VIT_ADD12   EQU 0x18
VIT_ADD13   EQU 0x06
VIT_ADD14   EQU 0x07
VIT_ADD15   EQU 0x04
VIT_ADD16   EQU 0x06
VIT_ADD17   EQU 0x04
VIT_ADD18   EQU 0x06

;dados a colocar nos registos em cada um dos 18 passos
VIT_REG1    EQU 0x00
VIT_REG2    EQU 0x00
VIT_REG3    EQU 0x08
VIT_REG4    EQU 0x01
VIT_REG5    EQU 0x01
VIT_REG6    EQU 0xf4
VIT_REG7    EQU 0xf9
VIT_REG8    EQU 0xfc
VIT_REG9    EQU 0xff
VIT_REG10   EQU 0xff
VIT_REG11   EQU 0x00
VIT_REG12   EQU 0x00
VIT_REG13   EQU 0x08
VIT_REG14   EQU 0x00
VIT_REG15   EQU 0x05
VIT_REG16   EQU 0x0a
VIT_REG17   EQU 0x01
VIT_REG18   EQU 0x08

;-----------------------------------------------------
; Macro de programacao do Q1900 (Viterbi Codec)

    variable I=1, J=1, K=0

VIT_WR  macro   step
    ; macro para escrita de registos de programacao do Q1900
    movlw   VIT_REG#v(step)
    movwf   PORTC
    movlw   VIT_ADD#v(step)+nRD+nWR
    movwf   PORTD
    movlw   VIT_ADD#v(step)+nRD
    movwf   PORTD
    movlw   VIT_ADD#v(step)+nRD+nWR
    movwf   PORTD
    endm
;-----------------------------------------------------
; Variaveis
W_ISR           EQU 0x20    ; guarda W em ISR
STATUS_ISR      EQU 0x21    ; guarda STATUS em ISR
COUNTERA_LOW    EQU 0x22
COUNTERA_HIGH   EQU 0x23
```

```
COUNTERB_LOW    EQU 0x24
COUNTERB_HIGH   EQU 0x25

    org 0
    goto    START
    org 4

;-----------------------------------------------------
; Interrupt Service Routine

ISR bcf     STATUS,RP0  ; banco 0
    movwf   W_ISR       ; salvaguarda de contexto
    swapf   STATUS,W    ;   "
    movwf   STATUS_ISR  ;   "
    btfss   INTCON,RBIF ; se int. nao for causado por RB4..7 =>sai
    goto    FIM_ISR
    movf    PORTB,W

TST_UPA btfsc   PORTB,COUNTER_WA
    call    UP_A

TST_DNA btfsc   PORTB,COUNTER_RA
    call    DOWN_A

TST_UPB btfsc   PORTB,COUNTER_WB
    call    UP_B

TST_DNB btfsc   PORTB,COUNTER_RB
    call    DOWN_B

FIM_ISR movf PORTB,W    ; ends mismatch
    andlw   0xf0        ; para verificar se ocorreram mudancas
                        ; nos niveis RB(4..7)
    btfss   STATUS,Z    ; se RB(4..7)estiverem inactivos, sai.
    goto    TST_UPA     ; caso contrario, ...

    swapf   STATUS_ISR,W
    movwf   STATUS
    swapf   W_ISR,F
    swapf   W_ISR,W

    bcf     INTCON,RBIF
    retfie
;-----------------------------------------------------
UP_A btfss  PORTB,COUNTER_WA
    return

    movlw   nRESET_FF_RA+nRESET_FF_WB+nRESET_FF_RB
    movwf   PORTB       ; reset do FF correspondente
    movlw   nRESET_FF_WA+nRESET_FF_RA+nRESET_FF_WB+nRESET_FF_RB
    movwf   PORTB       ; reset do FF correspondente

    ; testa se contador interno == 0
    movf    COUNTERA_LOW,W
    btfss   STATUS,Z
    goto    NOTZ_A      ; COUNTER_LOW != 0
    movf    COUNTERA_HIGH,W
    btfss   STATUS,Z
    goto    NOTZ_A      ; COUNTER_HIGH != 0

    ; caso contador seja 0 e' preciso fazer
    ; reset do ponteiro de leitura no FIFO
    movlw   RESET_RA
    movwf   PORTA       ; => RSTR=1
    movlw   RESET_RA+CLK_RA
    movwf   PORTA       ; ...=> SRCK=1
    movlw   RESET_RA
    movwf   PORTA       ; ...=> SRCK=0
    movlw   .0
    movwf   PORTA       ; => RSTR=0
    incf    COUNTERA_LOW,F
    goto    SEND_A

NOTZ_A  incf    COUNTERA_LOW,F
    btfsc   STATUS,Z
    incf    COUNTERA_HIGH,F

SEND_A
    movf    COUNTERA_HIGH,W
```

```
    movwf   PORTC
    movlw   SEL_HIGH    ;REG_A HIGH
    movwf   PORTE
    movlw   SEL_HIGH+CLK_REG
    movwf   PORTE
    movlw   SEL_HIGH
    movwf   PORTE
    movf    COUNTERA_LOW,W
    movwf   PORTC
    movlw   .0      ;REG_A LOW
    movwf   PORTE
    movlw   CLK_REG
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    return

;-----------------------------------------------------
DOWN_A btfss PORTB,COUNTER_RA
    return

    movlw   nRESET_FF_WA+nRESET_FF_WB+nRESET_FF_RB
    movwf   PORTB
    movlw   nRESET_FF_WA+nRESET_FF_RA+nRESET_FF_WB+nRESET_FF_RB
    movwf   PORTB

    movlw   .1
    subwf   COUNTERA_LOW,F
    btfss   STATUS,C    ; C==0 => COUNTER_LOW==FF => COUNTER_HIGH--
    decf    COUNTERA_HIGH,F
    movf    COUNTERA_HIGH,W
    movwf   PORTC
    movlw   SEL_HIGH    ; REG_A HIGH
    movwf   PORTE
    movlw   SEL_HIGH+CLK_REG
    movwf   PORTE
    movlw   SEL_HIGH
    movwf   PORTE
    movf    COUNTERA_LOW,W
    movwf   PORTC
    movlw   .0          ; REG_A LOW
    movwf   PORTE
    movlw   CLK_REG
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    return

;-----------------------------------------------------
UP_B btfss PORTB,COUNTER_WB
    return

    movlw   nRESET_FF_RB+nRESET_FF_WA+nRESET_FF_RA
    movwf   PORTB       ; reset do FF correspondente
    movlw   nRESET_FF_WB+nRESET_FF_RB+nRESET_FF_WA+nRESET_FF_RA
    movwf   PORTB       ; reset do FF correspondente

    ; testa se contador interno == 0
    movf    COUNTERB_LOW,W
    btfss   STATUS,Z
    goto    NOTZ_B      ; COUNTER_LOW != 0
    movf    COUNTERB_HIGH,W
    btfss   STATUS,Z
    goto    NOTZ_B      ; COUNTER_HIGH != 0

    ; caso contador seja 0 e' preciso fazer
    ; reset do ponteiro de leitura no FIFO
    movlw   RESET_RB
    movwf   PORTA       ; => RSTR=1
    movlw   RESET_RB+CLK_RB
    movwf   PORTA       ; ...=> SRCK=1
    movlw   RESET_RB
    movwf   PORTA       ; ...=> SRCK=0
    movlw   .0
    movwf   PORTA       ; => RSTR=0
    incf    COUNTERB_LOW,F
    goto    SEND_B

NOTZ_B incf COUNTERB_LOW,F
```

```
    btfsc   STATUS,Z
    incf    COUNTERB_HIGH,F

SEND_B
    movf    COUNTERB_HIGH,W
    movwf   PORTC
    movlw   SEL_HIGH+SEL_REG    ;REG_B HIGH
    movwf   PORTE
    movlw   SEL_HIGH+SEL_REG+CLK_REG
    movwf   PORTE
    movlw   SEL_HIGH+SEL_REG
    movwf   PORTE
    movf    COUNTERB_LOW,W
    movwf   PORTC
    movlw   SEL_REG             ;REG_B LOW
    movwf   PORTE
    movlw   SEL_REG+CLK_REG
    movwf   PORTE
    movlw   SEL_REG
    movwf   PORTE
    return

;-----------------------------------------------------
DOWN_B  btfss   PORTB,COUNTER_RB
    return

    movlw   nRESET_FF_WB+nRESET_FF_WA+nRESET_FF_RA
    movwf   PORTB
    movlw   nRESET_FF_WB+nRESET_FF_RB+nRESET_FF_WA+nRESET_FF_RA
    movwf   PORTB

    movlw   .1
    subwf   COUNTERB_LOW,F
    btfss   STATUS,C    ; C==0 => COUNTER_LOW==FF => COUNTER_HIGH--
    decf    COUNTERB_HIGH,F
    movf    COUNTERB_HIGH,W
    movwf   PORTC
    movlw   SEL_REG+SEL_HIGH    ; REG_B HIGH
    movwf   PORTE
    movlw   SEL_REG+SEL_HIGH+CLK_REG
    movwf   PORTE
    movlw   SEL_REG+SEL_HIGH
    movwf   PORTE
    movf    COUNTERB_LOW,W
    movwf   PORTC
    movlw   SEL_REG             ; REG_B LOW
    movwf   PORTE
    movlw   SEL_REG+CLK_REG
    movwf   PORTE
    movlw   SEL_REG
    movwf   PORTE
    return

;-----------------------------------------------------
; Rotina que efectua o atraso necessario antes dos passos
; 17 e 18 de programacao do Codec Q1900
; Assume-se um clock maior que 9000 bps e CLKIN=20MHz
; (20MHz/4)/9000bps * 2cycles => 1120 = 4 * 255 + 100

DELAY
J=1
    while J<=5
    ifdef   DELAYED
    movlw   .255
L#v(J)  addlw   .255    ; decrementa de uma unidade
    btfss   STATUS,Z
    goto    L#v(J)
    endif
J+=1
    endw
    return

;-----------------------------------------------------
; Rotina de inicializacao
CFG bcf     STATUS,RP0 ; banco 0
    movlw   .0
    movwf   PORTA
    movwf   PORTB
    movwf   PORTC
```

```
    movwf   PORTD
    movwf   PORTE
    movlw   CFGADCON0
    movwf   ADCON0
    movlw   CFGINTCON
    movwf   INTCON
    movlw   CFGT1CON
    movwf   T1CON
    movlw   CFGT2CON
    movwf   T2CON
    movlw   CFGCCP1CON
    movwf   CCP1CON
    movlw   CFGCCP2CON
    movwf   CCP2CON
    movlw   CFGSSPCON
    movwf   SSPCON
    movlw   CFGRCSTA
    movwf   RCSTA

    bsf STATUS,RP0 ; banco 1
    movlw   CFGOPTION
    movwf   OPTION_REG
    movlw   CFGPIE1
    movwf   PIE1
    movlw   CFGPIE2
    movwf   PIE2
    movlw   CFGTRISA
    movwf   TRISA       ; define entradas/saidas de porto A
    movlw   CFGTRISB
    movwf   TRISB       ; define entradas/saidas de porto B
    movlw   CFGTRISC
    movwf   TRISC       ; define entradas/saidas de porto C
    movlw   CFGTRISD
    movwf   TRISD       ; define entradas/saidas de porto D
    movlw   CFGTRISE
    movwf   TRISE       ; define entradas/saidas de porto E e slave port
    movlw   CFGADCON1
    movwf   ADCON1
    movlw   CFGTXSTA
    movwf   TXSTA

    bcf STATUS,RP0 ; banco 0

    ;Reset Global
    movlw   G_RESET
    movwf   PORTA

    while   I<=.18
        if I>=.17
            call DELAY; atrasa steps 17 e 18 de acordo c/ datasheet
        endif
        VIT_WR  I   ; executa step I de configuracao Q1900
I+=1
    endw

    ; desactivacao de CS de Q1900 (activo a LOW)
    movlw   nWR+nRD+nCS
    movwf   PORTD

    bcf STATUS,RP0 ; banco 0

    ; inicializacao de contadores internos
    movlw   .0
    movwf   COUNTERA_LOW
    movwf   COUNTERA_HIGH
    movwf   COUNTERB_LOW
    movwf   COUNTERB_HIGH

    ; reset registos de saida do contador
    movwf   PORTC

M=0
    while M<=3
        movlw   CLK_REG+M
        movwf   PORTE
M+=1
    endw

    movlw   .0
```

```
    movwf   PORTE   ; completa ciclo para REG_B HIGH

    ; reset de escrita dos FIFOS
    movlw   G_RESET ; G_RESET==RSTW
    movwf   PORTA
    movlw   G_RESET+CLK_W
    movwf   PORTA
    movlw   G_RESET
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ; fim de reset FFs UP/DOWN
    movlw   nRESET_FF_WA+nRESET_FF_RA+nRESET_FF_WB+nRESET_FF_RB
    movwf   PORTB
    return

;-----------------------------------------------------
; Rotina Principal
START   bcf INTCON,GIE  ; desactiva interrupcoes
    call    CFG
    movf    PORTB,F     ;ends mismatch
    bcf     INTCON,RBIF
    bsf     INTCON,GIE  ; activa interrupcoes

LOOP        NOP
    goto    LOOP
    END
```

I.4 MICROCONTROLADOR PIC16F74A DO MÓDULO DE CONTROLO DO *BUFFER* DE RECEPÇÃO

```
;*******************************************************************************
;                            TRABALHO FINAL DE CURSO                            *
;*******************************************************************************
; AUTORES: Alexandre Abreu  - N. 39775                 ANO LECTIVO: 1997/1998 *
;          Nuno Roma        - N. 39921                                         *
;*******************************************************************************
; FICHEIRO: "pic_rec.asm"                                                      *
; DESCRIÇÃO: Programa para mirocontrolador na placa de recepcao, que controla  *
;            FIFOs de recepcao                                                 *
;*******************************************************************************
    list    p=PIC16C74A
    include <p16C74a.inc>

#define DELAYED
;-----------------------------------------------------
;   - PORTA -
; SINAIS
#define RSTW    .1  ; saida
#define CLK_W   .2  ; saida
#define RSTR    .4  ; saida
#define CLK_R   .8  ; saida
#define RA4     .16 ; NC
#define RA5     .32 ; NC
;
; CONFIGURACAO
#define CFGTRISA    0x10
;-----------------------------------------------------
;   - PORTB -
; SINAIS
#define RST_VITERBI     .1  ; saida
#define FPGA3_RESET     .2  ; saida
#define nFFUP_RESET     .4  ; saida
#define nFFDOWN_RESET   .8  ; saida
#define CTR_UP          .4  ; entrada (interrupt)
#define CTR_DOWN        .5  ; entrada (interrupt)
#define RST_FIFO_FLAG   .6  ; entrada (interrupt)
#define RB7             .128    ; NC
;
; CONFIGURACAO
#define CFGTRISB    0xf0
#define CFGINTCON   0x08    ; enable RB chg intr
                            ; disable RB0 intr
;-----------------------------------------------------
;   - PORTC -
; SINAIS
; (constitui o porto de saida do contador UP/DOWN
;  juntamente com o barramento de dados do Q1900)
;
; CONFIGURACAO
#define CFGTRISC    0x00
;-----------------------------------------------------
;   - PORTD -
; SINAIS
; (constitui o porto de saida para programacao do Q1900)
#define nCS .32
#define nRD .64
#define nWR .128
;
; CONFIGURACAO
#define CFGTRISD    0x00
;-----------------------------------------------------
;   - PORTE -
; SINAIS
#define CLK_LOW     .1  ; saida
#define CLK_HIGH    .2  ; saida
#define RE2         .4  ; NC
;
; CONFIGURACAO
#define CFGTRISE    0x04    ; mode: general purpose I/O
;
;-----------------------------------------------------
CFGADCON0   EQU 0x00    ; so' bit 0 tem significado: desliga A/D
CFGADCON1   EQU 0x0f    ; so' bits 1 e 2 tem significado: colocam
                        ; portos em I/O digital
CFGOPTION   EQU 0x80    ; disable pull-ups PORTB
```

```
CFGPIE1     EQU 0x00
CFGPIE2     EQU 0x00
CFGT1CON    EQU 0x00    ; Timer1 OFF
CFGT2CON    EQU 0x00    ; Timer2 OFF
CFGCCP1CON  EQU 0x00    ; PWM1 OFF
CFGCCP2CON  EQU 0x00    ; PWM2 OFF
CFGSSPCON   EQU 0x00    ; Serial Port OFF
CFGRCSTA    EQU 0X00
CFGTXSTA    EQU 0X00
;-----------------------------------------------------
; Registos para Viterbi CODEC (Q1900) de acordo com
; os 18 passos de programcao indicados no datasheet
; para operacao '3/4 Parallel Viterbi' (pg. 1-39 no
; referido datasheet)

;enderecos de registos a alterar em cada um dos 18 passos
VIT_ADD1    EQU 0x15
VIT_ADD2    EQU 0x16
VIT_ADD3    EQU 0x02
VIT_ADD4    EQU 0x03
VIT_ADD5    EQU 0x04
VIT_ADD6    EQU 0x08
VIT_ADD7    EQU 0x09
VIT_ADD8    EQU 0x0a
VIT_ADD9    EQU 0x0b
VIT_ADD10   EQU 0x0c
VIT_ADD11   EQU 0x17
VIT_ADD12   EQU 0x18
VIT_ADD13   EQU 0x06
VIT_ADD14   EQU 0x07
VIT_ADD15   EQU 0x04
VIT_ADD16   EQU 0x06
VIT_ADD17   EQU 0x04
VIT_ADD18   EQU 0x06

;dados a colocar nos registos em cada um dos 18 passos
VIT_REG1    EQU 0x00
VIT_REG2    EQU 0x00
VIT_REG3    EQU 0x08
VIT_REG4    EQU 0x01
VIT_REG5    EQU 0x01
VIT_REG6    EQU 0xf4
VIT_REG7    EQU 0xf9
VIT_REG8    EQU 0xfc
VIT_REG9    EQU 0xff
VIT_REG10   EQU 0xff
VIT_REG11   EQU 0x00
VIT_REG12   EQU 0x00
VIT_REG13   EQU 0x08
VIT_REG14   EQU 0x00
VIT_REG15   EQU 0x05
VIT_REG16   EQU 0x0a
VIT_REG17   EQU 0x01
VIT_REG18   EQU 0x08

;-----------------------------------------------------
; Macro de programacao do Q1900 (Viterbi Codec)

    variable I=1, J=1

VIT_WR macro step
    ; macro para escrita de registos de programacao do Q1900
    movlw   VIT_REG#v(step)
    movwf   PORTC
    movlw   VIT_ADD#v(step)+nRD+nWR
    movwf   PORTD
    movlw   VIT_ADD#v(step)+nRD
    movwf   PORTD
    movlw   VIT_ADD#v(step)+nRD+nWR
    movwf   PORTD
    endm
;-----------------------------------------------------
; Variaveis

W_ISR           EQU 0x20    ; guarda W em ISR
STATUS_ISR      EQU 0x21    ; guarda STATUS em ISR
COUNTER_LOW     EQU 0x22
COUNTER_HIGH    EQU 0x23
```

```
    org 0
    goto    START
    org 4
;-----------------------------------------------------
; Interrupt Service Routine

ISR bcf     STATUS,RP0  ; banco 0
    movwf   W_ISR       ; salvaguarda de contexto
    swapf   STATUS,W    ;   "
    movwf   STATUS_ISR  ;   "
    btfss   INTCON,RBIF ; se int. nao for causado por RB4..7 =>sai
    goto    FIM_ISR
    movf    PORTB,W

TST_RF btfss PORTB,RST_FIFO_FLAG
    goto    TST_UP
    call    RFIFO
    goto    FIM_ISR

TST_UP btfsc PORTB,CTR_UP
    call    UP

TST_DN btfsc PORTB,CTR_DOWN
    call    DOWN

FIM_ISR movf PORTB,W    ; ends mismatch
    andlw   0x30        ; para verificar se ocorreu novo UP e/ou DOWN

    btfss   STATUS,Z    ; se UP/DOWN estiverem inactivos, sai.
    goto    TST_RF      ; caso contrario, ...

    swapf   STATUS_ISR,W
    movwf   STATUS
    swapf   W_ISR,F
    swapf   W_ISR,W
    bcf INTCON,RBIF
    retfie
;---------------------------
RFIFO   btfss   PORTB,RST_FIFO_FLAG
    return

    movlw   RSTW+RSTR
    movwf   PORTA       ; => RSTW=1,RSTR=1
    movlw   RSTW+RSTR+CLK_W+CLK_R
    movwf   PORTA       ; ...=> SWCK=1,SRCK=1

    ; reset de contador interno
    movlw   .0
    movwf   COUNTER_LOW
    movwf   COUNTER_HIGH

    movlw   RSTW+RSTR
    movwf   PORTA       ; ...=> SWCK=0,SRCK=0

    ; reset de registo de saida
    movlw   .0
    movwf   PORTC
    movlw   CLK_LOW
    movwf   PORTE
    movlw   CLK_LOW+CLK_HIGH
    movwf   PORTE
    movlw   nFFDOWN_RESET
    movwf   PORTB
    movlw   .0
    movwf   PORTB
    movlw   CLK_HIGH
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    movlw   nFFUP_RESET
    movwf   PORTB
    movlw   nFFUP_RESET+nFFDOWN_RESET
    movwf   PORTB
    movlw   .0
    movwf   PORTA       ; => RSTW=0;RSTR=0
    return

UP  btfss   PORTB,CTR_UP
    return
```

```
    movlw   nFFDOWN_RESET
    movwf   PORTB       ; reset do FF correspondente
    movlw   nFFDOWN_RESET+nFFUP_RESET
    movwf   PORTB

    ; testa se contador interno == 0
    movf    COUNTER_LOW,W
    btfss   STATUS,Z
    goto    NOTZERO     ; COUNTER_LOW != 0
    movf    COUNTER_HIGH,W
    btfss   STATUS,Z
    goto    NOTZERO     ; COUNTER_HIGH != 0

    ; caso contador seja 0 e' preciso fazer
    ; reset do ponteiro de leitura nos FIFOs

    movlw   RSTR
    movwf   PORTA       ; => RSTR=1
    movlw   RSTR+CLK_R
    movwf   PORTA       ; ...=> SRCK=1
    movlw   RSTR
    movwf   PORTA       ; ...=> SRCK=0
    movlw   .0
    movwf   PORTA       ; => RSTR=0

    incf    COUNTER_LOW,F
    goto    SEND_CTR

NOTZERO incf COUNTER_LOW,F
    btfsc   STATUS,Z
    incf    COUNTER_HIGH,F

SEND_CTR
    movf    COUNTER_LOW,W
    movwf   PORTC
    movlw   CLK_LOW
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    movf    COUNTER_HIGH,W
    movwf   PORTC
    movlw   CLK_HIGH
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    return

DOWN btfss  PORTB,CTR_DOWN
    return

    movlw   nFFUP_RESET
    movwf   PORTB
    movlw   nFFDOWN_RESET+nFFUP_RESET
    movwf   PORTB
    movlw   .1
    subwf   COUNTER_LOW,F
    btfss   STATUS,C    ; C==0 => COUNTER_LOW==FF => COUNTER_HIGH--
    decf    COUNTER_HIGH,F
    movf    COUNTER_HIGH,W
    movwf   PORTC
    movlw   CLK_HIGH
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    movf    COUNTER_LOW,W
    movwf   PORTC
    movlw   CLK_LOW
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    return

;-----------------------------------------------------
; Rotina que efectua o atraso necessario antes dos passos
; 17 e 18 de programacao do Codec Q1900
; Assume-se um clock maior que 9000 bps e CLKIN=20MHz
; (20MHz/4)/9000bps * 2cycles => 1120 = 4 * 255 + 100
```

```
DELAY
J=1
    while J<=5
    ifdef   DELAYED
        movlw   .255
L#v(J)  addlw   .255    ; decrementa de uma unidade
        btfss   STATUS,Z
        goto    L#v(J)
    endif
J+=1
    endw
    return

;-----------------------------------------------------
; Rotina de inicializacao
CFG bcf STATUS,RP0 ; banco 0
    movlw   .0
    movwf   PORTA
    movwf   PORTB
    movwf   PORTC
    movwf   PORTD
    movwf   PORTE
    movlw   CFGADCON0
    movwf   ADCON0
    movlw   CFGINTCON
    movwf   INTCON
    movlw   CFGT1CON
    movwf   T1CON
    movlw   CFGT2CON
    movwf   T2CON
    movlw   CFGCCP1CON
    movwf   CCP1CON
    movlw   CFGCCP2CON
    movwf   CCP2CON
    movlw   CFGSSPCON
    movwf   SSPCON
    movlw   CFGRCSTA
    movwf   RCSTA

    bsf STATUS,RP0 ; banco 1
    movlw   CFGOPTION
    movwf   OPTION_REG
    movlw   CFGPIE1
    movwf   PIE1
    movlw   CFGPIE2
    movwf   PIE2
    movlw   CFGTRISA
    movwf   TRISA      ; define entradas/saidas de porto A
    movlw   CFGTRISB
    movwf   TRISB      ; define entradas/saidas de porto B
    movlw   CFGTRISC
    movwf   TRISC      ; define entradas/saidas de porto C
    movlw   CFGTRISD
    movwf   TRISD      ; define entradas/saidas de porto D
    movlw   CFGTRISE
    movwf   TRISE      ; define entradas/saidas de porto E e slave port
    movlw   CFGADCON1
    movwf   ADCON1
    movlw   CFGTXSTA
    movwf   TXSTA

    bcf STATUS,RP0 ; banco 0

    ; inicializacao de CODEC Viterbi Q1900
    movlw   RST_VITERBI+FPGA3_RESET
    movwf   PORTB

    while   I<=.18
        if I>=.17
            call DELAY; atrasa steps 17 e 18 de acordo c/ datasheet
        endif
        VIT_WR  I   ; executa step I de configuracao Q1900
I+=1
    endw

    ; desactivacao de CS de Q1900 (activo a LOW)
    movlw   0xe0    ; nWR=nRD=nCS=1
    movwf   PORTD
```

```
    bcf STATUS,RP0 ; banco 0
    ; inicializacao de contador interno
    movlw   .0
    movwf   COUNTER_LOW
    movwf   COUNTER_HIGH

    ; reset registos de saida do contador
    movwf   PORTC
    movlw   CLK_LOW
    movwf   PORTE
    movlw   .0
    movwf   PORTE
    movlw   CLK_HIGH
    movwf   PORTE
    movlw   .0
    movwf   PORTE

    ;fim de reset a FPGA3 e continuacao de reset a Q1900
    movlw   RST_VITERBI
    movwf   PORTB

    ;clocks escrita/leitura dos FIFOs para funcionamento correcto
    ;como esta' mencionado no datasheet (pg.2-'Power-up')
    movlw   CLK_R+CLK_W
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ; reset de leitura dos FIFOS (== reset RST_FIFO_FLAG)
    movlw   RSTR
    movwf   PORTA
    movlw   RSTR+CLK_R
    movwf   PORTA
    movlw   RSTR
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ; reset de escrita dos FIFOS (== reset PAL1)
    movlw   RSTW
    movwf   PORTA
    movlw   RSTW+CLK_W
    movwf   PORTA
    movlw   RSTW
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ;clocks escrita/leitura dos FIFOs para funcionamento correcto
    ;como esta' mencionado no datasheet (pg.2-'Power-up')
    movlw   CLK_R+CLK_W
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ; reset de leitura dos FIFOS (== reset RST_FIFO_FLAG)
    movlw   RSTR
    movwf   PORTA
    movlw   RSTR+CLK_R
    movwf   PORTA
    movlw   RSTR
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ; reset de escrita dos FIFOS (== reset PAL1)
    movlw   RSTW
    movwf   PORTA
    movlw   RSTW+CLK_W
    movwf   PORTA
    movlw   RSTW
    movwf   PORTA
    movlw   .0
    movwf   PORTA

    ; fim de reset FFs UP/DOWN
    movlw   nFFUP_RESET+RST_VITERBI
    movwf   PORTB
```

```
    movlw   nFFUP_RESET+nFFDOWN_RESET+RST_VITERBI
    movwf   PORTB

    ;fim de reset a codificador Viterbi
    movlw   nFFUP_RESET+nFFDOWN_RESET
    movwf   PORTB

    return

;-----------------------------------------------------
; Rotina Principal
START bcf   INTCON,GIE  ; desactiva interrupcoes
    call    CFG
    movf    PORTB,F     ;ends mismatch
    bcf     INTCON,RBIF
    bsf     INTCON,GIE  ; activa interrupcoes

LOOP NOP
    goto    LOOP
    END
```

J Listagem do código de programação das PALs utilizadas

J.1 PAL1 DO MÓDULO DE CONTROLO DO *BUFFER* DE EMISSÃO

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    PAL1 no emissor
PATTERN
REVISION
AUTHOR   Alexandre Abreu & Nuno Roma
COMPANY  IST/INESC
DATE     08/21/98

CHIP  pale1  PAL22V10
;---------------------------------- PIN Declarations ---------------
PIN  1          FBIT                    COMBINATORIAL   ; INPUT
PIN  2          RESET                   COMBINATORIAL   ; INPUT
PIN  3          B17A                    COMBINATORIAL   ; INPUT
PIN  4          B17B                    COMBINATORIAL   ; INPUT
PIN  5          QHA                     COMBINATORIAL   ; INPUT
PIN  6          QHB                     COMBINATORIAL   ; INPUT
PIN  7          CLK_RA                  COMBINATORIAL   ; INPUT
PIN  8          CLK_RB                  COMBINATORIAL   ; INPUT
PIN  14         SRCKA                   COMBINATORIAL   ; OUTPUT
PIN  15         SRCKB                   COMBINATORIAL   ; OUTPUT
PIN  16         SEL                     REGISTERED      ; OUTPUT
PIN  17         CLKSR                   COMBINATORIAL   ; OUTPUT
PIN  18         SHFNLD                  COMBINATORIAL   ; OUTPUT
PIN  19         DOUT                    COMBINATORIAL   ; OUTPUT
PIN  20..23     /COUNTER_MACH[3..0]     REGISTERED      ; OUTPUT
NODE 1          GLOBAL

;---------------------------------- STRING Declarations ---------------
; ESTADOS DE MAQUINA DE ESTADOS DE SINCRONIZACAO
STRING  E0 '#D0'
STRING  E1 '#D1'
STRING  E2 '#D2'
STRING  E3 '#D3'
STRING  E4 '#D4'
STRING  E5 '#D5'
STRING  E6 '#D6'
STRING  E7 '#D7'
STRING  E8 '#D8'
STRING  E9 '#D9'
STRING  E10 '#D10'
STRING  E11 '#D11'
STRING  E12 '#D12'
STRING  E13 '#D13'
STRING  E14 '#D14'
STRING  E15 '#D15'

; ESTADOS DE MAQUINA DE ESTADOS DE SELECCAO DE FIFO
STRING  SELA '#D0'
STRING  SELB '#D1'
;---------------------------------- Boolean Equation Segment ------
EQUATIONS

   GLOBAL.RSTF=RESET
   CASE (COUNTER_MACH[3..0])
   BEGIN
      E0:
         BEGIN
            COUNTER_MACH[3..0]=E1
         END
      E1:
         BEGIN
            COUNTER_MACH[3..0]=E2
         END
      E2:
         BEGIN
            COUNTER_MACH[3..0]=E3
         END
      E3:
         BEGIN
            COUNTER_MACH[3..0]=E4
```

```
         END
      E4:
         BEGIN
            COUNTER_MACH[3..0]=E5
         END
      E5:
         BEGIN
            COUNTER_MACH[3..0]=E6
         END
      E6:
         BEGIN
            COUNTER_MACH[3..0]=E7
         END
      E7:
         BEGIN
            COUNTER_MACH[3..0]=E8
         END
      E8:
         BEGIN
            COUNTER_MACH[3..0]=E9
         END
      E9:
         BEGIN
            COUNTER_MACH[3..0]=E10
         END
      E10:
         BEGIN
            COUNTER_MACH[3..0]=E11
         END
      E11:
         BEGIN
            COUNTER_MACH[3..0]=E12
         END
      E12:
         BEGIN
            COUNTER_MACH[3..0]=E13
         END
      E13:
         BEGIN
            COUNTER_MACH[3..0]=E14
         END
      E14:
         BEGIN
            COUNTER_MACH[3..0]=E15
         END
      E15:
         BEGIN
            COUNTER_MACH[3..0]=E0
         END
   END

   CASE (SEL)
   BEGIN
      SELA:
         BEGIN
            IF (/B17A) THEN
               BEGIN
                  SEL=SELA
               END
            ELSE
               BEGIN
                  IF (COUNTER_MACH[3..0]=E15) THEN
                     BEGIN
                        SEL=SELB
                     END
                  ELSE
                     BEGIN
                        SEL=SELA
                     END
               END
         END
      SELB:
         BEGIN
            IF (/B17B) THEN
               BEGIN
                  SEL=SELB
               END
            ELSE
               BEGIN
```

```
                IF (COUNTER_MACH[3..0]=E15) THEN
                   BEGIN
                      SEL=SELA
                   END
                ELSE
                   BEGIN
                      SEL=SELB
                   END
             END
       END
END

IF ((COUNTER_MACH[3..0]=E15) * /FBIT) THEN
   BEGIN
      SRCKA=/SEL+CLK_RA
      SRCKB=SEL+CLK_RB
   END
ELSE
   BEGIN
      SRCKA=CLK_RA
      SRCKB=CLK_RB
   END

CLKSR = /FBIT
IF (COUNTER_MACH[3..0]=E0) THEN
   BEGIN
      SHFNLD = GND
   END
ELSE
   BEGIN
      SHFNLD = VCC
   END

DOUT = QHA * /SEL + QHB * SEL
```

J.2 PAL2 DO MÓDULO DE CONTROLO DO *BUFFER* DE EMISSÃO

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    PAL2 no emissor
PATTERN
REVISION
AUTHOR   Alexandre Abreu & Nuno Roma
COMPANY  IST/INESC
DATE     08/21/98

CHIP  pale2  PAL22V10
;---------------------------------- PIN Declarations ---------------
PIN  1          OUT0_A      COMBINATORIAL ; INPUT
PIN  2          OUT0_B      COMBINATORIAL ; INPUT
PIN  3          NWR_A       COMBINATORIAL ; INPUT
PIN  4          NWR_B       COMBINATORIAL ; INPUT
PIN  5          EOC_WA      COMBINATORIAL ; INPUT
PIN  6          EOC_RA      COMBINATORIAL ; INPUT
PIN  7          EOC_WB      COMBINATORIAL ; INPUT
PIN  8          EOC_RB      COMBINATORIAL ; INPUT
PIN  9          CLK_W       COMBINATORIAL ; INPUT
PIN  10         RESET_RA    COMBINATORIAL ; INPUT
PIN  11         RESET_RB    COMBINATORIAL ; INPUT
PIN  13         G_RESET     COMBINATORIAL ; INPUT
PIN  14         SWCK_A      COMBINATORIAL ; OUTPUT
PIN  15         SWCK_B      COMBINATORIAL ; OUTPUT
PIN  16         MR_WA       COMBINATORIAL ; OUTPUT
PIN  17         MR_RA       COMBINATORIAL ; OUTPUT
PIN  18         MR_WB       COMBINATORIAL ; OUTPUT
PIN  19         MR_RB       COMBINATORIAL ; OUTPUT

;---------------------------------- Boolean Equation Segment ------
EQUATIONS
   SWCK_A = OUT0_A * /NWR_A + CLK_W
   SWCK_B = OUT0_B * /NWR_B + CLK_W
   MR_WA = EOC_WA + G_RESET
   MR_RA = EOC_RA + RESET_RA
   MR_WB = EOC_WB + G_RESET
   MR_RB = EOC_RB + RESET_RB
```

J.3 PAL3 DO MÓDULO DE CONTROLO DO *BUFFER* DE EMISSÃO

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    PAL3 no emissor
PATTERN
REVISION
AUTHOR   Alexandre Abreu & Nuno Roma
COMPANY  IST/INESC
DATE     08/21/98

CHIP  pale3  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          OUT1_A          COMBINATORIAL ; INPUT
PIN  2          OUT1_B          COMBINATORIAL ; INPUT
PIN  3          NRD_A           COMBINATORIAL ; INPUT
PIN  4          NRD_B           COMBINATORIAL ; INPUT
PIN  5          CLK_REG         COMBINATORIAL ; INPUT
PIN  6          SEL_REG         COMBINATORIAL ; INPUT
PIN  7          SEL_HIGH        COMBINATORIAL ; INPUT
PIN  8          K0              COMBINATORIAL ; INPUT
PIN  9          K1              COMBINATORIAL ; INPUT
PIN  10         VCO_OUT         COMBINATORIAL ; INPUT
PIN  11         OE_REGA         COMBINATORIAL ; INPUT
PIN  13         OE_REGB         COMBINATORIAL ; INPUT
PIN  14         CLK_HIGH_REGA   COMBINATORIAL ; OUTPUT
PIN  15         CLK_LOW_REGA    COMBINATORIAL ; OUTPUT
PIN  16         CLK_HIGH_REGB   COMBINATORIAL ; OUTPUT
PIN  17         CLK_LOW_REGB    COMBINATORIAL ; OUTPUT
PIN  18         K0K1            COMBINATORIAL ; OUTPUT
PIN  19         NVCO_OUT        COMBINATORIAL ; OUTPUT

;---------------------------------- Boolean Equation Segment ------
EQUATIONS
   OE_REGA = OUT1_A * /NRD_A
   OE_REGB = OUT1_B * /NRD_B
   CLK_HIGH_REGA = CLK_REG + /SEL_REG +  SEL_HIGH
   CLK_LOW_REGA  = CLK_REG + /SEL_REG + /SEL_HIGH
   CLK_HIGH_REGB = CLK_REG +  SEL_REG +  SEL_HIGH
   CLK_LOW_REGB  = CLK_REG +  SEL_REG + /SEL_HIGH
   K0K1 = K0 * K1
   NVCO_OUT = /VCO_OUT
```

J.4 PAL1 DO MÓDULO DE CONTROLO DO *BUFFER* DE RECEPÇÃO

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    PAL1 no receptor - 2 maquinas de estados
PATTERN
REVISION
AUTHOR   Alexandre Abreu & Nuno Roma
COMPANY  IST/INESC
DATE     08/21/98

CHIP  palr1  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          FBIT                COMBINATORIAL ; INPUT
PIN  2          RESET               COMBINATORIAL ; INPUT
PIN  3          NW                  COMBINATORIAL ; INPUT
PIN  4          DI                  COMBINATORIAL ; INPUT
PIN  5          CLK_W               COMBINATORIAL ; INPUT
PIN  6          EOC_W               COMBINATORIAL ; INPUT
PIN  14         MR_W                COMBINATORIAL ; OUTPUT
PIN  15         WE                  COMBINATORIAL ; OUTPUT
PIN  16..19     COUNTER_MACH[3..0]  REGISTERED ; OUTPUT
PIN  20..23     SYNC_MACH[3..0]     REGISTERED ; OUTPUT
NODE 1 GLOBAL
;---------------------------------- STRING Declarations ---------------
; ESTADOS DE MAQUINA DE ESTADOS DO CONTADOR DE 4 BITS
STRING  E0 '#D0'
STRING  E1 '#D1'
STRING  E2 '#D2'
STRING  E3 '#D3'
STRING  E4 '#D4'
STRING  E5 '#D5'
STRING  E6 '#D6'
STRING  E7 '#D7'
STRING  E8 '#D8'
STRING  E9 '#D9'
STRING  E10 '#D10'
STRING  E11 '#D11'
STRING  E12 '#D12'
STRING  E13 '#D13'
STRING  E14 '#D14'
STRING  E15 '#D15'

; ESTADOS DE MAQUINA DE ESTADOS DE SINCRONIZACAO
STRING  NW0_HUNT    '#D0'
STRING  NW0_HUNTED  '#D1'
STRING  B0          '#D2'
STRING  B1          '#D3'
STRING  B2          '#D4'
STRING  B3          '#D5'
STRING  B4          '#D6'
STRING  B5          '#D7'
STRING  NW1_HUNT    '#D8'
STRING  NW1_HUNTED  '#D9'

STRING  CLR_COUNTER '((SYNC_MACH[3..0]=NW0_HUNT) +
                      (SYNC_MACH[3..0]=NW0_HUNTED) +
                      (SYNC_MACH[3..0]=NW1_HUNT) +
                      (SYNC_MACH[3..0]=NW1_HUNTED))'
;---------------------------------- Boolean Equation Segment ------
EQUATIONS

   GLOBAL.RSTF=RESET
   MR_W=RESET+EOC_W

   IF(((SYNC_MACH[3..0]=NW0_HUNT)*NW)+(COUNTER_MACH[3..0]=E15)+CLK_W) THEN
      BEGIN
         WE=VCC
      END
   ELSE
      BEGIN
         WE=GND
      END

   CASE (SYNC_MACH[3..0])
```

```
BEGIN
   NW0_HUNT:
      BEGIN
         IF (/NW) THEN
            BEGIN
               SYNC_MACH[3..0]=NW0_HUNT
            END
         ELSE
            BEGIN
               IF (/DI) THEN
                  BEGIN
                     SYNC_MACH[3..0]=NW0_HUNTED
                  END
               ELSE
                  BEGIN
                     SYNC_MACH[3..0]=B0
                  END
            END
      END
   NW0_HUNTED:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=NW0_HUNTED
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=B0
            END
      END
   B0:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=B1
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNT
            END
      END
   B1:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=B2
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNT
            END
      END
   B2:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=B3
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNT
            END
      END
   B3:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=B4
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNT
            END
      END
   B4:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=B5
```

```
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNT
            END
      END
   B5:
      BEGIN
         SYNC_MACH[3..0]=B5
      END
   NW1_HUNT:
      BEGIN
         IF (/NW) THEN
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNT
            END
         ELSE
            BEGIN
               IF (/DI) THEN
                  BEGIN
                     SYNC_MACH[3..0]=NW1_HUNTED
                  END
               ELSE
                  BEGIN
                     SYNC_MACH[3..0]=B0
                  END
            END
      END
   NW1_HUNTED:
      BEGIN
         IF (/DI) THEN
            BEGIN
               SYNC_MACH[3..0]=NW1_HUNTED
            END
         ELSE
            BEGIN
               SYNC_MACH[3..0]=B0
            END
      END
END

CASE (COUNTER_MACH[3..0])
BEGIN
   E0:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E1
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E1:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E2
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E2:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E3
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E3:
      BEGIN
         IF (/CLR_COUNTER) THEN
```

```
            BEGIN
               COUNTER_MACH[3..0]=E4
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E4:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E5
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E5:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E6
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E6:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E7
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E7:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E8
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E8:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E9
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E9:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E10
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E10:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E11
```

```
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E11:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E12
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E12:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E13
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E13:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E14
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E14:
      BEGIN
         IF (/CLR_COUNTER) THEN
            BEGIN
               COUNTER_MACH[3..0]=E15
            END
         ELSE
            BEGIN
               COUNTER_MACH[3..0]=E0
            END
      END
   E15:
      BEGIN
         COUNTER_MACH[3..0]=E0
      END
   OTHERWISE:
      BEGIN
         COUNTER_MACH[3..0]=E0
      END
END
```

J.5 PAL2 DO MÓDULO DE CONTROLO DO *BUFFER* DE RECEPÇÃO

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    PAL2 no receptor
PATTERN
REVISION
AUTHOR   Alexandre Abreu & Nuno Roma
COMPANY  IST / INESC
DATE     08/20/98

CHIP  palr2  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          CLOCK           COMBINATORIAL ; INPUT
PIN  2          OUT0            COMBINATORIAL ; INPUT
PIN  3          OUT1            COMBINATORIAL ; INPUT
PIN  4          NRD             COMBINATORIAL ; INPUT
PIN  5          NWR             COMBINATORIAL ; INPUT
PIN  6          RSTFIFO         COMBINATORIAL ; INPUT
PIN  7          NHOLD           COMBINATORIAL ; INPUT
PIN  8          A0              COMBINATORIAL ; INPUT
PIN  9          FBIT            COMBINATORIAL ; INPUT
PIN  10         CLK_W           COMBINATORIAL ; INPUT
PIN  11         CLK_R           COMBINATORIAL ; INPUT
PIN  13         EOC_R           COMBINATORIAL ; INPUT
PIN  15         MR_R            COMBINATORIAL ; OUTPUT
PIN  16         SRCK            COMBINATORIAL ; OUTPUT
PIN  17         SWCK            COMBINATORIAL ; OUTPUT
PIN  18         RE              COMBINATORIAL ; OUTPUT
PIN  19         RSTW            COMBINATORIAL ; INPUT
PIN  20         OE_BUF245       COMBINATORIAL ; OUTPUT
PIN  21         OE_REG          COMBINATORIAL ; OUTPUT
PIN  22         RSTFIFO_FLAG    REGISTERED    ; OUTPUT
PIN  23         CLOCKOUT        COMBINATORIAL ; OUTPUT
NODE 1          GLOBAL
;---------------------------------- Boolean Equation Segment ------
EQUATIONS

MR_R = RSTFIFO + EOC_R
SRCK = (NRD * /RSTFIFO) + CLK_R + (/NHOLD * A0)
SWCK = (/FBIT * /RSTW) + CLK_W
RE = OUT0 + CLK_R
OE_BUF245 = /OUT0
OE_REG = /(OUT1 * /NRD)
RSTFIFO_FLAG := GND
GLOBAL.RSTF = RSTFIFO
CLOCKOUT = OUT1 * /NWR
```

K Artigo: "Digital Video Transmission Through the Electrical Power Lines"

No presente Apêndice apresentamos o artigo realizado com vista a uma posterior apresentação formal na conferência *"The Second European DSP Education and Research Conference"*, organizada pela Texas Instruments e que teve lugar durante os dias 23 e 24 de Setembro de 1998 na *École Supérieure d' Ingénieurs en Electrotechnique et Electronique-ESIEE* em Noisy le Grand, Paris.

Todo este processo culminou na inclusão do artigo produzido na publicação "*Proceedings – Image Processing"* [26] da Texas Instruments.

INSTITUTO SUPERIOR TÉCNICO

- TRABALHO FINAL DE CURSO -

SISTEMA DE TRANSMISSÃO DE VÍDEO ATRAVÉS DA REDE ELÉCTRICA

Autores: **Alexandre Abreu** **Nº 39775**
Nuno Roma **Nº 39921**

Professor Responsável: Prof. Doutor José A. B. Gerald
Docente Acompanhante: Prof. Doutor Leonel A. Sousa

Licenciatura em Engenharia Electrotécnica e de Computadores
Secção de Electrónica

Lisboa

Dezembro de 1998