Projeto e Concepção de um Microprocessador Dedicado para Pré-Decodificação de Vídeo MPEG-4

Relatório referente ao Trabalho de Conclusão de Curso, sob orientação dos professores Elmar U. K. Melcher e R. C. S. Freire, submetido à coordenação do curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, como parte dos requisitos necessários a obtenção do diploma de graduação.

Elmar Uwe Kurt Melcher, Dr.
Orientador

Campina Grande, Paraíba, Brasil

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Dedico este trabalho à minha filha Paloma.

Agradecimentos

Agradeço a todos aqueles que me ajudaram durante o decorrer do meu estagio.

Agradeço a equipe do Laboratório de Arquiteturas Dedicadas da qual participei como membro.

De forma especial agradeço:

Aos meus pais, o engenheiro Carlos A. Javier Zurita, e Rosa Maria V. Zurita, a quem devo muitos dos meus conhecimentos e minha formação, e que lutaram pela realização do meu curso, tornando este sonho possível.

Ao professor Elmar U. K. Melcher pela sua orientação e pelos inúmeros e valiosos ensinamentos e experiência de inestimável valor que me passou longo de todo este trabalho.

Aos professor Raimundo C. S. Freire e Glauco Fontgalland, que me integraram no projeto de intercambio Capes/Braftec, permitindo-me ampliar meus conhecimentos em microeletrônica, essenciais para este projeto.

Ao engenheiro Wilson Rosas V. Neto, pela fundamental participação em todas as fases deste projeto, tendo prestado por diversas vezes excelentes ideias na solução de problemas e aperfeiçoamento do sistema.

A George Silveira, pelo seu entusiasmo e pelo inestimável trabalho de codificar todo o algoritmo do pré-decodificador na linguagem assembly do microprocessador desenvolvido, reportando erros e alterações a serem feitas.

Resumo

O presente documento descreve o processo de projeto e concepção de um microprocessador dedicado a pré-decodificação de vídeos MPEG-4.

Uma vez que este projeto envolve uma bagagem considerável de conhecimentos nas áreas de processamento e compressão de vídeo, projeto de sistemas digitais de larga escala, arquitetura de computadores e projeto de circuitos integrados, cada assunto pertinente é abordado da maneira mais breve possível, de forma a minimizar o volume deste trabalho.

O padrão MPEG-4 define um complexo sistema de decodificação de vídeo. Por esta razão, os capítulos inciais destinam-se a apresentar suas principais características e pontos de interesse do projeto. O Bloco de pré-decodificação, alvo principal do decodificador neste trabalho, é abordado de forma mais detalhada no Capitulo 4. Os capítulos seguintes exploram todo o processo de concepção do sistema em questão.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A difusão de vídeo sob a forma digital tem crescido rapidamente nos últimos anos. Este crescimento se deve principalmente à popularização das mídias de armazenamento digital de alta capacidade, como CD's e DVD's e ao crescimento acentuado da internet. A evolução dos sistemas de transmissão de TV para o sistemas de TV digital de alta definição (HDTV), que já se encontra em fase inicial de utilização em alguns países, deve provocar um aumento ainda maior nessa taxa de crescimento.

Todavia, a produção de um vídeo sob a forma digital geralmente produz uma quantidade muito elevada de dados, o que dificulta, ou mesmo inviabiliza, sua transmissão ou armazenamento de forma direta. Faz-se então necessário comprimir os dados que contém o vídeo antes de o transmitir ou armazenar.

Existem atualmente vários padrões para a compressão de vídeo, no entanto os padrões MPEG são atualmente os mais difundidos. A sigla MPEG vem de Moving Picture Expert Group, um grupo formado por pesquisadores e colaboradores de diversos países, e tem por objetivo gerar as especificações do padrão junto a ISO (International Standardization Organization) e ao IEC (International Electrotechnical Commission).

O microprocessador desenvolvido neste trabalho destina-se unicamente a sistemas de decodificação de vídeo MPEG, ou, mais precisamente, vídeos MPEG-4, cujas características serão abordadas no capitulo seguinte.

Capítulo 2

Os Padrões MPEG

2.1 O Padrão MPEG-1

O primeiro padrão desenvolvido pelo grupo MPEG, apelidado MPEG-1, foi o código que combinava sinais audiovisuais a uma taxa de 1,5Mb/s.

Ele foi motivado pelo fato de que se estava tornado possível em 1988 armazenar sinais de vídeo em CD's com qualidade comparável às fitas cassete VHS.

MPEG-1 foi um padrão muito inovador. Pela primeira vez um único padrão audiovisual foi produzido e todas as precauções foram tomadas para que as peças do padrão se encaixassem. O sucesso causado pelo padrão fez com que várias empresas que até então possuíam departamentos de áudio e vídeo independentes se reorganizassem. Além disso o padrão foi o primeiro em outras coisas:

- primeiro padrão para processamento de sinais desenvolvido usando código em C;
- primeiro código de vídeo independente do formato do vídeo;
- primeiro padrão a incluir implementação em software

MPEG-1 é um padrão que fornece uma apresentação normalizada o que permite aos desenvolvedores realizar o tradicional paradigma dos sistemas de comunicação. A informação audiovisual pode ser gerada em tempo real de um acontecimento natural ou vir de um servidor. Em ambos os casos um fluxo de bits multiplexado chega ao decodificador através de um meio de distribuição (uma rede de comunicação, um canal de transmissão, etc.). No caso de um disco local a parte de distribuição do modelo desaparece mas o restante continua válido. Os fluxos de áudio e vídeo codificados, forçados a ter uma base de tempo comum e combinados em um mesmo fluxo pela camada de sistema do MPEG, são

extraídos e manipulados pelos decodificadores de áudio e vídeo apropriados que produzem seqüências de amostras PCM representando informação de som e imagem.

O padrão MPEG-1, formalmente conhecido como ISO/IEC 11172, é dividido em 5 partes. As três primeiras partes são Sistema, Vídeo e Áudio respectivamente. As outras duas partes são: Teste de Conformidade, que especifica a metodologia para verificação das exigências de conformidade do padrão pelos fabricantes de equipamentos, e Software Simulation, uma implementação na linguagem C do codificador e decodificador do padrão MPEG-1.

2.2 O Padrão MPEG-2

Cumprindo a promessa de distribuir vídeo em qualidade VHS e qualidade de áudio de CD a uma taxa de 1.5Mb/s, MPEG-1 tornou o vídeo digital possível. Embora tenha sido mostrado que este padrão operasse bem em imagens de alta resolução e a maiores taxas de transmissão, um grande numero de industrias tinham o interesse no campo muito mais amplo da "televisão digital"e se sentiam muito desconfortáveis com uma limitação básica do vídeo MPEG-1: a limitação das imagens progressivas. Um ganho, eventualmente estimado em cerca de 20%, era esperado se fosse explorado a correlação entre imagens entrelaçadas. Esta foi a motivação principal para a criação do padrão MPEG-2 intitulado "Generic coding of moving pictures and associated audio".

Em 1990 quando começaram os trabalhos neste novo padrão era esperado que a tecnologia VLSI estaria pronta para implementar decodificadores de vídeo integrados para manipular imagens de TV em tela cheia a taxas de transmissão acima de 10Mb/s.

As funções do decodificador MPEG-2 são semelhantes ao do MPEG-1, mas um importante componente tecnológico é o suporte à interação cliente-servidor por meios de um protocolo padrão de comunicação.

Ao contrario do MPEG-1 que é basicamente um padrão para guardar vídeos em um disco em baixas taxas de transmissão, o grande numero de aplicações do padrão MPEG-2 forçou o grupo MPEG a desenvolver e implementar uma especie de "kit de ferramentas". Diferentes ferramentas de codificação servindo a diferente propósitos foram desenvolvidas e padronizadas. Diferentes conjuntos de ferramentas, chamadas perfis (*profiles*), que são também padronizados e podem ser usados para diferentes necessidades. Cada perfil tem em geral diferentes níveis para alguns parâmetros (ex: tamanho da imagem e taxa de transmissão).

MPEG-2, formalmente conhecido como ISO/IEC 13818, também é um padrão de múltiplas partes. As 5 primeiras partes têm a mesma função das suas correspondentes no padrão MPEG-1. A parte 6 do MPEG-2, intitulada "Digital Storage Media Command

and Control (DSM-CC)", é a especificação de uma série de protocolos para funções de controle e operações especificas para trabalhar com os fluxos de bits MPEG. A parte 7 é a especificação de um algoritmo de canal múltiplo de codificação de áudio sem a obrigação de ser compatível com o padrão MPEG-1. A parte 9 do padrão MPEG-2 é a especificação da Interface em Tempo Real para os decodificadores de fluxo de transporte que devem ser utilizados para adaptação à todas as redes contendo fluxo de transporte.

2.3 O Padrão MPEG-4

Antecipando a rápida convergência das industrias de telecomunicações, computadores, filme e televisão, o grupo MPEG deu inicio em 1994 ao processo de padronização de um novo padrão para compressão de vídeo, o MPEG-4.

O padrão MPEG-4 foi criado reunindo uma série de algorítimos e ferramentas de codificação e de representação flexível de dados áudio-visuais, permite também o uso de outros tipos de mídia interagindo com o vídeo (como textos e fotos digitais por exemplo), destinando-se a atender os desafios das futuras aplicações multimídia em diferentes ambientes.

Em particular, a forma de endereçamento dos dados codificados no MPEG-4 permite que o mesmo possa ser empregado em qualquer meio, seja ele de armazenamento (como CD's, DVD's ou HD's, por exemplo), ou por transmissão em fluxo continuo (como na HDTV ou em videoconferências, por exemplo). Tal característica somada a sua robustez em ambientes propícios a erros, o suporte a interatividade, capacidade de codificação de vídeos de origem real e sintética (animação gráfica) bem como uma elevada eficiência de compressão levaram este padrão a ser rapidamente absorvido pela industria, e a se popularizar como opção preferencial em diversos meios, apos a publicação oficial do padrão na ISO/IEC em 1998, dez anos apos o lançamento do MPEG-1.

A taxa de compressão, assim como no MPEG-1 e no MPEG-2 também é variável (um mesmo padrão, como o MPEG-4, pode ser utilizado para comprimir mais ou comprimir menos o conteúdo original). No entanto, ao contrário do MPEG-2, cuja qualidade é aproximadamente fixa em torno do padrão DVD de qualidade, para o MPEG-4 essa variação é bem maior. Praticamente qualquer valor de taxa de compressão pode ser utilizado, permitindo a visualização das imagens do vídeo não importando a capacidade do meio de transmissão (Internet de banda larga ou linha discada por exemplo).

Contudo, o aumento da taxa de compressão e recursos agregados tem como custo o aumento do esforço computacional despendido na manipulação dos vídeos, tanto na codificação quanto na decodificação, sendo maior do que o exigido no formato MPEG-2, que por sua vez também exige mais do que o padrão MPEG-1.

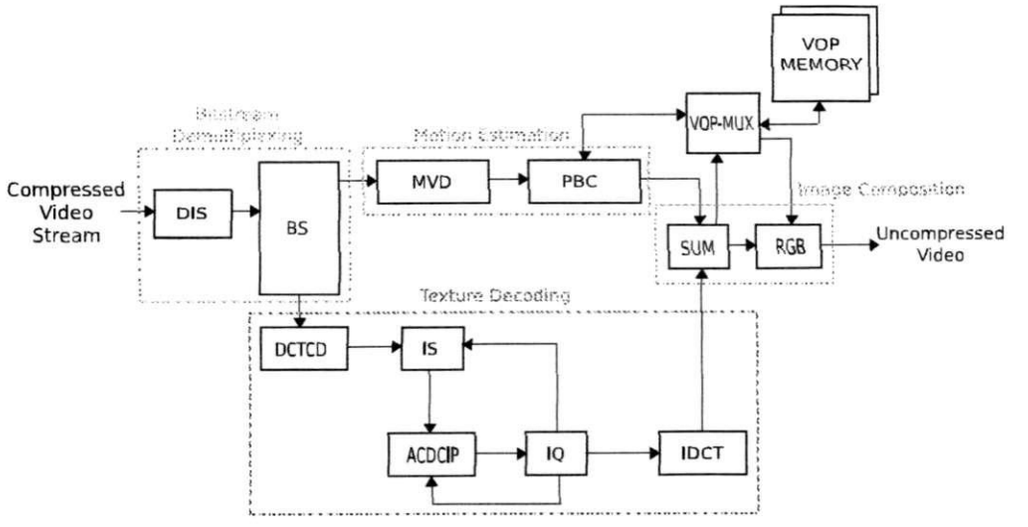

Figura 2.1: Diagrama de blocos de um decodificador MPEG-4

Como mencionado, o MPEG-4 é um padrão ISO/IEC pertencente à família MPEG e resultado de um esforço internacional de centenas de pesquisadores e engenheiros de todo o mundo. O padrão segue o paradigma de orientação a objetos, e seu foco está na definição de documentos hipermídia, comumente chamados de cenas no MPEG-4.

Uma grande inovação em termos de compressão no padrão MPEG-4 com relação aos padrões anteriores é o fato do mesmo considerar os objetos da imagem como sendo independentes entre si. Isto permite que regiões que permaneçam inalteradas entre quadros subsequentes do vídeo não precisem ser re-enviadas pelo codificador. O uso de técnicas de predição para explorar as correlações entre quadros consecutivos do vídeo também permitiu um aumento significativo na eficiência da compressão.

Uma cena MPEG-4 é dividida em objetos textuais, objetos gráficos, objetos de áudio e objetos de vídeo, podendo os dois últimos serem naturais ou sintéticos. Esses objetos podem ser colocados em qualquer posição da cena, tanto do ponto de vista temporal quanto do ponto de vista espacial, sendo possível também que transformações sejam aplicadas aos mesmos. Ainda é possível agrupar esses objetos que compõem uma cena (também chamados de objetos de mídia) de modo a formar objetos mais complexos. Através desse agrupamento, ou composição, torna-se possível à interação entre os próprios objetos e a interação do usuário final com a cena.

Podemos definir as principais características do padrão MPEG-4 como sendo:

- Compressão de imagens estáticas em níveis de cinza ou cores;
- Algoritmo baseado na codificação por transformada do cosseno discreta- DCT;

- Exploração das características visuais humanas para melhorar a compressão;
- Codificação do vídeo com ou sem perdas (irreversível ou reversível);
- Compromisso entre a qualidade e o fator de compressão;

2.3.1 Modos de Codificação

2.3.1.1 Modo Intra (Codificação da Textura)

A codificação de um Bloco Intra (também chamados blocos I) utiliza somente codificação por transformada e serve como ponto para o acesso aleatório à seqüência de vídeo codificada. Todos os blocos são codificados com DCT (Transformada do Cosseno Discreta), quantificados e em seguida é codificados usando um código de comprimento variável, assim como a compressão e codificação de imagens JPEG.

A sequência de codificação de um Bloco Intra é:

Passo 1: A imagem é dividida em blocos de dimensão 8x8.

Passo 2: É calculada a DCT bidimensional de cada bloco com base na equação 2.1.

$$F_{DCT}(x,y) = \frac{C(x)C(y)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} f_{DCT}(i,j) cos\left(\frac{(2i+1)x\pi}{16}\right) cos\left(\frac{(2j+1)y\pi}{16}\right) \quad (2.1)$$

onde $f_{DCT}(i,j)$ são 64 amostras de um componente de cor (Y, U ou V) do bloco a ser codificado e $F_{DCT}(x,y)$ são os 64 coeficientes DCT de saida e $C(x)$ e $C(y)$ são constantes definidas pela Equação 2.2

$$C(n) = \begin{cases} \frac{1}{\sqrt{2}} & , n = 0 \\ 1 & , n \neq 0 \end{cases} \quad (2.2)$$

Passo 3: Os coeficientes de cada bloco são selecionados e quantificados de acordo com uma matriz de quantificação T (Equação 2.4), conforme a Equação 2.3.

$$F_{DCT}(x,y) = round\left(\frac{F_{DCT}(x,y)}{T(x,y)}\right) \quad (2.3)$$

$$T = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{2.4}$$

A matriz de quantificação é determinada de acordo com as características visuais humanas e pode ser escalada para permitir diferentes níveis de compressão.

Passo 4: Os coeficientes quantificados são reordenados utilizando um procedimento em ziguezague, formando uma seqüência unidimensional.

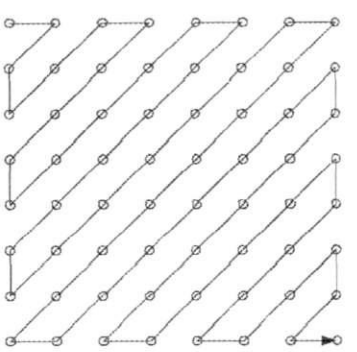

Figura 2.2: Sequência de zig-zag num bloco

Aos coeficientes DC é aplicada uma codificação diferencial, ao passo que aos coeficiente AC é aplicada uma codificação RLC (Run Length Code) ao número de coeficientes de valor 0 que o antecedem.

Passo 5: Os coeficientes diferentes de zero são codificados de acordo com o código de Huffman. Imagens em cores são consideradas no formato YUV, ou seja, luminância (Y) e crominância (U e V), cujos valores podem ser calculados a partir de seus componentes RGB (*Red-Green-Blue*), conforme as Equações 2.5, 2.6 e 2.7.

$$Y = 0.30R + 0.59G + 0.11B \tag{2.5}$$

$$U = \frac{B - Y}{2} + 0.5 \tag{2.6}$$

$$V = \frac{R - Y}{1.6} + 0.5 \tag{2.7}$$

Os componentes de cor YUV são agrupados e formam o que se define como Minimum Coded Unit (MCU), formato 4:2:0 - luminância com o dobro da resolução das crominâncias MCU - formado com 4Y, 1U e 1V.

A codificação e decodificação de um bloco de textura é ilustrada na Figura 2.3.

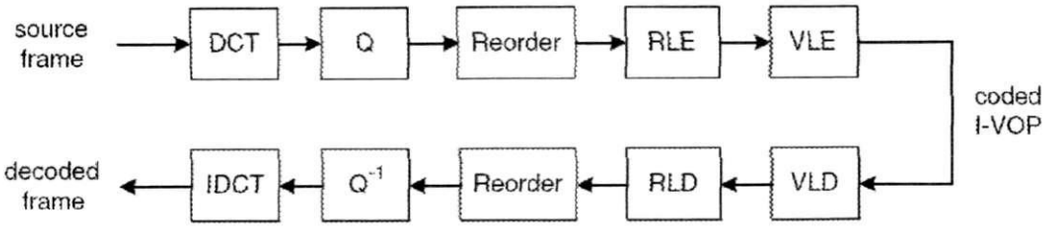

Figura 2.3: Codificação e decodificação de um bloco de textura (bloco I)

2.3.1.2 Modo Inter (Predição e Compensação de Movimento)

É codificada utilizando compensação de movimento de uma imagem I ou P anterior . Esta técnica é chamada predição posterior (forward) de I/P para P. Este modo é similar ao modo Inter da norma H.261.

Tira-se partido da redundância temporal que existe entre imagens consecutivas, existe a necessidade de estimar o movimento que ocorreu entre as imagens consecutivas. A técnica mais utilizada para a estimação do movimento é considerar o movimento por blocos, ou seja, todos os pixels dentro de um bloco obedecem ao mesmo tipo de movimento, um modelo simples para o movimento é considerar somente translações. Assim, para cada bloco é necessário estimar o vetor de deslocamento.

Para a estimação do vetor do movimento não se utiliza a imagem toda mas uma janela de procura.Os algoritmos de estimação do movimento do bloco diferem nos seguintes aspectos:

- Critério de semelhança entre blocos;
- Estratégia de procura;
- Determinação das dimensões do bloco;

A codificação e decodificação de um bloco de predição (bloco P) é ilustrada na Figura 2.6.

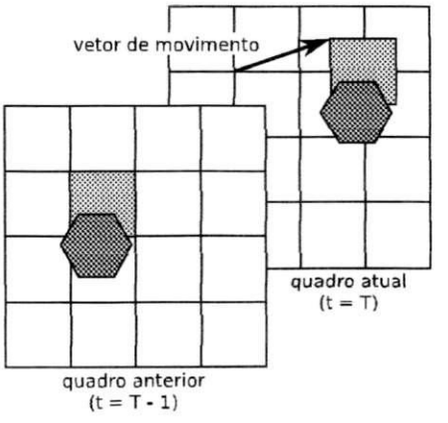

Figura 2.4: Estimação de movimento entre dois quadros de um vídeo

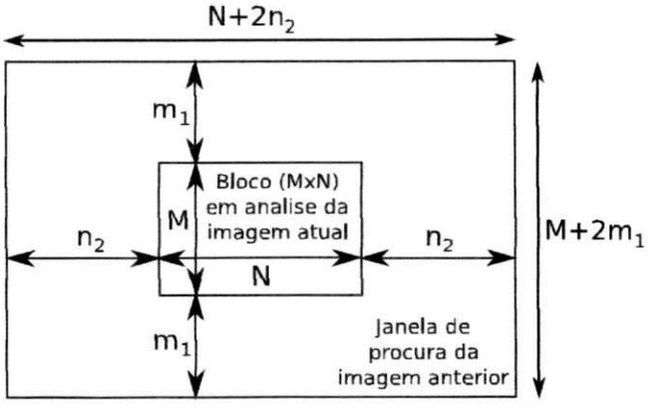

Figura 2.5: Determinação das dimensões do bloco

2.3.2 Perfis e Níveis

O padrão MPEG-4 oferece um grande e rico conjunto de ferramentas para a codificação de objetos audiovisuais. Com o objetivo de permitir implementações efetivas do padrão, subconjuntos dos conjuntos de ferramentas do MPEG-4 Systems, MPEG-4 Visual e MPEG-4 Áudio foram identificados e podem ser utilizados para especificar aplicações. Esses subconjuntos são chamados de perfis (profiles) e limitam o conjunto de ferramentas que um decodificador deve implementar. Para cada um desses perfis, um ou mais níveis devem ser definidos, restringindo a complexidade computacional. A abordagem é parecida com o MPEG-2, cuja combinação perfil-nível mais conhecida é o perfil principal nível principal (Main Profile @ Main Level). Uma combinação perfil nível permite que o codificador implemente apenas o subconjunto do padrão que ele precisa, como permite

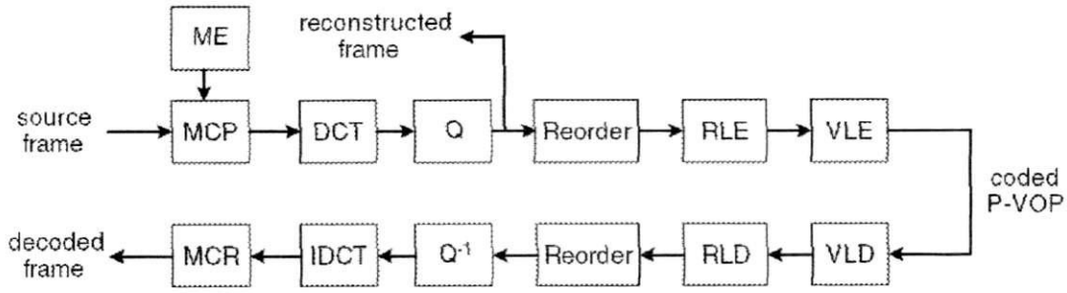

Figura 2.6: Codificação e decodificação de um bloco de predição (bloco P)

também a verificação se os dispositivos MPEG-4 estão de acordo com o padrão. Perfis existem para vários tipos de conteúdo de mídia (áudio, visual e gráfica) e para descrições de cena. O MPEG não prescreve ou aconselha combinações desses perfis, mas existiu um cuidado para que bons casamentos fossem realizados.

Para assegurar a interoperabilidade entre diferentes implementações do MPEG-4, perfis e níveis foram padronizados. Diferentes perfis foram criados para diferentes áreas de aplicação, permitindo aos usuários implementar apenas um subconjunto das ferramentas disponíveis no padrão MPEG-4 e ainda se manterem compliantes. Os níveis definem os limites da complexidade para um perfil particular, como a máxima taxa de bits ou a máxima resolução, por exemplo. Os perfis visuais mais populares são o Perfil Simples (SP) e o Perfil Simples Avançado (ASP).

O Perfil Simples Avançado é um superconjunto do Perfil Simples, contendo ferramentas que realçam a eficiência da compressão.

O perfil simples foi criado para aplicações da baixa-complexidade. As áreas de aplicação incluem serviços moveis de multimídia, vídeos de baixa taxa de bits na Internet ou a gravação de vídeo em chips de memoria.

Tanto o Perfil Simples como o Perfil Simples Avançado definem um conjunto de ferramentas muito similares a aquelas usados nos primeiros padrões MPEG. A codificação de vídeo natural retangular continua utilizando o esquema convencional de codificação hibrida baseada em blocos, mas com novas e melhores ferramentas.

Um resumo das principais características do perfis Simples e Simples Avançado é apresentado na Tabela 1.

O decodificador implementado no projeto do qual o processador aqui relatado fez parte, destina-se a decodificar vídeos MPEG-4 codificados em Simple@L0 (Perfil Simples Nível 0). Conforme pode-se observar na Tabela 1, o Simple@L0 suporta uma definição de imagem máxima tipo QCIF (176x144 pixels) a uma taxa máxima de 15 quadros por

Visual Profile	Level	Session format	Session size	Max. no. of objects	Frame rate (Hz)	Max. video packet length (bits)	Bitrate (kbits/s)
Simple	L0	QCIF	176x144	1	15	2048	64
Simple	L1	QCIF	176x144	4	15	2048	64
Simple	L2	CIF	352x288	4	15	4096	128
Simple	L3	CIF	352x288	4	30	8192	384
ASP	L0	QCIF	176x144	1	30	2048	128
ASP	L1	QCIF	176x144	4	30	2048	128
ASP	L2	CIF	352x288	4	15	4096	384
ASP	L3	CIF	352x288	4	30	4096	768
ASP	L3b	CIF	352x288	4	30	4096	1500
ASP	L4	2CIF	352x576	4	30	8192	3000
ASP	L5	ITU-R601	720x576	4	30	16384	8000

Tabela 2.1: Principais características dos perfis Simples e Simples Avançado

segundo, correspondendo a uma taxa de bits máxima de 64Kbits/s.

A decodificação MPEG-4 será explorada no capitulo seguinte.

Capítulo 3

O Decodificador MPEG-4

Este capítulo descreve resumidamente o processo de decodificação de um vídeo codificado no padrão MPEG-4. Somente as partes relativas ao projeto realizado, isto é, aquelas incluídas no Simple@L0 são descritas. O diagrama de blocos do decodificador em questão é mostrado na A Figura 3.

Conforme exposto no capitulo anterior, o MPEG-4 é um padrão mais avançado do que seu antecessor, o MPEG-2. Além da melhoria nos processos de compressão, que se traduzem em arquivos comprimidos com tamanho menor e sem perda aparente de qualidade, permite também o uso de outros tipos de mídia interagindo com o vídeo (como textos e fotos digitais por exemplo).

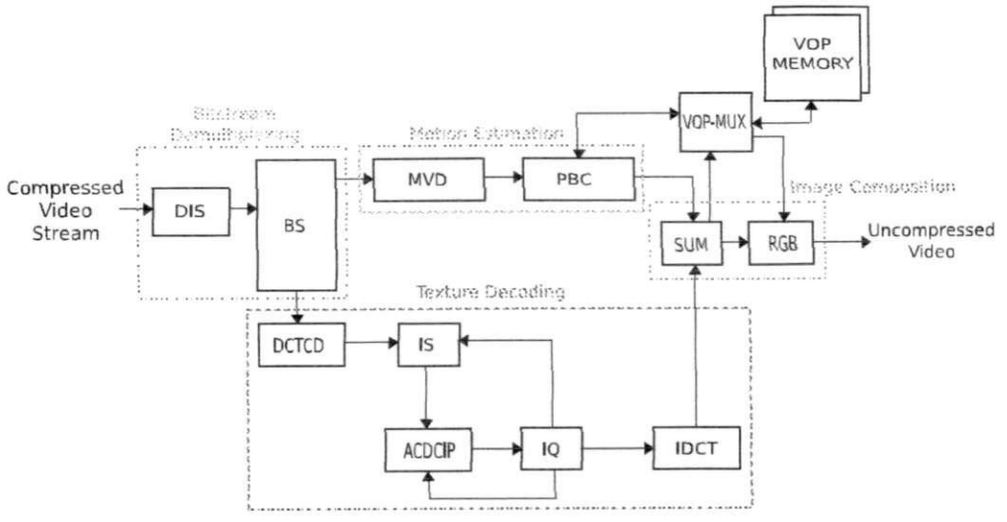

Figura 3.1: Diagrama de blocos de um decodificador MPEG-4

O decodificador e composto de uma parte dedicada a decodificação da textura e de

outra dedicada a decodificação do movimento. Quando o decodificador recebe Mbs (Macro Blocos) codificados tipo "intra"nenhuma informação de movimento é codificada, e o MB é reconstituído a partir da decodificação dos valores de textura de cada pixel. Ao receber MB codificados tipo "inter"os valores decodificados de movimento e de textura dos pixels são somados para formar MB. Em ambos os casos os valores do pixels são saturados dentro do intervalo [0; 255]. Cada VOP codificado inclui um cabeçalho de VOP seguido pelos MBs codificados. Cada MB codificado contem o cabeçalho do MB e dependendo de seus tipo, dados do movimento e da textura. A seções seguintes descrevem brevemente o papel de cada um dos blocos do decodificador MPEG-4.

3.1 Elementos de um Decodificador MPEG-4

As seções seguintes descrevem as principais funções e operações do decodificador MPEG-4. Algumas das funções principais do decodificador são descritas um pouco mais detalhadamente para destacar os aspectos responsáveis por diferenças significativas no desempenho entre diferentes implementações.

3.1.1 Decodificador de Bitstream

Cabe a este bloco receber os dados de vídeo codificados e dar a eles o primeiro tratamento. Este tratamento consiste basicamente em identificar e interpretar as informações dos cabeçalhos, decodificar códigos de comprimento variável (VLC) transformado-os em códigos de comprimento fixo, detectar a presença de erro nos dados, e, na ausência de erro, repassa-los aos seus respectivos blocos de destino.

Em razão de todo este primeiro tratamento realizado por este bloco nos dados codificados, pode-se dizer que sua função é a pré-decodificação do vídeo. Por esse motivo, as referencias seguintes a este bloco o denominarão por "Bloco de Pré-Decodificação".

O Bloco de Pré-Decodificação, foco deste trabalho, será tratado de maneira mais detalhada no capitulo seguinte.

3.1.2 Decodificador de Vetores de Movimento (DVM)

Sua função é realizar o escalonamento, decodificação diferencial (somar mediana de três vizinhos). O bit 'coded' indica se o vetor de movimento diferencial estava codificado no bitstream. Caso não, o DVM deve repetir o último vetor. Caso o bit 'coded' do primeiro bloco de um macrobloco estiver setado, o DVM deve gerar um vetor de movimento zero.

3.1.3 Copiador de Blocos de Predição (CBP)

Sua função é ler do quadro anterior valores de luminância e crominância para compor a próxima imagem, para cada macrobloco. O MVxy de cada bloco de luminância informa onde se encontram os valores de luminância dentro do quadro anterior. O MVxy para valores de crominância deve ser calculado fazendo-se a média dos MVxy de luminância. Se o bit ?intra? estiver setado o CBP deve produzir um macrobloco zerado.

3.1.4 Decodificador dos Coeficientes DCT (DCDCT)

Sua função é a recomposição das carreiras de zeros que foram suprimidas na codificação.

3.1.5 Scan Inverso (SI)

O scan inverso é um algoritmo utilizado para transformar um vetor QFS[n] em uma matriz bidimensional PQF[u][v]. O vetor deve conter os coeficientes em ordem codificada que será transformado em uma matriz ordenada PQF[u][v], em função do tipo de predição atual.

Existem três seqüências de ordenação, ver Figura 7-4 do padrão:

- Zig-zag é usado para macro-blocos intra quando o flag de predição AC é igual a zero e quando se trata de macro-bloco inter.
- Vertical scan é usado para macro-blocos intra quando o flag de predição AC é igual a um e a direção de predição DC é horizontal.
- Horizontal scan é usado para macro-blocos intra quando o flag de predição AC é igual a um e a direção de predição DC é vertical.

3.1.6 Quantização Inversa (QI)

Sua função é realizar o cálculo do fator de escalonamento do coeficiente DC (dcScaler de 6 bits) a partir de quantiserScale, ou seja, decodificar e, portanto, recuperar a matriz de pixels do sinal original.

- Quantização - Multiplicar os valores oriundos da predição DC/AC pelos valores de escalonamento. Existem valores de escalonamento diferentes para DC e para AC.
- Saturação - Faixa de valores entre [-2048, 2047];

3.1.7 Predição Inversa (PIACDC)

É um termo de MPEG para designar uma imagem que é codificada utilizando a predição por compensação de movimento, a partir de uma imagem de referência passada ou futura.

3.1.8 Transformada Discreta Inversa do Cosseno (IDCT)

A transforma do cosseno é uma função que leva os dados da imagem representados em coordenadas cartesianas passem a ser representados no domínio da freqüência. De maneira geral, a compressão ainda não é realizada nesta fase, mas a mesma informação passa a ser representada de uma forma mais apropriada para a compressão.

De fato, a imagem é representada em uma faixa de componentes de freqüência, onde os componentes de freqüência mais elevada denotam bordas 'mais afiadas' e mudanças na imagem, ao passo que componentes de freqüência mais baixa denotam mudanças mais graduais e suaves na imagem.

A Transformada Discreta Inversa do Cosseno, necessária para trazer os dados da imagem novamente para o domínio cartesiano, e reconstruir assim um bloco de imagem, é descrita pela Equação 3.1.

$$f_{DCT}(x,y) = \sum_{x=0}^{7}\sum_{y=0}^{7} \frac{C(x)C(y)}{4} F_{DCT}(i,j) cos\left(\frac{(2i+1)x\pi}{16}\right) cos\left(\frac{(2j+1)y\pi}{16}\right) \quad (3.1)$$

onde $F_{DCT}(x,y)$ são os 64 coeficientes DCT a serem decodificados, $f_{DCT}(i,j)$ são 64 componentes de cor (Y, U ou V) correspondentes a decodificação, $C(x)$ e $C(y)$ são as mesmas constantes da transformada DCT direta, definidas pela Equação 2.2 no capitulo anterior, reapresentada a seguir por conveniência:

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

A presença de somatórios e multiplicações na Equação 3.1 revela a dificuldade de sua implementação de maneira trivial, principalmente em se tratando de soluções em hardware. De fato, a implementação de multiplicadores em hardware consome recursos lógicos consideráveis e, da mesma forma, as implementações cassicas de somatórios costumam ser grandes consumidoras de tempo ou de recursos lógicos. Por essa razão, a implementação em hardware da IDCT tem sido alvo de vários trabalhos, principalmente destinados a descompressão de vídeo.

3.1.9 Somador

Os pixels chegam na mesma seqüência nas duas entradas e são somados diretamente. A soma deve ser colocada no endereço correto para que a memória da imagem atual possa servir mais tarde como memória da imagem anterior ao bloco ?Copiador de Bloco de Predição?.

Simultaneamente ao armazenamento, os pixels de crominância são enviados ao bloco ?Conversor de Cores?. Após o armazenamento de todos os blocos de um macrobloco são gerados endereços para a leitura dos pixels de luminância daquele mesmo bloco para que possam ser enviadas ao bloco ?Copiador de Bloco de Predição?. Esta medida diminui os recursos de armazenamento necessários naquele bloco.

Uma RAM externa ao SOMADOR armazena a imagem corrente. No final do quadro esta RAM é desconectada do SOMADOR e conectada ao CBP.

3.1.10 Conversor de Cores

Seu papel é calcular os valores R, G e B a partir dos valores Y, Cr e Cb de cada pixel.

Primeiro chegam dois blocos de valores de crominância de um macrobloco. Em seguida, chegam os valores de luminância. Quatro valores de luminância enviados em seqüência correspondem a um par de valores de crominância. Os grupos de quatro valores de luminância são enviados numa seqüência correspondente à seqüência dos valores de crominância enviados antes.

3.2 Aspectos Relevantes do Decodificador MPEG4

3.2.1 Resiliência de Erro

Uma das tarefas mais importantes para um decodificador é a sua capacidade de se adaptar e de se recuperar dos erros que inevitavelmente aconteçam, especialmente quando conectado em meios propensos a erro como redes sem fios. A resiliência a erro consiste em três mecanismos diferentes usados quando um erro é descoberto. A detecção pode acontecer de três modos:

- No caso de um erro, em que é descoberto o erro de fato;
- No caso de um erro, em que não é descoberto o erro;
- No caso de nenhum erro, o decodificador "sabe" que não há nenhum erro.

Se o erro é descoberto, o sistema de resiliência de erro do decodificador tenta esconder o erro. Os recursos de resiliência de erro podem ser implementados em um sistema MPEG4 usando o conjunto padrão de ferramentas providas pelo padrão MPEG4 para detectores de erros. Para um sistema ser compatível com o MPEG4, e assim interoperável com outros sistemas MPEG4, ele tem que comportar as ferramentas de detecção de erros unificadas.

As principais ferramentas de resiliência de erro que podem ser usadas serão descritas brevemente a seguir.

3.2.2 Ressincronização

Ferramentas de ressincronização tentam habilitar a ressincronização entre o decodificador e o bitstream após a descoberta de um erro ou uma série de erros. Geralmente, os dados entre o ponto de sincronização antes do erro e o primeiro ponto onde a sincronização é restabelecida, é descartado. Se a aproximação de sincronização é efetiva na localização da quantia de dados descartada pelo decodificador, então a habilidade dos outros tipos de ferramentas, para recuperar dados e/ou esconder os efeitos dos erros, é fortemente ampliada.

3.2.3 Recuperação de Dados

Depois que a sincronização é restabelecida, ferramentas de recuperação de dados tentam recuperar dados que seriam normalmente perdidos. Essas ferramentas não são simplesmente códigos de correção de erro, mas técnicas que codificam os dados de maneira a serem robustas a erros. Por exemplo, uma ferramenta particular que foi empregada pelo Video Group é a de Códigos de Comprimento Variável Reversíveis (RVLC). Nessa aproximação, as palavras do código de comprimento variável são projetadas para que possam ser lidas tanto normalmente quanto de trás para frente.

Um exemplo que ilustra o uso de um RVLC é apresentado na Figura abaixo. Geralmente, em uma situação como esta, na qual um estouro de erros corrompeu uma porção dos dados, todos os dados entre os dois pontos de sincronização, seriam perdidos. Porém, como mostrado na Figura 3.2, um RVLC permite que alguns desses dados sejam recuperados. Deve-se se notar que os parâmetros QP e HEC na Figura 2 representam os campos reservados no cabeçalho do pacote de vídeo para o parâmetro de quantização e a extensão de código de cabeçalho, respectivamente.

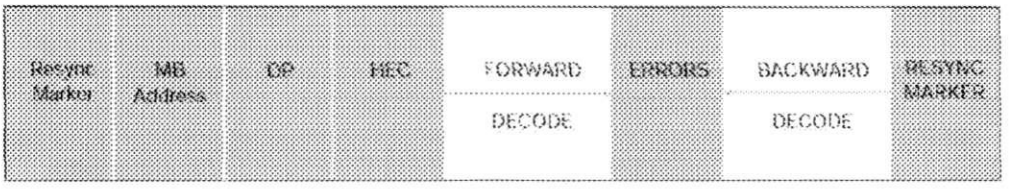

Figura 3.2: Exemplo de um Código de Largura Variável Reversível (RVLC)

3.2.4 Tamanho da Imagem

Como o tamanho de imagem tem um impacto significativo em quase todos os aspectos da solução, inclusive no poder de consumo, exigências de poder de processamento, quantidade de memória, área de silício (em soluções integradas), deve-se observar isto com mais detalhes. Para se ter um ponto de partida para avaliar os tamanhos de imagem, as figuras seguintes descrevem uma mesma imagem em três resoluções comuns:

- SubQCIF
- QCIF
- QVGA

Como pode ser visto nas Figuras 3.3, 3.4 e 3.5 a resolução, é o fator principal na experiência do usuário. Pode ser argumentado que o subQCIF é suficiente para algumas aplicações, porém, está bastante claro que determinando a escolha o usuário é atraído a um dispositivo com uma resolução maior.

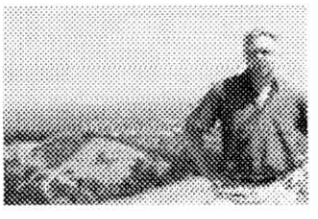

Figura 3.3: Resolução SubQCIF (128 x 96 pixels)

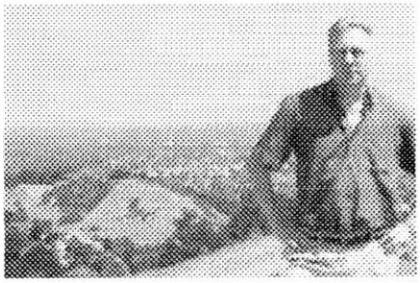

Figura 3.4: Resolução QCIF (176 x 144 pixels)

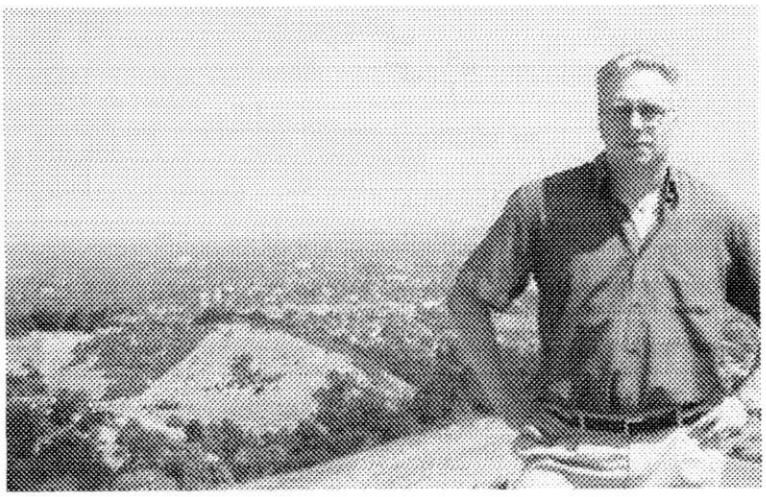

Figura 3.5: Resolução QVGA (320 x 240 pixels)

Capítulo 4

O Bloco de Pre-Decodificação

O bloco de pré-decodificação, assim como alguns outros blocos, esta presente em todos os três decodificadores apresentados. Embora sua complexidade tenha aumentado com a evolução dos padrões, sua funções básicas permaneceram as mesmas: receber o fluxo de bits contendo o vídeo compactado, separar e interpretar as informações dos cabeçalhos e decodificar dados comprimidos em código de comprimento variável (VLC), verificar a existência de erros nos dados, e, na ausência de erro, repassa-los aos seus respectivos blocos de destino. Cabe também ao pré-decodificador a importante tarefa de ressincronização.

E importante notar que este padrão foi desenvolvido visando não somente a reprodução de vídeo a diretamente partir de um meio de armazenamento de dados controlado pelo usuário (tal como o CD, DVD ou HD, por exemplo), mas também para reprodução a partir de difusão em fluxo continuo, como em videoconferências e na TV Digital, por exemplo.

Este capitulo explora um pouco das características dos dados que devem ser tratados pelo pré-decodificador, assim como também um pouco de sua estrutura interna. Ao final é feito um breve resumo de algumas abordagens existentes para sua implementação.

4.1 Características dos Dados de Vídeo

Conforme exposto, os dados contendo o vídeo MPEG-4 compactado são recebidos sequencialmente pelo decodificador. Por esta razão é necessário que o decodificador esteja sincronizado com a recepção dos dados de forma a poder decodifica-los corretamente. Esta exigência é mais facilmente satisfeita quando o vídeo está armazenado num meio controlado pelo usuário, e pouco propenso a erros, tal como num Disco Rígido, por exemplo. Uma vez que o inicio da reprodução do vídeo é comandado pelo usuário, é fácil para o decodificador, conhecendo o inicio da sequência, manter-se sincronizado, desde que não hajam erros. No entanto, em se tratando de meios de difusão em fluxo continuo, o sincro-

nismo entre os dados transmitidos e o decodificador não pode ser assegurado da mesma forma, pois a decodificação pode ter inicio em qualquer momento da transmissão. Uma transmissora de TV, por exemplo, transmite vídeo continuamente, ao passo que os receptores de TV são ligados e desligados de maneira aleatória, por diferentes usuários, e todos devem ser capazes de decodificar o vídeo da mesma forma.

Este ultimo aspecto implica numa importante exigência por parte do padrão: a capacidade de ressincronização. Esta capacidade, é assegurada pelo padrão por 'marcas' inseridas na codificação sob a forma de uma sequência singular e conhecida de bits. Ela garante não só a possibilidade da decodificação poder ter inicio em qualquer trecho do vídeo, como também confere ao decodificador o reforço de outra importante característica: a robustez a erros nos dados.

Caso o decodificador receba dados com erros, tais erros serão completamente eliminados na próxima ressincronização, desde que esta também não contenha erros.

Estas e outras importantes características dos dados de vídeo recebidos pelo pré-decodificador serão tratados nessa seção.

4.1.1 O Pacote de Vídeo

Um VOP transmitido consiste em um ou mais pacotes de vídeo. Um pacote vídeo consiste em um marcador de ressincronização, um campo de cabeçalho e em uma série de macroblocos codificados e ordenados segundo a sequência de varredura do vídeo, conforme a Figura 4.1. O marcador de ressincronização é seguido por um contador do número do próximo macrobloco, que permite ao decodificador posicionar corretamente o primeiro macrobloco do pacote. Em seguidad temos o parâmetro da quantização e um *flag*, o HEC (Header Extension Code - Código de Extensão do cabeçalho). Se o HEC for 1, ele será seguido por uma duplicata do cabeçalho do VOP atual, aumentando a quantidade de informação que deve ser transmitida mas permitindo ao decodificador recuperar o cabeçalho do VOP caso o primeiro cabeçalho do VOP tenha sido corrompido por um erro.

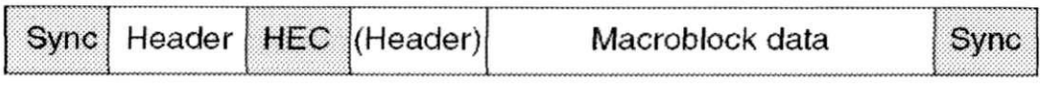

Figura 4.1: Estrutura de um pacote de vídeo

A ferramenta do pacote de vídeo pode ajudar o decodificador na recuperação de um erro de diversas maneiras, como por exemplo:

1. Quando um erro é detectado, o decodificador pode ressincronizar no início do próximo pacote de vídeo, evitando assim que do erro se propague além do limite do pacote vídeo.

2. Se usado, o campo de HEC permite ao decodificador recuperar o cabeçalho do VOP perdido a partir de qualquer outra parte do VOP.

3. A codificação preditiva (tal como a codificação diferencial do parâmetro de quantização, a predição dos vetores de movimento e a predição intra DC/AC) não ultrapassa o limite entre os pacotes vídeo. Isto impede, por exemplo, que um erro nos dados de um vetor de movimento se propague para um outro pacote vídeo.

4.1.2 Particionamento dos Dados

A ferramenta de particionamento dos dados permite ao codificador reorganizar os dados codificados dentro de um pacote vídeo para reduzir o impacto de erros de transmissão. O pacote é dividido em duas partes, a primeira (imediatamente apos o cabeçalho do pacote de vídeo) contem a informação do modo de codificação para cada macrobloco juntamente com os coeficientes DC de cada bloco (para macroblocos intra) ou vetores do movimento (para macroblocos inter). Os dados restantes (coeficientes da AC e coeficientes DC de macroblocos inter) são colocados na segunda parte e em seguida outro marcador de ressincronização.

A informação emitida na primeira parte é considerada como sendo a mais importante para a decodificação correta do pacote video. Se a primeira parte for recuperada, é geralmente possível para o decodificador tentar reconstruir razoavelmente o pacote, mesmo se a segunda parte for danificada ou perdida devido a erro(s) na transmissão.

4.1.3 Código de Supressão de Repetições - RLC

O RLC (Run Length Code) é uma técnica de compressão de dados sem perdas, ou seja a informação codificada pode ser sempre reconstituída exatamente como a original sem qualquer distorção ou perda de informação. Esta técnica consiste na supressão de repetições nos dados através da representação dos trechos repetidos por uma *sequência de escape*, que pode ser simplesmente um símbolo, seguido do numero de repetições do símbolo e finalmente o símbolo em questão. Por esta razão, a codificação RLC é indicada para casos onde exista um grande número de repetições de símbolos em seqüência.

Supondo-se a ocorrência de N repetições do símbolo K, pode-se substituir essa ocorrência representando-a como:

$$< sequncia\ de\ escape >< N >< K >$$

Por exemplo, considere a sequência hexadecimal abaixo (os pontos apenas indicam o limite entre os símbolos ou códigos):

$$"7A.00.00.00.00.00.00" \longrightarrow "7A.FF.06.00"$$

Obviamente a *sequência de escape* deve ser única dentro do código, isto é, não pode haver ambiguidade entre ela e algum outro símbolo ou sequência de símbolos. No exemplo dado, a sequência de escape é o símbolo "FF".

No MPEG-4 a compressão por códigos RLC são empregados tanto nos blocos I (textura) quanto nos blocos P (predição e compensação de movimento), correspondendo aos blocos RLE (Run Length Encoder) e RLD (Run Length Decoder) das Figuras 2.3 e 2.6 do Capitulo 1, antecedendo a compressão por Códigos de Comprimento Variável (VLC), que será discutida na seção seguinte.

Na compressão MPEG existem grandes seqüências de zeros após a leitura em ziguezague da matriz resultante da quantização, por isso a codificação RLE foi simplificada para apenas contar a ocorrência deste símbolo, além disso, como apenas os zeros é que são contados, não é necessário que o símbolo apareça após o número de ocorrências.

4.1.4 Código de Comprimento Variável - VLC

A codificação em Códigos de Comprimento Variável é, assim como o RLE, uma técnica de compressão de dados sem perdas. Ela visa diminuir o comprimento médio das palavra de código, idealizando um código ótimo (isto é, uma atribuição de palavras de código para símbolos) que dependa da probabilidade P com a qual um símbolo é usado. Se um símbolo é emitido com pouca freqüência, a ele é atribuída uma palavra de código longa. Símbolos emitidos mais freqüentemente são representados por códigos mais curtos.

O código de comprimento variável mais conhecido, e também empregado no MPEG-4 é o Código de Huffman. O código de Huffman para um conjunto de símbolos A, B, C, D, E, F e G cujas probabilidades de ocorrência são dadas pela Tabela 4.1 pode ser representado pela Figura 4.2, conhecida por *arvore de Huffman.*

Na Figura 4.2 a probabilidade da ocorrência de cada nó é indicada em seu interior, e os nós externos (ou folhas da arvore) representam os símbolos em questão. O código de Huffman para cada símbolo é obtido a partir da sequência de 0's e 1's associados a cada ramificação da arvore que se deve percorrer a partir da raiz até o símbolo (o código para

A	B	C	D	E	F	G
0.1	0.1	0.1	0.3	0.1	0.1	0.2

Tabela 4.1: Probabilidade da ocorrência de cada símbolo

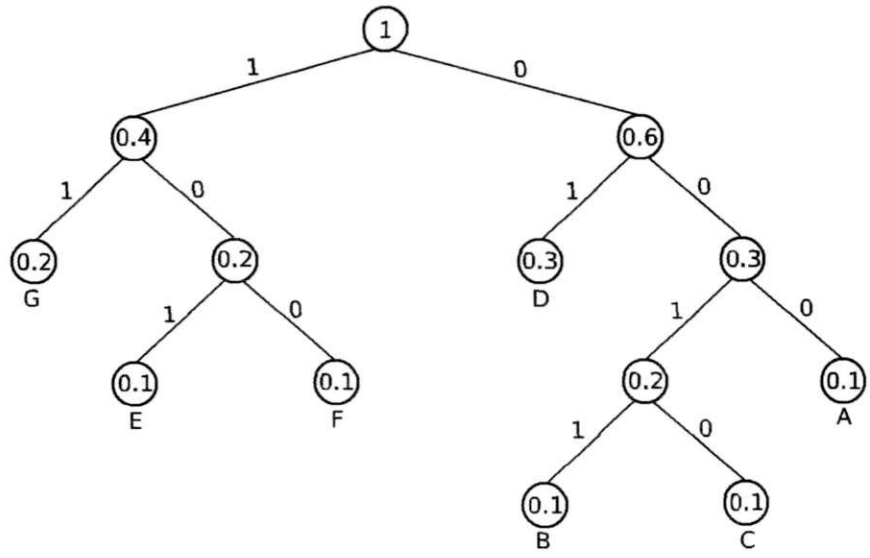

Figura 4.2: Exemplo de uma árvore de Huffman

A é 000 e para D é 01, por exemplo). Para comprimir a sequência "DAF", por exemplo, o codificador produz a palavra binaria "01000100".

Analogamente, para descomprimir palavra binaria "01000100"o decodificador percorre a a arvore de cima para baixo enquanto processa a palavra binaria da esquerda para a direita. Desta forma, a leitura de um "0"faz com que o decodificador escolha o ramo direita enquanto que um "1"leva a escolha do ramo da esquerda. Este processos é repetido até encontrar um nó externo (folha da arvore), que representa um símbolo. Após emitir o símbolo para a saída o processo recomeça a partir da raiz da arvore.

Apos a determinação do código de Huffman para cada símbolo é possível construir uma tabela que permite associar diretamente o símbolo ao seu código correspondente, conforme representado na Tabela 4.2.

No MPEG-4, assim como o RLE, a compressão por VLC também é aplicada tanto nos blocos I quanto nos blocos P, correspondendo aos blocos VLE (Variable Length Encoder) e VLD (Variable Length Decoder) das Figuras 2.3 e 2.6 do Capitulo 1. Em um codificador MPEG-4, a codificação VLC é a ultima etapa de codificação dos dados de vídeo, sendo apenas a eles acrescentados informações tais como cabeçalhos e marcas de sincronismo.

Simbolo	Probabilidade	Código Huffman
A	0.1	000
B	0.1	0011
C	0.1	0010
D	0.3	01
E	0.1	101
F	0.1	100
G	0.2	11

Tabela 4.2: Codigo de Huffman associado a cada símbolo

No entanto, apesar de sua eficiência, a codificação por Huffman torna difícil a correta decodificação em caso de erros nos dados. Isto se deve ao fato das palavras do código não terem tamanho fixo, ou seja, caso ocorra algum erro o tamanho da palavra código pode ser alterado comprometendo assim a decodificação das palavras código seguintes. O exemplo a seguir mostra a sequência de símbolos 'DAFGCB' codificada usando Huffman e decodificada como 'DDDBCB' apos um unico erro no terceiro bit (da esquerda para a direita). Neste caso o erro comprometeu 3 símbolos.

$$\begin{aligned} DAFGCB &\longrightarrow 01.000.100.11.0010.0011 \\ &\qquad 01.01.01.0011.0010.0011 \longrightarrow DDDBCB \end{aligned}$$

Uma solução para minimizar esse tipo de erro, empregada como recurso adicional no MPEG-4, é o uso de Comprimento Variável Reversíveis, conforme será apresentado na seção seguinte.

4.1.5 VLCs Reversíveis

Um conjunto opcional de Códigos de Comprimento Variável Reversíveis (RVLCs) podem ser usado para codificar coeficientes DCT. Como o nome sugere, estes códigos podem corretamente ser decodificados em ambos os sentidos, direto e reverso, tornando possível para o decodificador minimizar a área da imagem afetada por um erro.

Um decodificador decodifica primeiramente cada pacote vídeo no sentido direto e, se um erro for detectado (porque a sintaxe do bitstream foi violada, por exemplo), o pacote é decodificado no sentido reverso, partindo do próximo marcador de ressincronização. Usando esta técnica, os danos causados por um erro podem ser limitados a apenas um macrobloco, tornando fácil esconder a região de erro. O uso da decodificação com resiliência de erro é ilustrado na Figura 4.3.

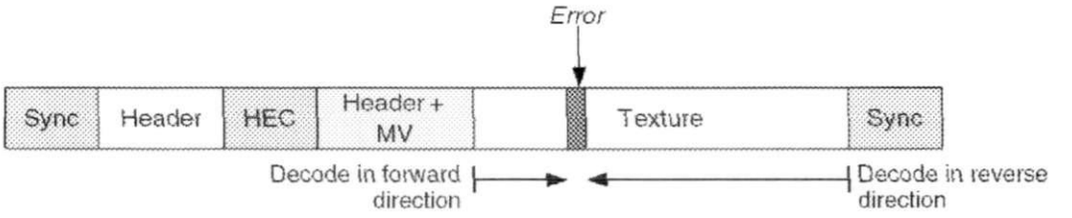

Figura 4.3: Recuperação de erros utilizando RVLCs

Na figura 4.3 temos um pacote vídeo que usa HEC, particionamento dos dados e RVLCs. Um erro ocorre dentro dos dados de textura e o decodificador faz a varredura direta e reversa para recuperar os dados de textura do outro lado do erro.

Capítulo 5

Solução Proposta

5.1 Escolha do Tipo de Abordagem

Conforme foi visto no Capitulo 3 o Pré-Decodificador acumula uma série de funções no decodificador MPEG-4, tendo um papel fundamental não só no repasse de dados decodificados como também na coordenação de vários outros blocos do sistema. De fato, uma parcela significativa da descompressão do vídeo é realizada por este bloco. Por estas razões o Pé-Decodificador é geralmente considerado o 'gargalo' do sistema.

Uma vez que o desempenho de todos os demais blocos do decodificador costuma estar limitado ao do pré-decodificador, a busca por desempenho costuma ser a 'guia mestre' de muitos projetistas para implementa-lo. A preocupação com desempenho leva geralmente a abordagens do tipo 'hard-core' com elevado nível de otimização e eficiência, mas cuja flexibilidade é fortemente penalizada. Todavia, embora esta abordagem surta bons resultados e seja em vários casos a mais adequada, principalmente em decodificadores de elevada taxa de bits (para HDTV, por exemplo), ela implica em uma elevada complexidade de projeto, por vezes incondizente com decodificadores de resolução reduzida.

Por outro lado, abordagens do tipo 'soft-core', que tem como núcleo um processador programado segundo as necessidades em questão, permitem uma notável redução da complexidade do projeto de 'hardware', deixando tarefas complexas e penosas à cargo do software, de natureza mais flexível e simples de se trabalhar. Em contrapartida, este tipo de abordagem não costuma alcançar os mesmos níveis de eficiência de soluções do tipo 'hard-core'.

O decodificador desenvolvido visava a decodificação de vídeos MPEG-4 em Perfil Simples Nivel Zero (Simple@L0) que, conforme exposto anteriormente, trata-se da implementação mais simples do padrão, comportando uma taxa máxima de entrada de video codificado de 64 kbits/s e gerando 176x144 pixels de saída à 15 quadros por segundo.

A abordagem tipo 'software-core' é portanto perfeitamente adequada a este caso. Um diagrama de blocos simplificado da implementação do bloco de pré-decodificação é apresentado na Figura 5.1.

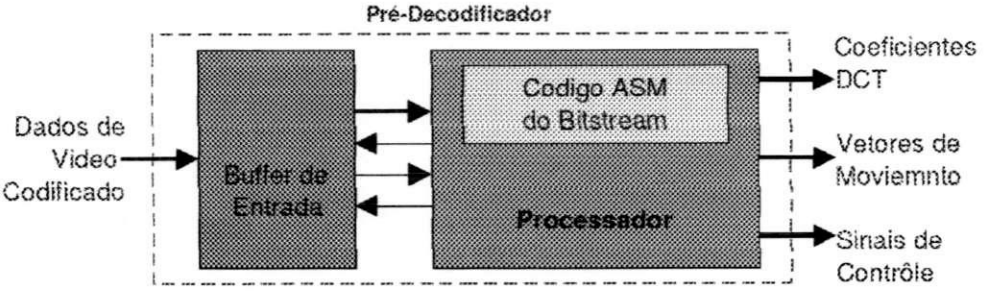

Figura 5.1: Diagrama de blocos simplificado do Pré-Decodificador

A primeira solução que se tentou empregar foi o uso de um processador de código aberto, conhecido por XR16. O XR16 é um processador RISC de 16-bits projetado para rodar apenas programas de base numérica inteira. O projeto do XR16 foi otimizado para uma eficiente implementação em pipeline em FPGAs, contendo um conjunto compacto de instruções de 16 bits, de maneira que a principio poderia atender as especificações necessárias ao processador do pré-decodificador. O diagrama do processador XR16 pode ser visto na Figura 5.2.

No entanto, devido ao fato o código obtido ainda não ter sido sintetizado, o que poderia ser uma fonte de problemas em potencial na etapa de síntese do projeto, e principalmente à sua complexa arquitetura, seu uso foi abandonado, e partiu-se para o projeto e implementação de um processador dedicado.

5.2 Projeto do Processador Dedicado

O projeto de um circuito integrado, parte dele ou protótipo FPGA deve seguir um fluxo preciso cujo objetivo principal é maximizar a confiabilidade do projeto e elevar as chances de sucesso na prototipação. O fluxo simplificado de projeto de um circuito integrado é mostrado na Figura 5.3.

Conforme se avança no fluxo de projeto, descendo-se no diagrama da Figura 5.3, temos uma diminuição no nível de abstração da descrição, o que implica numa elevação na sua complexidade. Ao mesmo tempo, quanto mais descemos no diagrama, mais a descrição do projeto se assemelha ao seu nível final: o protótipo em silício. Uma vez que a perda de abstração é gradual a cada nível que se desce, é possível realizar testes entre níveis de

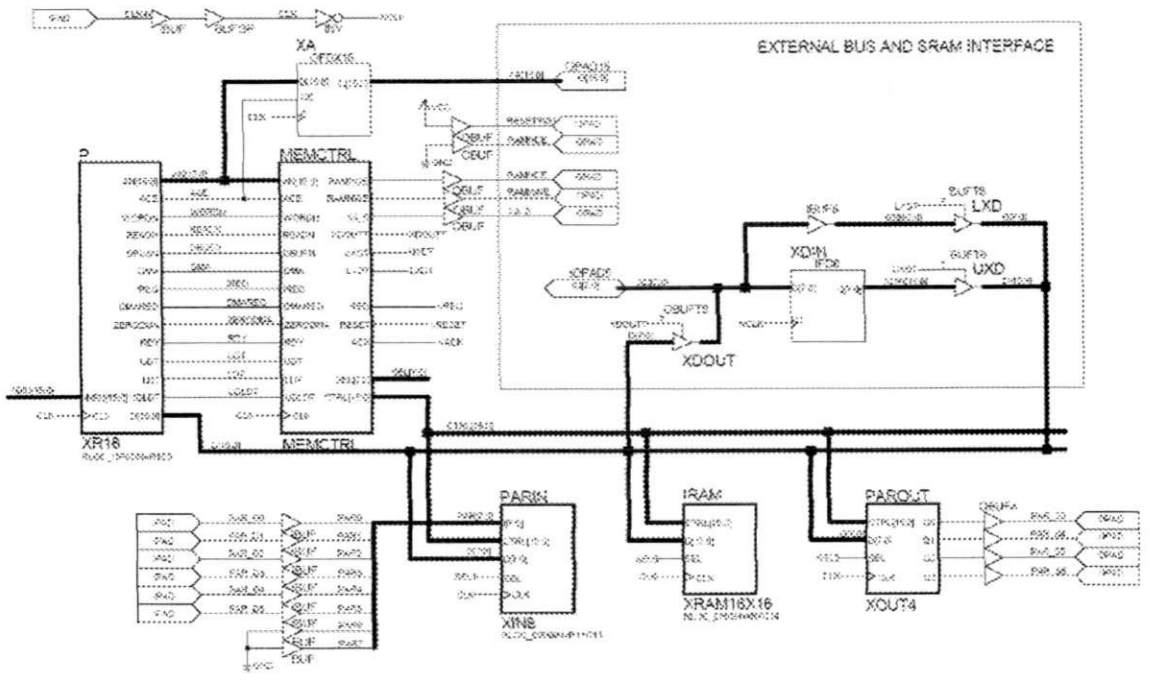

Figura 5.2: Diagrama do Processador XR16

descrição consecutivos, de forma a permitir verificação de que a descrição menos abstrata ainda corresponde a descrição anterior, mais abstrata. Os tipos de teste possíveis de serem feitos entre cada nível de descrição é indicado lado direito do diagrama.

Um detalhe importante deste fluxo de projeto é a capacidade de automatização. Esta capacidade é tanto maior quanto menor é o nível de abstração. Enquanto a Especificação e sua tradução para o nível mais baixo (a Descrição Comportamental) são feitas quase que inteiramente sem o auxilio da maquina, as etapas finais como a síntese da Descrição Estrutural para a Descrição em Layout é quase que totalmente automatizada. Esta característica é altamente louvável pois, como se sabe, os níveis de menor abstração são mais complexos e consequentemente mais críticos, propensos a erros e de lenta implementação manual. Desta forma, a redução da participação humana torna o fluxo não só mais rápido como também mais confiável.

Na fase inicial, de especificação, o projeto é especificado em elevado nível de abstração, isto é, descrevendo-se apenas o que o ele deve fazer e a quais restrições deve atender, sem se preocupar no 'como fazer'. A especificação é tipicamente feita em linguagem natural (português, inglês, francês, etc.).

A Descrição Comportamental por sua vez, possui um nível de abstração mais baixo

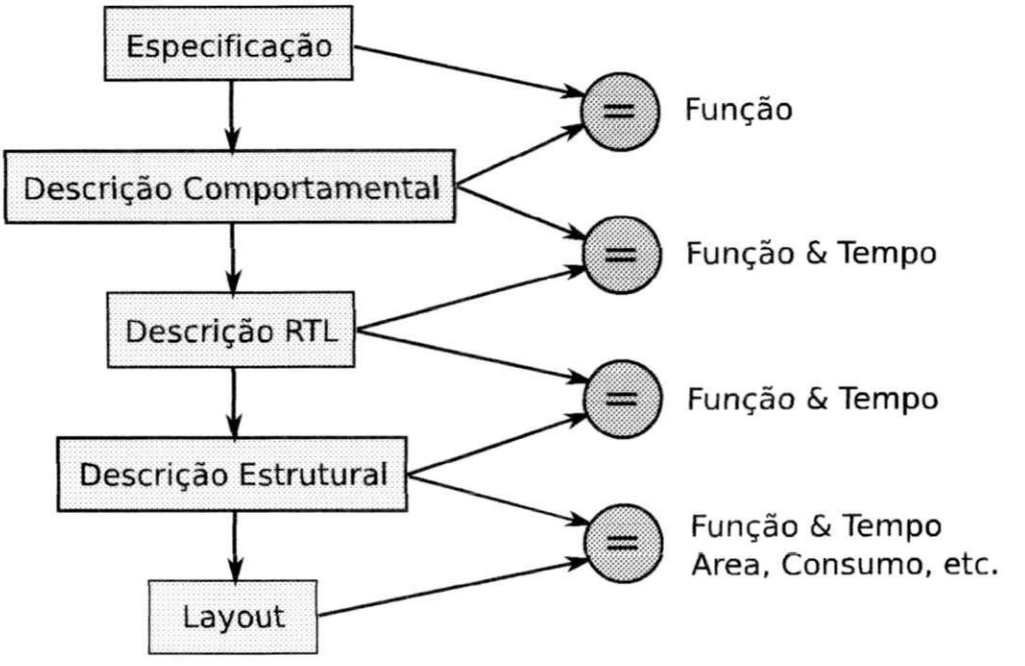

Figura 5.3: Fluxo simplificado de projeto de um circuito integrado

que a especificação (e portanto mais complexa), mas ainda é mais abstrata que a Descrição RTL. Nesta etapa é descrito como o projeto deve ser feito, e geralmente esta descrição pode ser representada por uma linguagem de alto nível, como C, por exemplo.

A Descrição RTL (Register Transfer Level) é uma das etapas mais importantes do projeto. Nela o projeto descrito pelas etapas anteriores passa a ser descrito por uma linguagem de descrição de hardware (HDL - Hardware Description Language). O projeto realizado utilizou a linguagem System-C, que será apresentada no capitulo seguinte.

Descrição Estrutural é geralmente obtida de forma automática por ferramentas síntese. Neste nível de descrição o sistema é representado por meio de blocos lógicos elementares interligados tais como registradores e portas logicas. Esta descrição pode ser feita tanto visando a prototipação em FPGA quanto em silício.

O Layout é nível mais baixo de abstração de um projeto VLSI. Neste nível, o sistema é descrito tal qual será implementado em silício, isto é, por meio de transistores e fios precisamente posicionados e dispostos em camadas no chip. Embora ainda seja possível se descer até o nível de descrição físico-química do circuito integrado, isto na pratica não costuma ser abordado pelos projetistas VLSI, cabendo quando necessário a competência de uma outra equipe especializada.

O restante deste capitulo limita-se a apresentar a especificação e a descrição compor-

tamental do processador, cabendo aos capítulos seguintes tratar da Descrição RTL e da verificação.

5.2.1 Especificação

Conforme foi visto, a primeira etapa na elaboração de projeto VLSI é a Especificação. No caso de processador dedicado isto implica em identificar exatamente a aplicação a qual ele se destina e então elaborar uma lista de características a serem atendidas pela etapa seguinte (a Descrição Comportamental).

- Conjunto de instruções reduzido;
- Arquitetura simples e flexível;
- Garantir a entrega dos dados de saída;
- Capacidade de operar o mais independentemente possível dos blocos a ele ligados.

Ter um conjunto de instruções reduzido, característica marcante dos processadores RISC, implica não só numa maior simplicidade de implementação como também propicia arquiteturas mais eficientes, evitando-se, por exemplo, consumir área de silício, tempo de projeto (por vezes também problemas de temporização), com instruções pouco utilizadas e que podem ser substituídas pela combinação de outras. Esta característica também aumenta a facilidade de codificação de programas para o processador, já que um menor numero de instruções precisa ser assimilado pelo programador.

A arquitetura simples e flexível garantem a capacidade de se poder realizar alterações futuras como aumentar ou diminuir o numero de portas de saída ou criar novas instruções, por exemplo. A simplicidade também costuma ser um bom requisito para se alcançar um bom desempenho.

Assim como foi visto nos capítulos anteriores, o decodificador MPEG-4 é formado por diversos blocos, alguns dos quais possuem algum tipo de comunicação com o pré-decodificador. Cada um destes blocos possuem tempos e ritmos de operação distintos, e muitas vezes inconstantes, de maneira que um determinado dado colocado em uma porta do microprocessador pode ainda não ter sido lido pelo bloco a qual se destinava, quando o dado subsequente precisar ser escrito na porta. Por esta razão, o microprocessador deve garantir de alguma forma a entrega de dados escritos nas suas portas de saída aos seus respectivos blocos de destino.

Esta ultima exigência pode levar o microprocessador a se tornar temporalmente dependente dos blocos a ele conectados, ou seja, no caso de um bloco 'lento' conectado a uma porta de taxa mais elevada o microprocessador deve aguardar até que o dado atual seja

lido antes de disponibilizar o dado seguinte na porta. Contudo, isto não deve bloquear as demais atividades do microprocessador.

5.3 Arquitetura

A analise das especificações do microprocessador juntamente com o algorítimo do bloco de pré-decodificação definido pela norma ISO/IEC 14496-2 permitiu definir-se o conjunto mínimo de instruções necessário para sua implementação. As instruções foram divididas em 7 categorias como:

- Comparação
- Desvio condicional e incondicional
- Manipulação de flags
- Escrita em porta
- Escrita e leitura em memoria
- Decremento
- Manipulação binaria da fila de entrada

A própria definição das categorias de instrução levou a necessidade de criação de uma memoria de dados. Uma simples analise revela que a memoria de dados e de programa possuem natureza distintas. Uma vez definido o programa de implementação do pré-decodificador a ser executado pelo microprocessador, este não precisara mais ser alterado. Isto sugere claramente a utilização de uma memoria ROM (Read Only Memory). A memoria de dados por sua vez precisa ser escrita e lida de forma aleatória, o que sugere uma memoria do tipo RAM (Ramdom Access Memory). Ao mesmo tempo, temos que a profundidade de bits ótima para cada memoria é diferente. Enquanto uma profundidade de 8 bits por palavra é considerada ótima para a memoria de programa, que contém instruções curtas, a memoria de dados deve ter 32 bits de profundidade para armazenar valores de forma eficiente (sem múltiplas leituras de memoria para um único dado).

A ideia de se ter a memoria de dados e de programa separadas uma da outra torna-se obvia. Resta então escolher o modelo de memoria: Havard ou Von-Neumann. A diferença esta na forma como os dados e o programa são armazenados pelo microprocessador. Na arquitetura Von-Neumann existe apenas um barramento interno por onde passam as instruções e os dados. Já na arquitetura Havard existem dois barramentos internos, um para dados e outro para instruções.

Na arquitetura Von-Neumann as memorias de dados e programa podem estar separadas fisicamente em dois dispositivos distintos ou contidas num mesmo dispositivo e separadas logicamente, o que é feito geralmente pela definição de uma região de dados e outra de programa, delimitadas por um endereço. A arquitetura Havard por sua vez implica quase que obrigatoriamente em dispositivos distintos.

A possibilidade de buscar a próxima instrução na memoria enquanto outra está sendo executada, garantida pela arquitetura Havard foi decisiva na sua escolha, permitindo-se alcançar desempenhos mais elevados no processamento.

Dois registradores de uso geral também foram criados sendo um de 8 bits (Ab) e outro de 16 bits (Aw), bem como flags de uso especifico (como o flag de zero, por exemplo) e de uso geral.

O ultimo ponto de decisão foi o tipo de arquitetura da maquina de execução de instruções: hardwired ou por microcódigo. Na arquitetura hardwired as instruções são executadas diretamente por 'logica fiada', isto é, por circuitos compostos por elementos básicos como portas logicas e registradores, especialmente projetados para cada instrução, ou, de forma mais abstrata, dizemos que cada instrução é implementada por uma maquina de estados dedicada. Na arquitetura baseada em microcódigo a execução de cada instrução é vinculada a leitura de uma pequena memoria, chamada memoria de microcódigo, onde parte do barramento de dados é associado aos sinais de controle da máquina (também conhecido como barramento de controle), e a outra parte a um registrador que aponta para próximo endereço de a ser lido. Desta forma a implementação de cada instrução resume-se simplesmente a programar a memoria adequadamente.

A implementação por microcódigo foi escolhida por atender melhor o requisito de simplicidade e flexibilidade, já que a adição ou remoção de qualquer instrução pode ser feita diretamente pela re programação da memoria de microcódigo.

A garantia da entrega dos dados de saída foi assegurada através de *handshakes* em cada porta de comunicação. O handshake é um protocolo extremamente simples que permite ao sistema transmissor ter a garantia de que os dados foram recebidos pelo sistema receptor, simplesmente através da adição de dois sinais de controle: pronto (*ready*) e válido (*valid*). Quando o sistema receptor está pronto para receber dados ele coloca o sinal "pronto" em nível logico 1. Tão logo o sistema transmissor estiver pronto ele disponibiliza o dado na saida e coloca o sinal "válido" em nível logico 1, sinalizando que o dado pode ser lido. Ao receber sinal de "válido" o receptor lê o dado e coloca o sinal "pronto" em nível logico 0, sinalizando que recebeu o dado. O transmissor entao coloca o sinal "válido" em nível logico 0 respondendo ao receptor que recebeu sua sinalização. Uma ilustração do protocolo handshake é mostrada na figura 5.4.

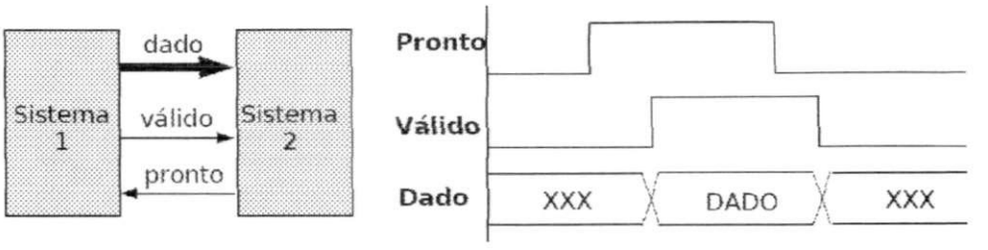

Figura 5.4: Protocolo handshake

Além disso, as portas de comunicação foram agrupadas em um subbloco capaz de operar quase que independentemente restante do microprocessador, sendo responsável pelo hand-shake de cada porta e por atribuir a cada uma delas uma FIFO's (First In First Out), evitando assim que o processador pare à espera de que algum bloco "consuma" o dado em uma porta antes de disponibilizar o novo dado. Desta forma, se o microprocessador precisar escrever em uma porta cujo dado atual ainda não tiver sido lido este dado será colocado na FIFO, ali ficando até que a porta seja liberada. A capacidade da FIFO pode então ser dimensionada de maneira que nunca esteja cheia no momento de escrita pelo microprocessador.

Um diagrama de blocos simplificado do microprocessador projetado é apresentado na Figura 5.5

5.4 Conjunto de Instruções

Esta seção descreve resumidamente cada uma das funções implementadas no microprocessador. A nomenclatura utilizada é apresentada a seguir:

imm	argumento imediato da instrução
f	numero de flag
addb	endereço de 8 bits
addw	endereço de 16 bits
imm	argumento imediato da instrução

5.4.1 Comparação

CMP n imm

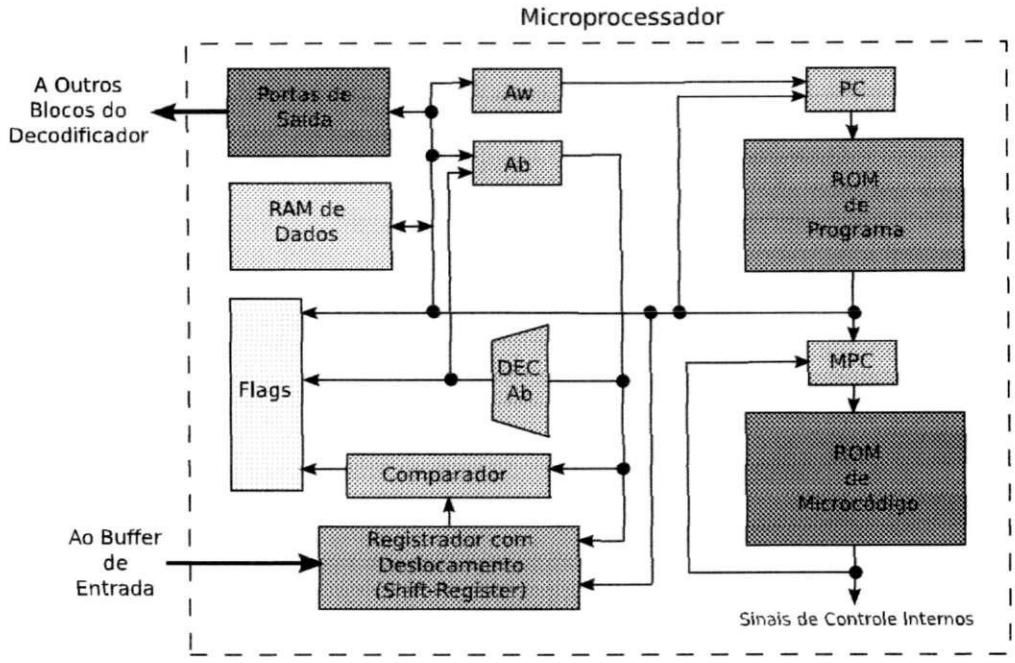

Figura 5.5: Diagrama de blocos simplificado do microprocessador

É utilizado para comparar um determinado numero de bits (n) do bitstream com um valor pré-determinado (imm). O resultado da comparação fica no flag (0).

5.4.2 Desvio condicional e incondicional

JMPs f addw

Salta para um endereço específico se o flag (f) estiver setado. Flag (1) sempre está setado.

JMPn f addw

Salta para um endereço específico se o flag (f) estiver setado.

JMPa

Pula para o endereço de código armazenado dentro do registrador Aw.

5.4.3 Manipulação de flags

SET f

Seta o valor lógico 1 no flag especificado (f).

CLR f

Seta o valor lógico 0 no flag especificado (f).

5.4.4 Escrita em porta

OUTb imm addb

Disponibiliza no endereço especificado (addb) o valor pré-determinado (imm) de 8 bits. O endereço add=0 corresponde ao registrador Ab.

OUTw imm addb

Disponibiliza no endereço especificado (addb) o valor pré-determinado (imm) de 16 bits. O endereço addb=0 corresponde ao registrador Aw.

OUTd imm addb

Disponibiliza no endereço especificado (addb) o valor pré-determinado (imm) de 32 bits.

OUTx addb

Disponibiliza no endereço especificado (addb) o valor composto por um numero de bits do bitstream. A quantidade de bits depende do valor armazenado no registrador Ab.

5.4.5 Escrita e leitura em memoria

STOb imm addb

Armazena no endereço especificado (addb) o valor pré-determinado (imm) de 8 bits. O endereço addb=0 corresponde ao registrador Ab.

STOw imm addb

Armazena no endereço especificado (addb) o valor pré-determinado (imm) de 16 bits. O endereço addb=0 corresponde ao registrador Aw.

STOd imm addb

Armazena no endereço especificado (addb) o valor pré-determinado (imm) de 32 bits.

LDAb addb

Carrega o registrador Ab com valor armazenado no endereço (addb).

5.4.6 Decremento

DECz

Decrementa valor no registrador Ab e seta flag 0 se zero.

5.4.7 Manipulação binaria da fila de entrada

DIS n

Descarta número(n) de bits do bitstream.

DISa

Descarta um número de bits do bitstream. A quantidade de bits que são descartados depende do valor armazenados no registrador Ab.

5.4.8 Código exemplo

São apresentadas a seguir duas listagens correspondentes a um trecho do algoritimo de implementação do pré-decodificador do MPEG-4, especificados pela norma ISO/IEC 14496-2. A primeira listagem (5.1) foi retirada diretamente da norma ao passo que a segunda listagem (5.2) é a implementação do mesmo algorítimo utilizando o assembly do microprocessador projetado.

Listagem 5.1: Código de Exemplo da ISO 14496-2

```
1  VisualObjectSequence() {
2    do {
3        Visual_object_sequence_start_code
4        profile_and_level_indication
5        while (next_bits() == user_data_start_code) {
6          user_data()
7        }
8        VisualObject()
9    } while (next_bits() != visual_object_sequence_end_code)
10   visual_object_sequence_end_code
11 }
```

Listagem 5.2: Código Equivalente em Assembly do Processador

```
1 inicio:
2         cmp   32,VOSSC
```

```
3           jmps 0,certo
4           dis  1
5           jmps 1,inicio
6   certo:
7           dis  32;
8           cmp  8, 55h;
9           jmps 0,achou;
10          jmps 1,inicio;
11  achou:
```

Capítulo 6

Implementação

Neste capitulo é apresentado um pouco a respeito da metodologia e ferramentas utilizadas na fase de implementação do pré-decodificador.

6.1 Metodologia Utilizada

A metodologia utilizada para concepção e implementação do IP´s está baseada em modernas técnicas de projeto:

A especificação dos IP´s foi realizada em SystemC e a metodologia de verificação funcional utilizada é baseada em testbenches com vários tipos de testes entre os quais testes corner-case, compliance e randômicos.

A síntese dos IP´s foi realizada utilizando ferramentas comerciais (Co-centric e FPGA Compiler da Synopsys), e a prototipação em FPGA foi feita utilizando a placa de desenvolvimento Stratix-II da Altera.

6.2 Técnicas de Prototipação em FPGA

Aborda-se aqui o processo de validação em hardware do decodificador MPEG4. A prototipação em FPGAs de hardware pode ser realizada por diferentes processos. Visando evidenciar o uso de SystemC como linguagem de partida para a prototipação de hardware, mostram-se aqui vários fluxos possíveis.

Primeiro, pode-se partir de SystemC e realizar a síntese lógica usando ferramentas Synopsys, seguido de síntese física com ferramentas específicas do vendedor de FPGAs. Pode-se também empregar ferramentas Synopsys para a síntese física, desde que dispondo das bibliotecas específicas do vendedor de FPGAs instaladas no ambiente Synopsys. Outra possibilidade é usar ferramentas Synopsys apenas para traduzir o código SystemC

para HDL independente de dispositivo, entrando a partir daí com o HDL gerado em um ambiente de CAD específico, tal como o Quartus da Altera.

A Figura 6.1 ilustra um conjunto de fluxos possíveis, com exemplos de ferramentas empregadas em cada passo. Entre cada dois passos consecutivos, o formato de comunicação de informações é via uma linguagem textual, seja de alto nível como HDLs e/ou SystemC, seja de baixo nível como EDIF.

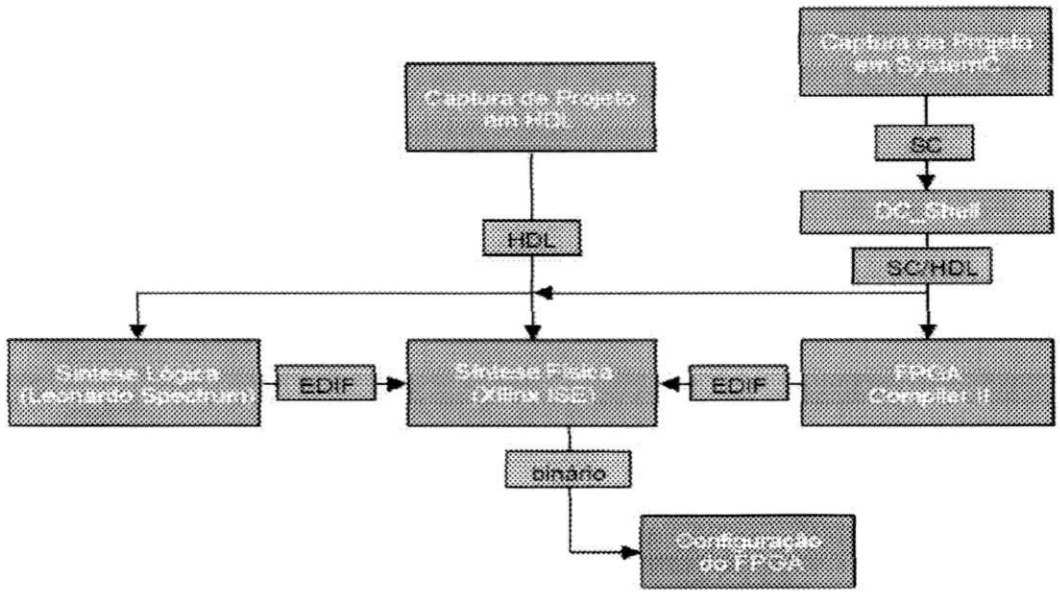

Figura 6.1: Possibilidades de composição de fluxos de projeto para prototipação

6.3 A Linguagem SystemC

O SystemC foi a linguagem de descrição de Hardware adotada pelo projeto. Ela pode ser usada para a modelagem de hardware em diversos níveis de abstração. No presente já é possível empregar descrições SystemC como descrição de entrada para fluxos automatizados de geração de hardware, desde que restrinja-se as construções de SystemC empregadas a um subconjunto dito sintetizável. Este subconjunto é denominado aqui de SystemC no nível de transferência entre registradores ou SystemC RTL (Register Transfer Level). A ferramenta principal para processar SystemC RTL usada neste documento é o CoCentric SystemC Compiler da empresa Synopsys, que permite realizar a síntese lógica de SystemC RTL para formatos EDIF ou HDL, ou apenas traduzir SystemC RTL para código HDL RTL (nas linguagens Verilog ou VHDL).

Como algumas das técnicas de projeto de hardware a partir de SystemC envolvem a passagem por código HDL intermediário, além do CoCentric SystemC Compiler emprega-se também ferramentas de síntese lógica como Leonardo Spectrum da Mentor Graphics e/ou Quartus-II da Altera. Para a prototipação de hardware a partir de código SystemC RTL cita-se o uso do ambiente ISE da Xilinx.

As técnicas de prototipação pressupõem descrições corretas. Para validar descrições SystemC RTL antes da prototipação, técnicas de validação funcional devem ser empregadas. Estas serão abordadas no capitulo seguinte.

Capítulo 7

Verificação

7.1 Verificação Funcional

Consiste basicamente em confrontar um modelo a ser verificado a outro modelo padrão, comparando a funcionalidade. A verificação funcional pode ser realizada a vários níveis:

- componente/unidade/sub-unidade, ...
- ASIC/FPGA/IP
- Sistema/SOC
- placa

Uma característica importante da verificação funcional é que ela pode provar a presença de erros, mas não pode provar a ausência de erros. Existem basicamente três tipos de abordagens possíveis na verificação de um modulo (DUV - Design Under Verification), são eles:

- Black Box
- Grey Box
- White Box

As principais características do DUV tipo Black Box, é que podem ser "vistas do lado de fora" apenas as entradas, saídas do modulo. Cada função implementada foi bem documentada, uma vez que para verificar, é preciso entender a função e prever as saídas sabendo as entradas. No modelo tipo White Box todas as variáveis internas são visíveis e

podem ser acessadas para verificação. Pode ser utilizado para teste de unidades pequenas nas folhas da hierarquia. Em um modelo do tipo Grey Box apenas uma seleção restrita de variáveis internas pode ser usada para verificação, como por exemplo os registradores de um processador. Testbench

Um testbench consiste numa montagem de teste para simulação. Ele cria estímulos e verifica as respostas, não tendo entrada nem saída, podendo ser visto como um modelo do universo em volta do projeto. Ele deve imprimir mensagens quando o DUV apresenta comportamento inesperado, e caso tudo esteja ok imprime uma única menagem no final. No caso deste projeto, convencionou-se que o código deveria ser escrito em SystemC.

A Figura 7.1 apresenta o modelo de testbench adotado pelo Projeto Fenix, com seus elementos internos

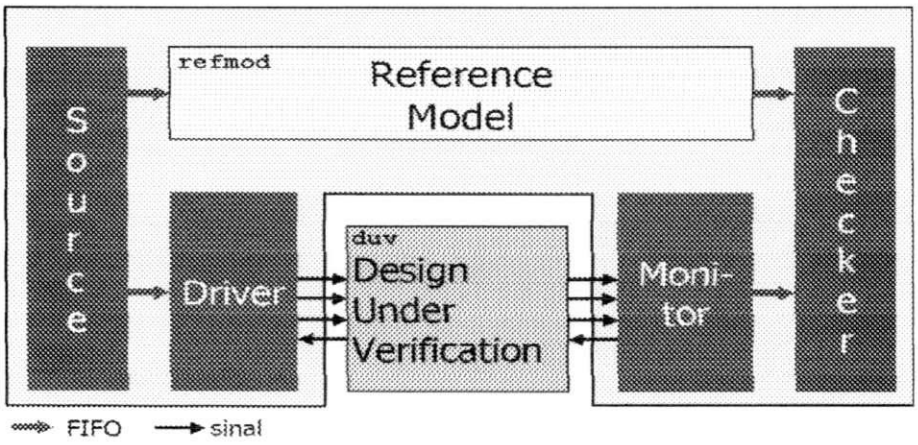

Figura 7.1: Modelo de testbench adotado no projeto

7.2 Elementos de um Testbench

Conforme pode-se ver na Figura 7.1, o testbench adotado é constituído de cinco elementos. São eles:

Source: Envia transações de entrada para o driver e o modelo de referencia.

Checker: Compara as transações de saída recebidos do monitor com as um modelo de referência. É desejável que seja reutilizável, ou seja, dependa pouco do DUV.

Modelo: de Referência - É o modelo de referencia do DUV a ser testado. É tipicamente timeless.

Driver: Recebe transações de entrada e os converta em transições de sinais da interface de entrada do DUV.

Monitor: Observa os sinais da interface de saída do DUV, implementa o protocolo de sinalização e gera transações de saída que repassando-os ao Checker.

Deve-se notar que o testbench para validação funcional abstrai o conceito de tempo, verificando apenas a compatibilidade entre as saídas produzidas pelo Modelo de Referencia e o DUV, para os mesmos estímulos. Tal abstração é feita através do uso de transações. Uma transação é uma operação que inicia num determinado momento no tempo e termina em outro, sendo caracterizada pelo conjunto de instruções e dados necessárias para realizar a operação.

Capítulo 8

Prototipação

Seguindo o fluxo de projeto apresentado no Capitulo 5, apos a especificação, descrição comportamental, descrição RTL, e respectivas verificações, cuja metodologia foi descrita no Capitulo 7, chegamos finalmente a fase de prototipação em FPGA. Nesta etapa todo o código HDL verificado e corrigido é então sintetizado sob a forma de um complexo arquivo contendo o codigo de configuração da FPGA (Field Programmable Gate Array).

Uma FPGA é um dispositivo semicondutor composto basicamente por três tipos de componentes: blocos de entrada e saída (IOB), blocos lógicos configuráveis (CLB) e chaves de interconexão (Switch Matrix). Cada CLB pode ser programado de forma a implementar pequenos circuitos lógicos, e cada IOB é conectado a um pino externo da FPGA, podendo implementar um pino de entrada, saída, bi-direcional ou mesmo um pino não conectado (alta impedância). As chaves de interconexão são capazes de conectar CLB's e IOB's formando então o sistema completo. O diagrama representativo de uma FPGA e seus componentes básicos (IOB's, CLB's e Switch Matrix), é mostrado na Figura 8.1.

Uma vez que todos os três componentes básicos da FPGA são configuráveis, ela é capaz de implementar literalmente qualquer sistema digital, desde que seu tamanho não ultrapasse a capacidade da FPGA e sua complexidade nao esgote a quantidade de recursos internos disponíveis. Por esta razão, as FPGA's são largamente utilizadas como meio de prototipagem de circuitos digitais, para valida-los antes de sua implementação em silício, ou mesmo como componentes de uso final.

No caso do microprocessador desenvolvido, sua prototipação foi feita juntamente com o sistema do qual ele fez parte, o decodificador de vídeo MPEG-4. A prototipação foi feita utilizando como base a placa de desenvolvimento "Nios II Development Kit, Stratix II Edition" da Altera, mostrada na Figura 8.2.

Desta forma, a validação do decodificador MPEG-4 em FPGA valida automaticamente também o microprocessador, mesmo sabendo-se que durante uma boa parcela do projeto

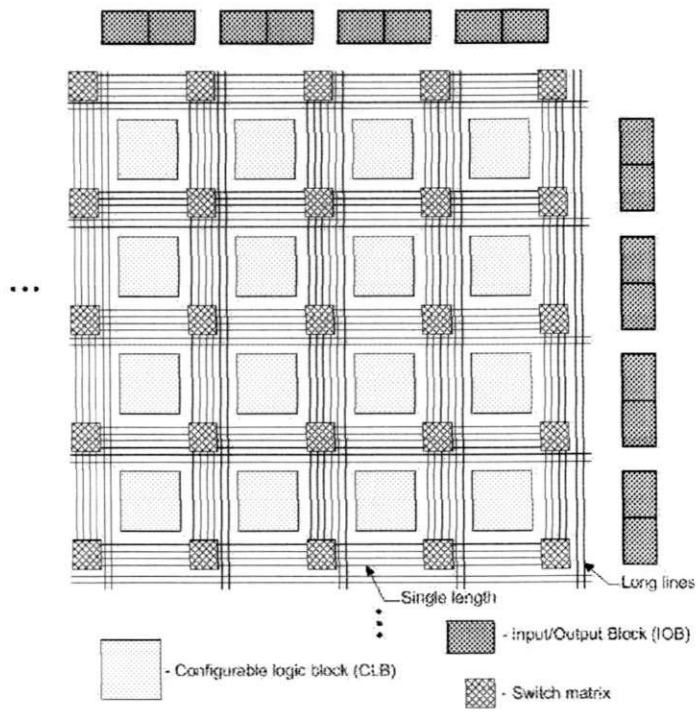

Figura 8.1: Diagrama representativo de uma FPGA e seus componentes básicos

o microprocessador foi desenvolvido individualmente do restante do sistema.

Como todo o sistema já havia sido exaustivamente simulado e corrigido antes da fase de prototipação, a possibilidade de erros no funcionamento do protótipo foram bastante minimizadas. De fato, uma verificação bem feita não só minimiza as chances de erro como também reduz suas origens quase que totalmente a problemas de temporização.

Apos prototipado o sistema passou a decodificar vídeos corretamente, funcionando perfeitamente como planejado, exceto por quatro pequenas manchas na área de vídeo. Tais manchas persistiam independentemente do vídeo decodificado, e custaram para a equipe de desenvolvimento três meses de depuração para poderem ser removidas. A causa, assim como esperado, foi um pequeno problema de temporização causado por uma memoria interna da FPGA que não operava na velocidade especificada pelo fabricante. As fotos do sistema prototipação em fase de depuração podem ser vistas nas Figuras 8.3 e 8.4. A figura 8.5 mostra em detalhe o vídeo corretamente decodificado e sem manchas.

Com a validação do sistema, através da prototipação bem sucedida, partiu-se para a implementação em silício do decodificador MPEG-4 (incluindo o microprocessador desenvolvido). O leiaute final da prototipação do decodificador MPEG-4 em circuito integrado

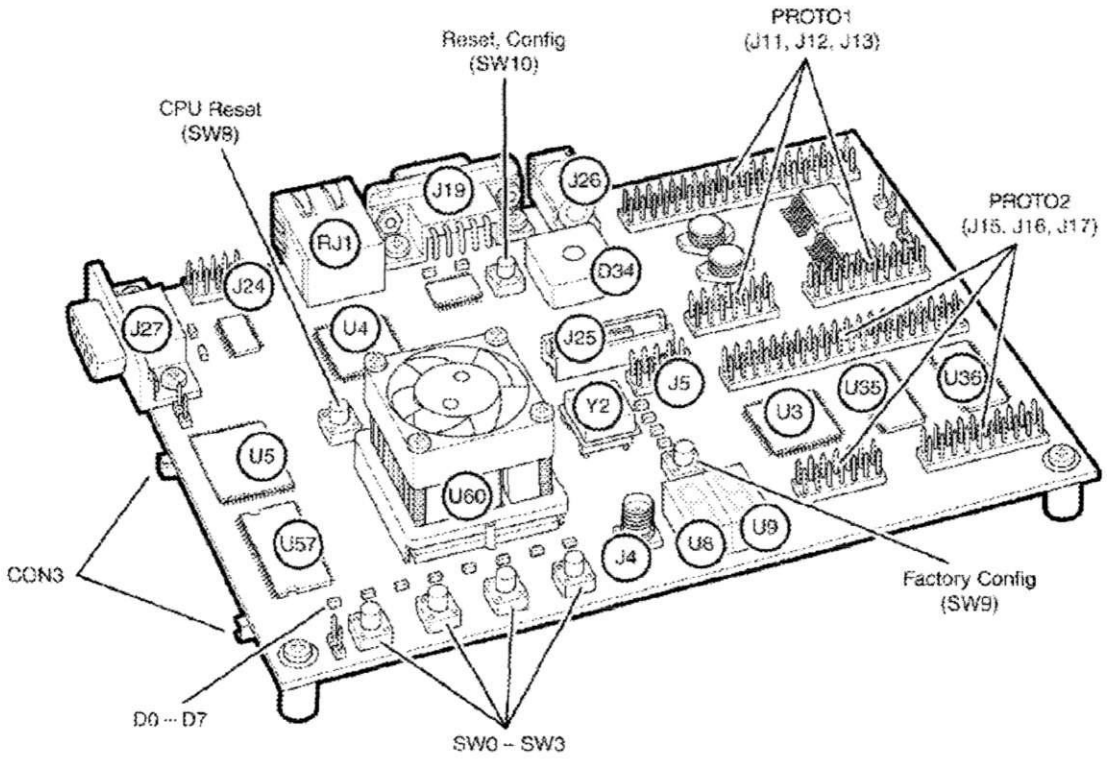

Figura 8.2: Placa de desenvolvimento Stratix-II da Altera

pode ser vista na Figura 8.6.

O leiaute assim como outras descrições do sistema, foi exaustivamente simulado e corrigido e finalmente enviado para fabricação na AMS, uma renomada *foundry* austríaca. Embora já tenha sido fabricado, até o momento da conclusão deste trabalho o circuito integrado ainda não chegou Brasil, o que deve ocorrer em breve.

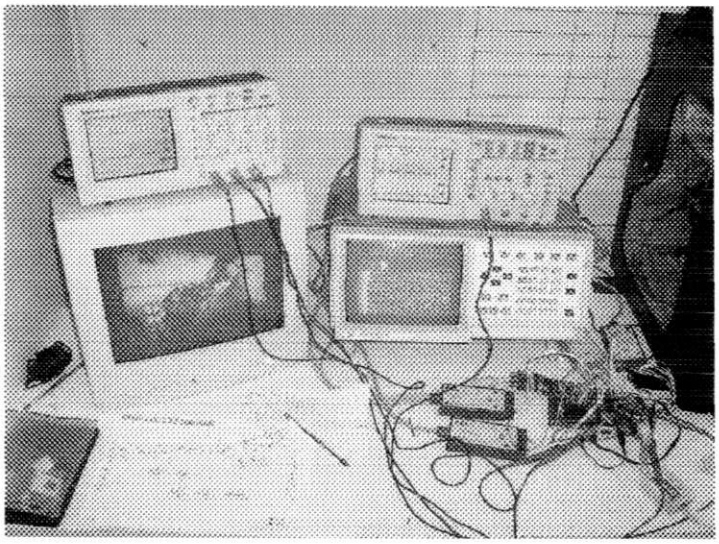

Figura 8.3: Decodificador MPEG-4 prototipado em depuração

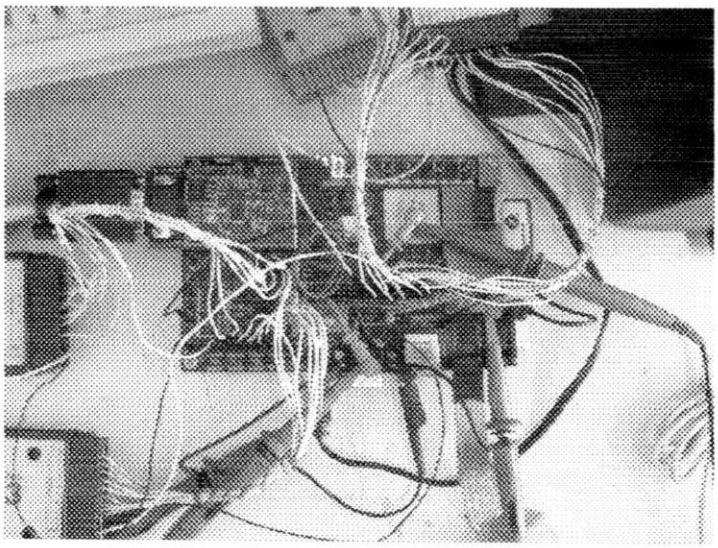

Figura 8.4: Placa Stratix-II da Altera com o Decodificador MPEG-4

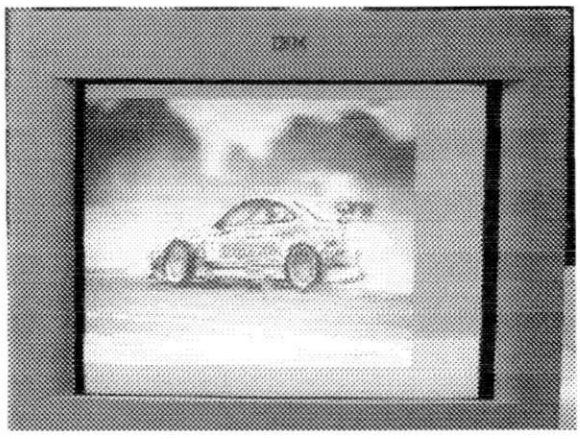

Figura 8.5: Video MPEG-4 decodificado pelo protótipo

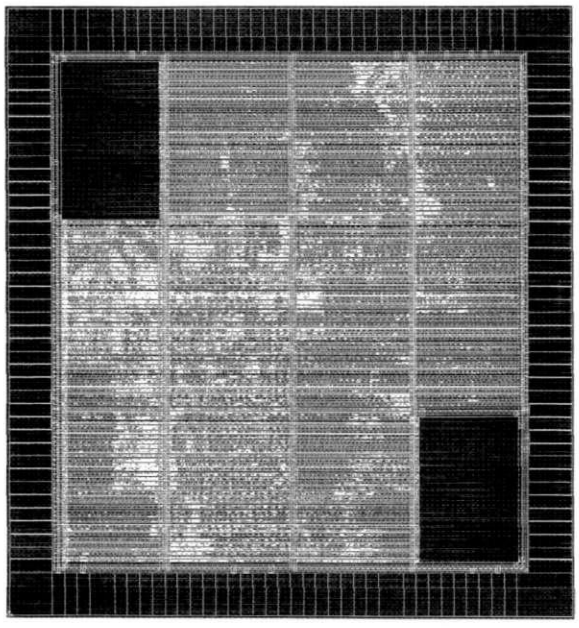

Figura 8.6: Leiaute do decodificador MPEG-4 para prototipação em silício

Capítulo 9

Conclusão

O trabalho desenvolvido ao longo deste projeto foi de profundo valor acadêmico. O contato com avançadas técnicas de concepção de circuito integrados VLSI, utilizando metodologias e fluxos de de projeto atuais na industria, bem como a experiência adquirida no uso de ferramentas profissionais agregam ainda um elevado valor curricular.

A abordagem de implementação do bloco de pré-decodificação de vídeo MPEG-4 baseado em microprocessador mostrou-se bem satisfatória, tendo atingido perfeitamente seus objetivos e todas os requisitos listados na fase de especificação. De fato, o requisito de flexibilidade revelou-se a característica mais marcante desta abordagem, estando este microprocessador, apos algumas alterações, atualmente em uso pela UNICAMP no projeto de um decodificador MP3, aplicação jamais imaginada na sua fase de projeto.

De forma geral, se pode considerar que os resultados obtidos foram muitos bons, além de terem sido de extrema relevância para meu aperfeiçoamento acadêmico, este trabalho fez parte de um projeto bem maior, o decodificador de vídeo MPEG-4, cuja complexa arquitetura é conhecida pela dificuldade de implementação em *hardware*, e foi um grande "motor" no reconhecimento do trabalho realizado por nossa instituição, não só nacional como internacionalmente.

Referências Bibliográficas

[1] Tai-Lun Chang, Ying-Ming Tsai, Chih-Da Chien, Chien-Chang Lin and Jiun-In Guo, *"A High-Performance MPEG4 Bitstream Processing Core"*, IEEE Int. Conf. on Multimedia and Expo, pp. 467-470, 2004.

[2] Yung-Chi Chang, Rho-Chieh Chang, Liang-Gee Chen, *"Design and Implementation of a Bitstream Parsing Coprocessor for MPEG-4 Video System-On-Chip Solution"*, Proc. VLSITSA 2001, pp. 188-191, April 2001.

[3] A.Mukhejee, N. Ranganathan, and M. Bassiouni, *"Efficient VLSI Designs for Data Transformations of Tree-Based Codes"*, IEEE Transactions on Circuits and Systems, pp. 306-314, 1991.

[4] T-H Tsai, W-C Chen and C-N Liu, *"A Low Power VLSI Implementation for Variable Length Decoder in MPEG-1 Layer III"*, IEEE Int. Conference on Multimedia and Expo, pp. 133-136, 2003.

[5] ,Stephen Molloy and Rajeev Jain, *"Low Power VLSI Architectures for Variable-Length Encoding and Decoding"*, IEEE Int. Conference on Multimedia and Expo, pp, pp. 997-1000, 1997.

[6] Ming-Ting Sun, *"VLSI Architecture and Implementation of a High-Speed Entropy Decoder"*, Proc. ISCAS-91, pp. 200-203, 1991.

[7] Dwu-Shian Ma, Jar-Ferr Yang and Jau-Yien Lee, *"A Programmable and Parallel Variable-Length Decoder for Video Systems"*, IEEE Trans. on CE, vol. 39, no. 3, pp. 448-454, 1993.

[8] K .R. G. da Silva, U. K. Melcher, G. Arajo, *"Automatic Testbench Generation Tool for a SystemC Functional Verification Methodology"*, Proceedings of the SBCCI2004, 17th Symposium on Integrated Circuits and Systems Design, p.66-70, 2004.

[9] K. R. G. da Silva, E. U. K. Melcher, I. Maia, and H. do N. Cunha, *"A methodology aimed at better integration of functional verification and rtl design"*, Design Automation

for Embedded Systems, vol. accepted for publication in a Special SystemC Edition, 2006

[10] ISO/IEC JTC1/SC29/WG11, *"N2502a, Generic Coding of Audio-Visual Objects: Visual 14496-2"*, Final Draft of International Standard, Dec. 1998.

[11] Altera Corporation, *"Nios Development Board Reference Manual, Stratix II Edition"*, July 2005.
2005, Altera Corporation

[12] Collins, Gerald W., *"Fundamentals of Digital Television Transmission"*, John Wiley & Sons, Inc., 2001.