Sistema de prototipagem rápida dirigido ao processamento de imagens

Marcos Ricardo Alcântara Morais

Dissertação de Mestrado submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Elmar Kurt Melcher, Dr.Ing.
Orientador

Campina Grande, Paraíba, Brasil
©Marcos Ricardo Alcântara Morais, outubro de 1998

Sistema de prototipagem rápida dirigido ao processamento de imagens

Marcos Ricardo Alcântara Morais

Dissertação de Mestrado apresentada em outubro de 1998

Elmar Kurt Melcher, Dr.Ing.
Orientador

Antônio Marcus Nogueira Lima, Ph.D.
Componente da Banca
João Marques de Carvalho, Ph.D.
Componente da Banca

Campina Grande, Paraíba, Brasil, outubro de 1998

M827s Morais, Marcos Ricardo Alcântara.
Sistema de prototipagem rápida dirigido ao processamento de imagens / Marcos Ricardo Alcântara de Morais. - Campina Grande, 1998.
88 f.

Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1998.
"Orientação: Prof. Dr. Elmar Uwe Kurt Melcher".
Referências.

1. Processamento de Imagens - Engenharia Elétrica. 2. Prototipagem - Sistema. 3. Interface - Barramento ISA. 4. Dissertação - Engenharia Elétrica. I. Melcher, Elmar Uwe Kurt. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 621:004.932(043)

SISTEMA DE PROTOTIPAGEM RÁPIDA DIRIGIDO AO PROCESSAMENTO DE IMAGENS

MARCOS RICARDO DE ALCÂNTARA MORAIS

Dissertação Aprovada em 21.10.1998

PROF. ELMAR UWE KURT MELCHER, Dr., UFPB
Orientador

PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFPB
Componente da Banca

PROF. JOÃO MARQUES DE CARVALHO, Ph.D., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Outubro - 1998

Dedicatória

Dedico este trabalho à minha esposa Flávia, com amor.

Agradecimentos

Agradeço à todos que me ajudaram no decorrer deste trabalho, em especial à minha família, pela compreensão e ajuda inestimável. Ao professor Elmar, pelo apoio, orientação e companheirismo em todas as horas. Em especial à minha esposa Flávia, pela compreensão nos momento difíceis. Obrigado a todos.

Resumo

Neste trabalho é apresentada a implementação de um sistema de prototipagem de hardware adequado ao processamento de imagens em tempo real. Através da análise da implementação de alguns algoritmos é mostrada a viabilidade do uso de hardware programável como principal elemento processador, obtendo-se flexibilidade similar a DSPs e poder computacional comparável a ASICs. O sistema desenvolvido utiliza dois FPGAs Altera e suporta a implementação de circuitos com até 140 mil portas equivalentes. O circuito possui interface com barramento ISA, o que permite a realização de co-projetos hardware/software. Resultados de mapeamento mostram que circuitos de complexidade relativamente grande podem ser devidamente inseridos no sistema.

Abstract

In this work an implementation of a prototype system capable of real-time image processing is shown. Through the analysis of the implementation of some algorithms the suitability of programmable hardware as main processor element is demonstrated, with flexibility similar to DSPs and computational power comparable to ASICs. The system developed uses two Altera FPGAs and can implement circuits consisting of up to 140 thousands equivalent gates. The systems architecture allows the aplication of data-flow algorithms, due to the high I/O throughput and fast local memory. The circuit has an ISA bus interface that allows the realization of hardware/software codesigns. Simulation results show that high complexity circuits can be mapped into the system.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O processamento digital de imagens é uma área de grande importância e que tem despertado acentuado interesse acadêmico, na indústria e na eletrônica de consumo. Através dele, pode-se transformar uma dada imagem de entrada, de modo a obter outra de saída mais adequada para uma finalidade específica [GW77]. Dessa forma, os diversos métodos de processamento são utilizados em uma ampla variedade de problemas que, apesar de nem sempre serem relacionados, necessitam de ferramentas capazes de melhorar a informação pictorial para a interpretação humana ou preparar imagens para a medição das características e estruturas presentes.

Encontram-se aplicações de processamento de imagens na medicina, no melhoramento de imagens de raios X, tomografia, ressonância magnética, etc.; também na astronomia, na microscopia, metalografia, fotografia, editoração eletrônica, televisão, entre outros.

Exemplos típicos de reconhecimento automático por máquinas que utilizam técnicas de processamento de imagens são reconhecimento automático de caracteres, visão de máquina na indústria para montagem e inspeção de produtos, medida de dimensões, controle de qualidade, reconhecimento militar, processamento automático de impressões digitais e processamento digital de imagens aéreas e de satélite para predição do tempo.

Atualmente, muita atenção tem sido direcionada à compressão e codificação de vídeo (sequência de imagens), devido a importância prática e econômica associada a tais assuntos. Para realizar este tipo de operação, a taxa de compressão e a qualidade de reconstrução são duas questões de grande importância. Atualmente as técnicas mais utilizadas para compressão de vídeo são extensões dos métodos usuais de compressão de imagens fixas, acrescentando-se técnicas de redução da redundância temporal (entre quadros). Isto normalmente é realizado através da estimação de movimento dos objetos presentes na cena [GFGT95, eMS96]. Baseado nesta estimação, decide-se se o próximo

quadro deverá ser comprimido independentemente ou usando-se referências ao quadro codificado anteriormente.

As técnicas mais modernas de codificação de imagens são baseadas em modelos. É uma forma poderosa de compactação, levando a codificação a baixíssima taxa de bits, como acontece nas normas MPEG-4. Uma das técnicas utilizadas é o modelamento poligonal, onde o codificador procura em cada cena por alvos que são identificados de acordo com alguns modelos padrão de objetos [Cor95a, Cor95c]. Altas taxas de compressão são obtidas uma vez que um objeto é identificado, isso porque ele pode ser enviado através de um número de quadros em uma sequência de imagens e apenas as mudanças subsequentes nos parâmetros do modelo (forma, movimento, etc.) precisam ser transmitidos. Para as aplicações de codificação de sequências de imagens ou vídeo é necessária que o processamento ocorra em tempo real.

Para aplicações em processamento de imagens em tempo real, é fundamental que a velocidade de processamento de toda a imagem seja igual ou inferior ao tempo de aquisição desta imagem. Certas aplicações como compressão de vídeo podem exigir um processamento de 30 quadros por segundo, onde cada quadro pode ser uma imagem de 512×512 pixels, com 256 níveis de cinza (8 bits). O tempo de processamento para operação em tempo real usando estes valores é de 120 nanossegundos para cada pixel. Este valor de performance está claramente fora do alcance para qualquer processador genérico ou DSP quando se considera a quantidade de operações normalmente envolvida no cálculo de cada um dos pixels da imagem. Observa-se que mesmo os DSPs atuais, como o C6201 da Texas, com arquiteturas específicas para processamento de imagens, só conseguem realizar em tempo real determinados algoritmos específicos para os quais foram projetados [Pet95]. Em vídeo, os elementos de imagem são adquiridos serialmente, um após o outro, uma linha após a outra. O mais natural então seria realizar o processamento da imagem ao mesmo tempo que ele é adquirida e, para tanto, a melhor escolha para implementação em hardware é em fluxo de dados, ou seja, ou seja, com a utilização de operadores encadeados que definem um caminho de processamento para os dados. Este tipo de implementação também é adequado aos algoritmos convolucionais, que possuem excelente características de performance.

A construção de um protótipo, ou seja, a realização material de um sistema funcional utilizando materiais, técnicas e métodos similares porém mais simples do que seriam utilizados durante a sua produção em quantidade, de forma a permitir a validação dos princípios e conceitos aplicados ao projeto do sistema, é útil durante as fases de desenvolvimento de uma implementação em hardware, pois permite também avaliar de forma mais próxima da realidade se o projeto de um sistema de hardware atende

suas especificações.

A prototipagem é dita rápica quando o tempo levado para a construção do protótipo não onera muito o tempo total de desenvolvimento, permitindo curtos ciclos de projeto. Atualmente, tem-se notado um crescente interesse na prototipagem rápida de sistemas digitais complexos, associado aos métodos usuais de simulação para verificação funcional. Assim, a prototipagem como método de verificação funcional pode ser utilizada numa ampla gama de projetos digitais, muito embora determinadas classes de projetos se beneficiem ainda mais com a prototipagem rápida, realizada através da modificação de interconexões, parâmetros ou reconfiguração de elementos funcionais já existentes.

Na metodologia de co-projeto em hardware-software, os sistemas digitais mais complexos que envolvem tanto processamentos de baixo nível quanto processamentos mais sofisticados, vários tipos de elementos processadores são utilizados. Realiza-se então uma divisão das tarefas e seu mapeamento nos vários elementos processadores. As tarefas mais elaboradas são melhor efetuadas por processadores de propósito geral, enquanto que as tarefas simples podem ser realizada a altas taxas em hardware dedicado.

Nessa metodologia, para que haja um fluxo de projeto paralelo de hardware e software, faz-se necessário que a plataforma de hardware esteja disponível antes das etapas finais do projeto, possibilitando exercitar o software associado. Além disso, o tempo de trabalho com o hardware final do sistema pode levar a detalhes significativos da operação do circuito além do que seria possível em simulações. Um exemplo típico de um projeto que poderia ser desenvolvido com a metodologia de co-projeto em hardware-software seria uma placa de vídeo para microcomputadores.

Recentemente emergiram novos paradigmas, como o hardware reconfigurável, que suportam a construção de hardware com grande rapidez [Pag96]. Um elemento chave tem sido o circuito reconfigurável na forma de FPGA (Field Programable Gate Array, matriz de portas programável). FPGAs podem ser programados facilmente para implementar sistemas de hardware de tamanho e velocidade razoáveis. A grande vantagem de utilizar ferramentas de compilação de hardware conjuntamente com FPGAs é que é possível construir sistemas de software-hardware em poucas horas. A limitação do uso de FPGAs é que eles não atingem a velocidade ou a densidade dos circuitos fundidos especificamente em silício.

Muitas das tarefas do processamento digital de imagens se adequam a uma metodologia de co-projeto em hardware-software, haja visto que muitos dos algoritmos utilizados no processamento digital de imagens utilizam tanto processamento de baixo nível quanto de alto nível. Os processamentos de alto nível são aqueles que atuam sobre a imagem utilizando alguma descrição desta e de forma global, geralmente através

de algoritmos sofisticados. Para um processamento de alto nível, a complexidade algorítmica, o pequeno volume de dados, a necessidade de modificação e adequação levam a uma implementação destes algoritmos em software sobre processadores de propósito geral ou DSPs. Por processamento de baixo nível entendemos aquele que atua sobre os pixels isolados ou sobre uma pequena região da imagem, realizando geralmente operações simples e repetitivas. Desta forma, o processamento de baixo nível atua na grande massa de pixels que forma a imagem, utilizando pequenas quantidades de dados por vez. Este tipo de processamento é adequado a uma implementação material em hardware, onde se pode explorar a simplicidade algorítmica, a velocidade de processamento e o paralelismo.

A implementação em hardware dos algoritmos de baixo nível de processamento de vídeo, como a maioria dos sistemas realizados segundo co-projeto hardare-software, seria beneficiada por uma etapa de prototipagem rápida. Além disso, devido à necessidade de flexibilidade, a muitas vezes um pequeno volume de produção e à especificidade dos algoritmos, muitos sistemas podem ser implementados de forma definitiva em elementos programáveis Dentro das considerações expostas anteriormente, pretendemos com este trabalho desenvolver um sistema de prototipagem rápida que seja capaz de realizar algoritmos de processamento de imagem de baixo nível em tempo real. Este sistema será utilizado para realizar projetos na metodologia de co-projeto hardware-software, onde os algoritmos de alto nível são executados pelo processador do ambiente hospedeiro. Durante o desenvolvimento e teste do sistema, foram utilizadas ferramentas CAD para projetos eletrônicos e o resultado final é apresentado na forma de uma placa de circuito impresso para microcomputadores tipo PC. Posteriormente, serão acrescentados ao sistema circuitos de aquisição e geração de vídeo.

O restante do texto se encontra organizado da seguinte forma:

O capítulo 2 resume alguns algoritmos de processamento de imagem, dando ênfase ao poder de cálculo exigido para cada um, fazendo uma análise comparativa da implementação destes algoritmos em FPGA, DSP e ASIC.

No capítulo 3 são traçadas considerações a respeito da prototipagem de sistemas digitais, enumerando alguns sistemas já existentes.

Apresentamos um modelo de arquitetura para o sistema de prototipagem rápida no capítulo 4. Aqui também todas as partes do projeto da placa serão detalhados.

O capítulo 5 trata da metodologia de projeto quando da utilização do sistema desenvolvido.

Os resultados obtidos durante o teste do sistema são mostrados no capítulo 6.

Finalizando o texto apresentamos as conclusões do trabalho.

Capítulo 2

Aspectos computacionais do processamento de imagens

Este capítulo relaciona uma lista não exaustiva de algoritmos de processamento digital de imagens, de diversas categorias, mostrando sua utilidade, suas características e dando a descrição matemática, através da qual é possível calcularmos o número de operações realizadas por pixel, avaliando assim a capacidade de cálculo necessária para que cada uma dessas operações possa ser realizada em tempo real numa sequencia de imagens.

Também é feita uma comparação das implementações destes algoritmos em ASIC, DSP ou lógica programável, a fim de avaliar a viabilidade de implementação destes algoritmos nestas classes de elementos processadores.

Este capítulo está organizado em quatro principais tópicos: considerações sobre processamento de imagens, classificação das operações sobre imagens, implementação em DSP, ASIC e FPGA e uma comparação entre estas implementações.

2.1 Considerações sobre processamento de imagens

O mesmo conjunto de elementos pode ser utilizado em uma variedade de algoritmos de processamento de imagem. Um sistema genérico de processamento de imagem geralmente realiza aquisição, armazenamento, processamento, comunicação e apresentação. Destes, nos atemos apenas às etapas de processamento.

Para a etapa de aquisição da imagem de forma digital, é necessário um dispositivo sensor, como um dispositivo CCD ou uma câmara de vídeo e um digitalizador, que converte a saída destes para a forma digital. O resultado deste processo é uma representação matricial cujos índices de linha e coluna identificam um ponto na ima-

gem e o elemento da matriz o valor de nível de cinza (ou outro parâmetro), ou seja, a intensidade naquele ponto. Os elementos são chamados de pixels (*picture elements*) ou elementos de imagem. Obviamente, os dados dos pixels não precisam representar apenas informação quantizada da iluminação. Outros tipos de imagens são:

- imagens coloridas, onde cada pixel é representado por um vetor com os componentes;
- imagens de profundidade, onde cada pixel representa um valor de distância;
- imagens de raio X, ultra-som ou microscópio eletrônico, onde cada pixel depende de uma densidade de um objeto ou de outro fenômeno físico; e
- imagens de tomografia computadorizada, em que cada imagem 2D (bi-dimensional) representa uma "fatia" de informação de densidade pertencente a um arranjo 3D.

O primeiro passo após a aquisição da imagem é realizar um pré-processamento básico de forma a adaptar melhor a imagem às etapas subsequentes. No pré-processamento são utilizadas técnicas como melhoramento de contraste, filtragem e remoção de ruído.

As técnicas de processamento podem operar no domínio espacial ou no domínio da frequência. O processamento no domínio espacial utiliza o próprio plano (linha, coluna) da imagem e manipula diretamente os valores dos pixels. Se apenas o valor do pixel na posição em questão é usado como entrada do processamento, este é dito pontual. São exemplos as transformações de intensidade, o processamento de histogramas, a subtração de imagens e a média de várias imagens.

Outros métodos realizam um processamento local levando em conta uma pequena vizinhança de cada pixel. O processamento denominado filtragem espacial é realizado através da convolução da imagem com "máscaras", como explicado mais adiante na secção 2.1.2. Exemplos são filtros de alisamento, gradiente, filtro da média, mediana, entre outros.

Por outro lado, diversos outros processamentos podem ser realizados no domínio da frequência, operando com a transformada de Fourier da imagem.

Um processamento mais complexo pode ser realizado como uma cadeia de etapas, onde a saída de uma etapa é a entrada da próxima. Um exemplo deste conceito pode ser visto na Figura 2.1, descrito em [Vil97].

O objetivo deste sistema é realizar a extração de pontos de vértice utilizados na modelagem poligonal de formas em imagens [MCCea96]. O processo é dividido em três etapas principais:

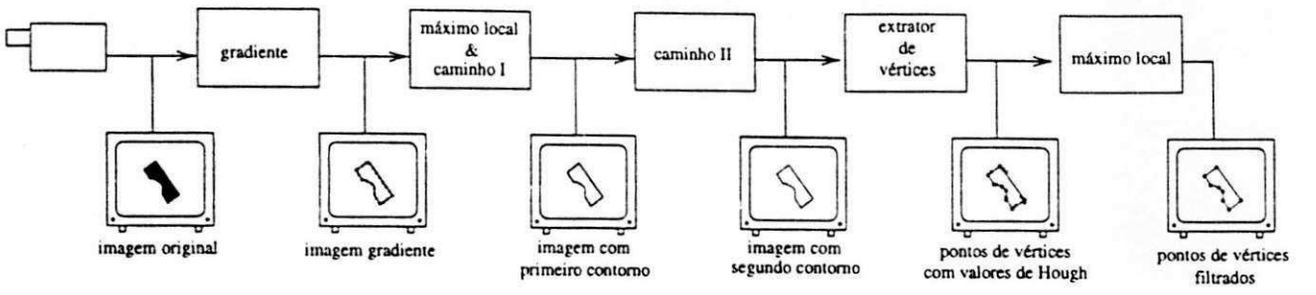


Figura 2.1: Algoritmos de fluxo de dados para extração dos contornos e vértices

- Aquisição e processamento das imagens da câmera;
- Extração dos contornos dos objetos presentes na cena;
- Extração dos vértices dos contornos processados.

A primeira etapa utilizada para obtenção dos contornos presentes na imagem de entrada é a aplicação do operador gradiente. Naquele trabalho foi utlizado o gradiente Spline [MCea96], por ser pouco sensível ao ruído e produzir bordas bastante definidas, com valores de pixels do contorno significativamente maiores que aqueles na sua vizinhança.

As bordas produzidas são afinadas, através das técnicas de máximo local e caminhos I e II [MNdC+97]. O máximo local seleciona os pixels de acordo com seu valor relativo aos outros pixels da vizinhança. A etapa caminho I determina se o pixel central da janela pertence a uma linha contínua, que passa por ele. A etapa de caminho II é similar a anterior e restringe a saída a um contorno contínuo com um pixel de largura.

Na extração de vértices a máscara é comparada a um conjunto formado por todos os vértices que se desejam identificar. Um resultado positivo indica a existência de um vértice no ponto central da janela.

Etapas de processamento semelhantes àquelas vistos anteriormente podem ser utilizados em diversos sistemas de processamento de imagens, sejam eles para compressão e codificação ou para reconhecimento e interpretação. Como resultado final destes processamentos obtemos uma imagem modificada, ou uma codificação mais adequada para armazenamento ou transmissão, ou ainda uma representação do seu conteúdo adequada à interpretação por máquina.

2.1.1 Imagens fixas e em sequência

Os diversos algoritmos da seção anterior podem ser aplicados a imagens fixas ou a uma sequencia de imagens, como em processamento de vídeo. Dados de imagens tipicamente

são produzidos e aplicados em forma de varredura, isto é, os pixels são apresentados serialmente, da esquerda para a direita para cada linha da imagem, começando com a linha superior. Desta forma o fluxo de pixels ocorre serialmente, um após o outro, uma linha após a outra. Se um quadro típico de imagem é de 512 linhas × 512 colunas de pixels de 8 bits, o total de dados em um único quadro é de 262,144 pixels, ou 2 megabits. Isto não apenas apresenta um desafio computacional mas também impõe problemas de armazenamento. Algumas tarefas relacionadas com imagem, como compressão, rastreamento e compensação de movimento, precisam de informação distribuída em uma única imagem (dados distribuídos espacialmente) e também dependem de dados presentes em quadros anteriores (dados distribuídos temporalmente). Devido a isto, o processador deve armazenar e recuperar muitos quadros rapidamente. A tabela apresenta os formatos de vídeo mais utilizados e os seus requisitos temporais, de acordo com a ordenação ITU-R BT.656 (MPEG, anteriormente CCIR-656) [Jac97a] para o fluxo de dados de vídeo digital que define as resoluções de amostragem e os espaços de cores utilizados (YCbCr e YCbCr 4:2:2). Para comparação é incluído na tabela o formato de HDTV e um formato com imagens quadradas de 512 × 512 pixels, a 30 quadros/s, utilizado em aplicações científicas [CNH93].

Sistema	Horizontal	Vertical	Frequência de campos (Hz)	Frequência de Pixels (MHz)	Tempo por pixel (ns)
NTSC	720	486	60	13.5	74
NTSC pixels quadrados	640	486	60	12.27	81
PAL	720	576	50	13.5	74
PAL pixels quadrados	768	576	50	14.75	68
HDTV	1280	1024	60	54	18.5
Científico	512	512	30	7.68	127

Tabela 2.1: Formatos de vídeo usuais.

Um processamento de imagem ocorre em tempo real se for realizado em período igual ou inferior ao tempo de aquisição de cada quadro. Neste caso, dizemos que a taxa média de dados processados (*throughput*) é suficiente, não havendo perda de quadros. Esta taxa de processamento é medida em quantidades de operações por segundo. Normalmente a latência, ou seja, o tempo entre o início da imagem de entrada e o início da imagem de saída, é significativo apenas em algumas aplicações,

como video-conferência, por exemplo.

2.1.2 Processamento no domínio espacial

As funções de processamento de imagens no domínio espacial podem ser expresssas como

$$g(x,y) = T[f(x,y)] \tag{2.1}$$

na qual $f(x,y)$ é a imagem de entrada, $g(x,y)$ é a imagem processada e T é um operador em f, definido sobre uma vizinhança de (x,y).

A maneira mais comum para definir uma vizinhança de (x,y) é definir uma subimagem, ou máscara, quadrada centrada em (x,y), como a Figura 2.2 mostra. O centro da subimagem é movido de pixel para pixel, iniciando, digamos, do canto superior esquerdo e aplicando o operador a cada nova posição (x,y). Embora outros formatos de vizinhança possam ser utilizados, as matrizes quadradas são mais utilizadas devido à facilidade de implementação.

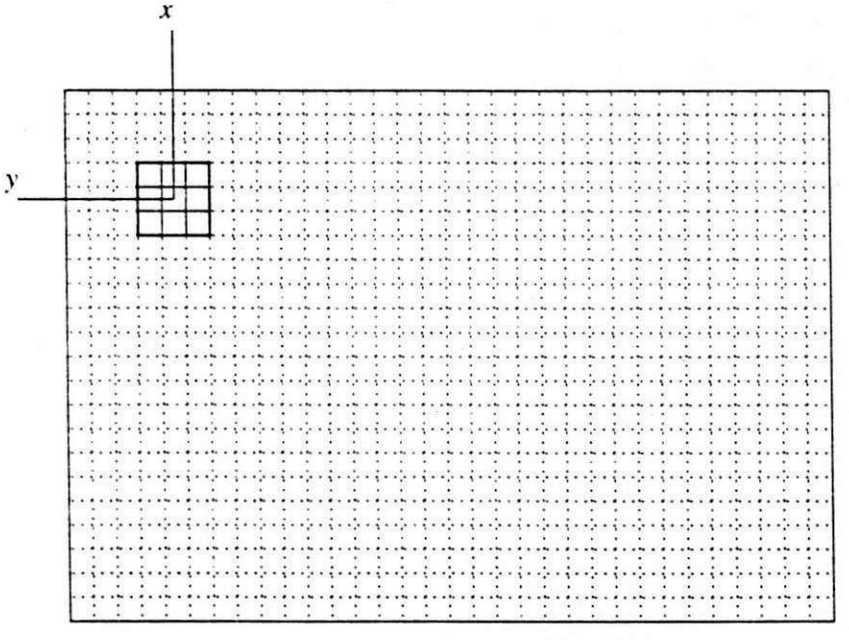


Figura 2.2: Janela centrada em (x,y)

A forma mais simples de T é quando a vizinhança é 1×1. Neste caso, T torna-se uma transformação pontual. Vizinhanças maiores permitem uma variedade de funções de processamento. Nesta formulação o processamentoé baseada no uso de máscaras. Basicamente, uma máscara é uma pequena matriz 2-D (por exemplo 3×3), em que

os valores dos seus elementos, os coeficientes de máscara, determinam a natureza do processo.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figura 2.3: Máscara 3×3 genérica

Na filtragem linear, a abordagem padrão é somar os produtos entre os coeficientes da máscara e as intensidades dos pixels sobre a máscara em posições específicas da imagem. A Figura 2.3 mostra uma máscara genérica 3×3. Esta máscara define uma região da imagem como a mostrada na Figura 2.2. Denominando-se os níveis de cinza dos pixels sob a máscara, ou seja, os valores dos pixels das posicões da região quadrada $(x-1, y-1)$ a $(x+1, y+1)$, $z_1, z_2, \ldots, z_9$, a resposta de uma máscara linear é

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \tag{2.2}$$

Na filtragem não linear os filtros espaciais também operam em vizinhanças realizando operações diretamente sobre os valores dos pixels. Um exemplo é o filtro de máximo, cuja resposta é $R = \max\{z_k \mid k = 1, 2, \ldots, 9\}$, usado para encontrar o ponto mais brilhante de uma imagem.

A velocidade e a simplicidade de implementação são características importantes do uso de máscaras espaciais em processamento de imagens. Os processamentos de imagem normalmente realizados no domínio da frequência, como filtragem passa-baixa ou passa-alta, podem ser realizados de forma aproximada com um processamento de máscara.

2.1.3 Processamento no Domínio da Frequência

Um processamento no domínio da frequencia é baseado na equação:

$$G(u, v) = H(u, v) F(u, v) \tag{2.3}$$

na qual $F(u,v)$ e $G(u,v)$ são as transformadas de Fourier das imagens de entrada e de saída, respectivamente, e $H(u,v)$ é a função de transferência do filtro. Pelo teorema da convolução, a equação 2.3 pode ser implementada no domínio espacial pela expressão

$$g(x,y) = \sum_{i=0}^{N-1} \sum_{k=0}^{N-1} h(x-i, y-k) f(i,k) \tag{2.4}$$

com $x, y = 0, 1, 2, ..., N-1$. Nesta expressão, h é a representação espacial do filtro, chamada de máscara convolucional. Se $h(x,y)$ for restrito a zero para valores de $x > n$ e $y > n$, com $n < N$, cria-se efetivamente uma máscara $n \times n$ que denominamos $\hat{h}(x,y)$, cujos coeficientes podem ser encontrados através de técnicas de minimização de erros. A convolução da imagem com $\hat{h}(x,y)$ pelo processo de máscara espaciais resulta em uma aproximação do processamento no domínio frequência. Este processo é muito utilizado na prática.

2.1.4 Processamento em fluxo de dados

Muito embora os pixels de uma vizinhança estejam próximos em uma imagem física, isto não significa que estejam próximos temporalmente na sequência de pixels produzida pela maioria das fontes de sinais de vídeo. A Figura 2.4 mostra o formato de uma sequencia de pixels produzida pela varredura de uma imagem. Para os propósitos de processamento da imagem, a abordagem mais simples seria armazenar a imagem inteira em memória local e utilizar cada pixel quando necessário para produzir a imagem de saída. No entanto, esta abordagem resulta em uma latência de pelo menos um quadro inteiro de imagem antes que o processador possa começar a gerar pixels de saída. Esta latência pode ser reduzida para menos do que n linhas (para uma vizinhança ou janela de $n \times n$) em uma arquitetura cuidadosamente projetada para entrelaçar leituras e escritas de memória, efetivamente utilizando a memória como uma linha de atraso.

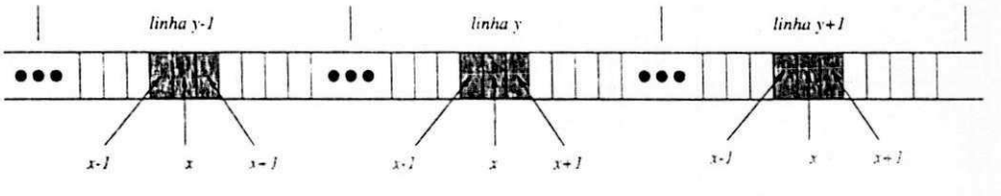


Figura 2.4: Sequência de pixels na varredura de uma imagem

Nestas arquiteturas especiais, denominadas de arquitetura em fluxo de dados, os vários operadores são estruturalmente dispostos e interligados da forma mais otimizada possível para a realização do algoritmo objetivado. As transferências de dados ocorrem

através de caminhos simples definidos entre origem e destino. Este tipo de arquitetura proporciona um aproveitamento máximo do paralelismo e do *pipelining*. Isto contrasta com uma implementação em processadores de uso geral, denominhada fluxo de instruções, em que a estrutura interna é predefinida (normalmente barramento) e o fluxo de dados é dirigido por programa. Neste tipo de implementação é possível que exista um gargalo de velocidade de transferência de dados devido a conflitos no barramento.

Arquitetura de fluxo de dados para processamento através de máscaras espaciais tem como grandes vantagens a pequena latência e a simplicidade algorítmica, associada à grande velocidade de processamento. Um núcleo processador de uma janela tem como entrada os pixels correspondentes às posições da janela, normalmente recebendo todos ao mesmo tempo de forma paralela, e gera um único pixel (valor) de saída. Os pixels de entrada são organizados em forma de janela por outro circuito gerador de janelas, que recebe o fluxo contínuo de pixels, armazena localmente as linhas em uma memória organizada como registrador de deslocamento e constrói a representação matricial da janela.

O resultado do processamento realizado desta maneira é uma saída de pixels na mesma taxa que são entrados. Isto permite que estes processamentos sejam facilmente encadeados, apenas conectando-os um após o outro. A latência e a necessidade de armazenamento de dados para realizar os cálculos dependem diretamente do tamanho da janela utilizada, como mostrado na tabela 2.2. Os valores apresentados são válidos para quadros com 512×512 pixels apresentados a uma taxa de 30 quadros/s.

Tamanho da Janela	Armazenamento (1 bit/pixel)	Armazenamento (8 bits / pixel)	Latência (μs)
3×3	1024	8192	61,7
5×5	2048	16384	123,2
7×7	3072	24576	184,8
8×8	3584	28672	215,6

Tabela 2.2: Desempenho dos algoritmos em fluxo de dados

Muitas operações diferentes de processamento de imagens podem ser implementadas usando uma arquitetura de fluxo de dados, como veremos adiante.

2.2 Classificação das operações sobre imagens

As operações sobre imagens foram classificadas em cinco classes genéricas [AA95]. Uma operação na **classe de combinação** utiliza duas imagens e produz uma nova imagem do mesmo tipo. Isto é realizado combinando-se cada par de elementos das imagens de entradas em um novo elemento. A **classe de transformação** aceita um imagem de um dado tipo e produz uma nova imagem do mesmo tipo. A **classe de medição** reduz uma imagem de um dado tipo a um escalar ou um vetor ou uma lista de escalares ou vetores. A **classe de conversão** refere-se as operações que tomam uma imagem de um dado tipo e a converte para um novo tipo. A **classe de geração** produz uma imagem a partir de uma descrição abstrata (síntese de imagem). A seguir veremos exemplos das classes de transformação, medição e conversão.

2.2.1 Operações de Transformação de Imagens

Como exemplo de operação da classe de transformação analisaremos o operador gradiente, que calcula o módulo da derivada espacial bidimensional da imagem e é amplamente utilizado como pré-processamento para a detecção de bordas. O efeito derivativo do gradiente realça as descontinuidades e atenua as regiões contínuas. Uma operação de limiar pode então isolar as regiões de borda.

O operador gradiente de Roberts é muito utilizado devido à sua simplicidade, pois utiliza máscaras 2×2. Para obter mais informação sobre a região em torno do pixel cujo gradiente deve ser calculado, vários outros operadores com máscaras 3×3 são utilizados. Dentre eles o operador de Sobel e os gradientes direcionais de Prewitt se destacam.

As máscaras utilizadas pelo operador Sobel são mostradas na figura 2.5.

-1	-2	-1
0	0	0
1	2	1

M_x

-1	0	1
-2	0	2
-1	0	1

M_y

Figura 2.5: Operadores Sobel

De uma forma geral, tomando-se uma janela $n \times n$, J, o gradiente na direção x, G_x, é definido pelo produto interno

$$G_x = M_x J \tag{2.5}$$

e na direção y, G_y, é calculado por:

$$G_y = M_y J \tag{2.6}$$

O gradiente no ponto central da área considerada será, então:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.7}$$

Devido a complexidade de cálculo da extração da raiz quadrada, normalmente se utiliza uma aproximação, de implementação mais fácil em computadores, com os valores absolutos:

$$G = |G_x| + |G_y| \tag{2.8}$$

A operação gradiente, por ser uma derivada, apresenta a característica de amplificar o ruído presente na imagem. Isto ocorre porque o ruído tipo *speckle* gera muitas descontinuidades pontuais. Um operador gradiente pode ser menos suscetível a ação do ruído se utilizar uma região maior da imagem, como ocorre com o gradiente Spline [MCea96]. Neste operador utilizam-se janelas 5×5, mostradas na figura 2.6.

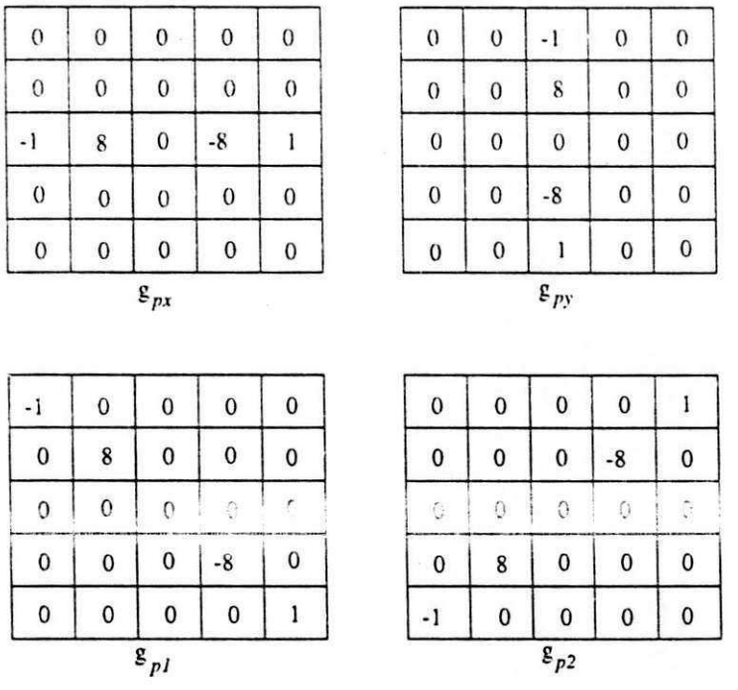


Figura 2.6: Máscaras utilizadas no cálculo do gradiente Spline

O gradiente bidimensional é definido por:

$$g_p = 3\left(|g_{px}| + |g_{py}|\right) + 2\left(|g_{p1}| + |g_{p2}|\right) \tag{2.9}$$

O operador é baseado na interpolação da imagem na vizinhança de interesse por splines cúbicas. Como resultado obtém-se um gradiente com bordas mais precisas, com os pixels pertencentes ao contorno com valores maiores que daqueles na vizinhança.

A menor sensibilidade deste operador gradiente ao ruído se deve ao fato de utilizar uma janela maior e determinar seu resultado a partir do valor médio das derivadas horizontal, vertical e diagonais.

2.2.2 Operações de Medição de Imagens

Diferente das outras classes de processamento, as operações de medição tipicamente não produzem uma nova imagem de saída. Em vez disso, o objetivo é extrair informações da imagem de entrada. O resultado obtido é um escalar, um vetor, ou uma lista de escalares ou vetores, com informações globais, tais como as frequências espaciais ou um histograma com o número de ocorrências dos valores particulares dos pixels.

Como exemplo desta classe analisaremos o processo de detecção de vértice, utilizado como pré-processamento em operações de modelagem poligonal de formas em imagens. O processo de dilatação de vértices pode ser melhor entendido após um breve descrição da modelagem poligonal.

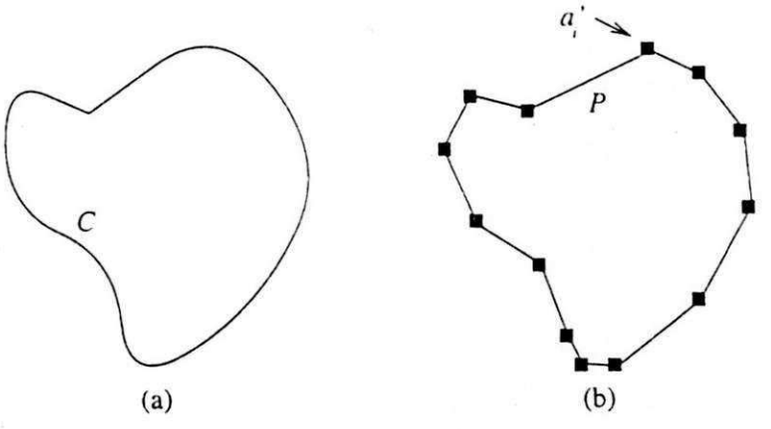


Figura 2.7: (a) Curva Fechada C; (b) modelagem poligonal da curva C.

Seja a curva fechada C, mostrada na Figura 2.7(a). Ela passa pelos pontos de borda segmentados de uma imagem. Estes pontos formam um conjunto com N elementos, $\hat{C} = a_1, a_2, \ldots, a_N$, no qual $a_1 = a_N$. O objetivo da modelagem poligonal é, então, aproximar a curva C por um polígono P, formado a partir dos vértices extraídos do

conjunto de pontos que aproxima a curva fechada, obedecendo a um determinado critério e limitado por um erro máximo de aproximação [Cor95a].

Portanto, considerando-se: um conjunto de vértices $V = a'_1, a'_2, \ldots, a'_{N'}$, no qual $V \subseteq \hat{C}$ e $N' \leq N$, S_k, um subconjunto de $\hat{C}$ cujos elementos pertencem ao intervalo $I_k = [a_i, a_j]$, com $a_i = a'_{k-1}, a_j = a'_k$ e $k = 1, \ldots, N'$ e o erro máximo ε, o critério de aproximação pode ser descrito pela função:

$$f(S_k) \leq \varepsilon \tag{2.10}$$

A definição de uma representação poligonal para a curva C implica que cada subconjunto é aproximado por um segmento de reta, como mostrado na Figura 2.7(b), cuja distância máxima aos pontos S_k é dada por $f(S_k)$ [Cor95b].

Os algoritmos de extração de vértice trabalham sobre uma imagem de contorno obtida através de pré-processamentos realizados com aplicação de uma operação de gradiente e afinamentos sucessivos da borda, de forma a obter contornos com largura de 1 pixel.

O processo de detecção de vértices foi implementado utilizando-se janelas e técnica de fluxo de dados por [Vil97], identificando os pontos de um contorno que formam vértice através da comparação com modelos pré-definidos de todos os possíveis vértices dentro de uma janela 8×8 . Estas máscaras, chamadas "máscara primitivas de vértices", percorrem toda a imagem, linha por linha, realizando comparações para verificar a ocorrência de alguma delas no quadro. Cada comparação positiva resultará em um vértice detectado, de forma que, ao final do processamento, é obtida uma lista dos vértices dos objetos presentes na cena. A composição da imagem poligonal é realizada numa etapa posterior, através da ordenação dos pontos obtidos.

2.2.3 Operações de conversão

Um exemplo de operação de conversão de imagens é a transformada discreta cossenoidal (DCT). A DCT é uma operação muito útil para aplicações de processamento de sinais, podendo ser definida globalmente ou sobre pequenos blocos da imagem. A utilização da DCT de toda a imagem é frequentemente evitada devido a grande exigência computacional. Neste caso, a DCT difere das operações descritas acima porque cada pixel transformado depende de todos os pixels da imagem.

A DCT é a base de muitos algoritmos padrão de compressão de imagens, notadamente JPEG e MPEG. Devido a isto, analisaremos a DCT em maiores detalhes. A DCT bi-dimensional é:

$$F(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \left[\frac{(2y+1)v\pi}{2N} \right] \quad (2.11)$$

para $u, v = 0, 1, \ldots, N-1$, e α é

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{para } u = 0 \\ \sqrt{\frac{2}{N}} & \text{para } u = 1, 2, \ldots, N-1 \end{cases}$$

A DCT bi-dimensional é uma operação separável, podendo ser realizada em dois passos de transformação unidimensional. Realiza-se a operação sobre as linhas e a operação é repetida sobre resultado, desta vez varrendo as colunas. A DCT é a transformação mais utilizada em sistemas de compressão de imagens devido à sua capacidade de agrupar informações em poucos coeficientes, à pequena complexidade computacional e à minimização da aparência de blocos, que resulta quando os limites entre subimagens tornam-se visíveis. Para aplicações em compressão de imagens, a DCT é realizada sobre um pequeno bloco da imagem, normalmente 8×8 ou 16×16.

2.3 Implementação em FPGA, ASIC e DSP

Nesta seção analisaremos algumas implementações dos exemplos de operações de processamento de imagens vistos na seção anterior em DSP ou processador genérico, ASIC e FPGAs. A cada exemplo é realizada uma análise da complexidade do processamento e das otimizações de software dependente da arquitetura do processador, e da estrutura para implementação em hardware, para FPGA e ASIC.

2.3.1 Gradiente Spline

O gradiente Spline é usado como exemplo de operação da classe de transformação e que opera com uma máscara convolucional 5×5, com operações de multiplicação, deslocamento, adição e extração de valor absoluto.

Implementação em DSP

Para avaliarmos a complexidade da implementação de um algoritmo em DSP, é necessário calcularmos o número de operações aritméticas ou lógicas e de movimentação de dados. Nesta avaliação não consideramos o tempo de aquisição dos dados de entrada, supondo que seja realizado ou por hardware externo ou pelo DSP sem perda de performance, devido a portas especiais de comunicação, por interrupção ou DMA.

Também não analisamos as possíveis otimizações devido a arquitetura do processador. As diferenças arquiteturais dizem respeito ao número de operações que podem ser realizadas de forma paralela, que se traduz no número de operações realizadas por ciclo de máquina. As otimizações arquiteturais possíveis incluem o uso de multiplicador-acumulador, vários caminhos de dados, várias barramentos de memória e operações em pipeline. Destas, apenas supomos o uso de multiplicador-acumulador, realizando uma multiplicação seguida de soma em um único ciclo de máquina, considerando estas operações como uma só, denominada MA, devido a sua existência em virtualmente todos os DSPs modernos.

Normalmente, a aplicação de um processador genérico em processamento de imagens se dá com a aquisição e armazenamento de um quadro inteiro de imagem. As operações são então realizadas utilizando a memória como entrada de dados, podendo esta ser acessada de forma aleatória e a uma taxa independente do fluxo de dados de entrada.

O gradiente Spline é calculado através de 4 gradientes unidimensionais, com os coeficientes ± 1e ± 8 como mostrado na Figura 2.6. Para cada um destes gradientes são executados 4 operações de cálculo do próximo endereço a ser acessado, 4 operações de leitura dos pixels e 4 operações aritméticas de MA. Os resultados destas operações devem ser armazenados temporariamente em registradores. Sobre estes valores são realizados 4 operações aritméticas de valor absoluto, 2 somas e 2 MAs. O total é então de 56 operações, que somadas as operações de transferência intermediária, temos aproximadamente 60 operações necessárias por pixel de saída.

Se considerarmos um DSP moderno capaz de realizar 30 MIPS/60MFLOPS (milhões de operações por segundo, milhões de operações em ponto flutuante), como o TMS320C40 [Inc92], o gradiente de uma imagem de 512×512 ou 262.144 pixels pode ser realizado a uma taxa de

$$\frac{60 \times 10^6}{512 \times 512 \times 60} = 3.8 \tag{2.12}$$

quadros por segundo. Para a operação em tempo real em DSP, supondo um sinal desta mesma resolução adquirido a 30 quadros/s, seria necessário um poder de processamento de $512 \times 512 \times 60 \times 30 = 472$ mips. Um sinal de vídeo D1 (13.5 MHz) exigiria $13.5 \times 60 = 810$ mips.

Algoritmos	Relógio(ns)	Frequencia max(MHz)	Celulas logicas	Recursos(%)
Vértice	44	22.7	415	18

Tabela 2.3: Resultado da implementação do algoritmo Spline em FPGA.

Implementação em ASIC

O gradiente Spline foi implementado em um circuito integrado dedicado por Melcher et al. [MCea96], usando biblioteca de células padrão da ES2 ECPD10 e o framework de projeto OPUS/Cadence.

A arquitetura utiliza árvores de Walace para acelerar multiplicações e somas. O atraso obtido foi de 33ns, o que resulta em uma taxa de pixels máxima de $1/33 \times 10^{-9} =$ 30MHz, atingindo performance de tempo real para todos os padrões de vídeo vistos na tabela 2.1, excluindo HDTV. A performance necessária para HDTV poderia ser obtida através do uso de pipeline. A área de silício utilizada no layout foi de $2532 \times 1665 \mu m^2$.

Implementação em FPGA

Para implementar o algoritmo Spline em FPGA foi utilizada uma descrição em Verilog sintetizável com entradas de dados em paralelo e saída em fluxo de dados, com resolução de 5 bits por pixel. A ferramenta de síntese utilizada foi o Max+plusII da Altera, versão 8.3, com entrada diretamente em Verilog. O dispositivo alvo foi um 10K40, da linha FLEX10K, com capacidade de integrar 40 mil portas lógicas. O resultado obtido pode ser visto na tabela 2.3. Observa-se que esta implementação pode atingir tempo real para aplicações em vídeo padrão NTSC (D1).

2.3.2 Detecção de Vértice

A detecção de vértice é um exemplo de algoritmo de da classe de medição. O algoritmo utiliza basicamente operações de comparação. As implementações em ASIC e FPGA utilizam uma arquitetura otimizada para implementação em hardware e é quase que exclusivamente combinacional.

Implementação em processador genérico

O algoritmo de detecção de vértice para modelamento poligonal foi implementado em estações de trabalho SparcStation-2 por [Cortez95]. O tempo de processamento deste algoritmo é dependente da complexidade da imagem de entrada. Os resultados obtidos para algumas imagens foram comparados com a implementação em ASIC em [Yuri97].

O tempo de processamento está na ordem de centenas de milissegundos para imagens de 256 × 256 pixels. Este nível de performance torna a extração de vértice um gargalo para o processamento do modelamento poligonal.

Implementação em ASIC

A detecção de vértice foi recentemente implementado em ASIC na tecnologia de células padrão ECPD07 da ES2 com largura mínima do canal de 0.7um [Vil97]. Esta implementação utilizou a metodologia de fluxo de dados, superando a performance exigida para operação em tempo real, considerando-se entrada de video no formato científico. A arquitetura do circuito integrado é mostrada na Figura 2.8.

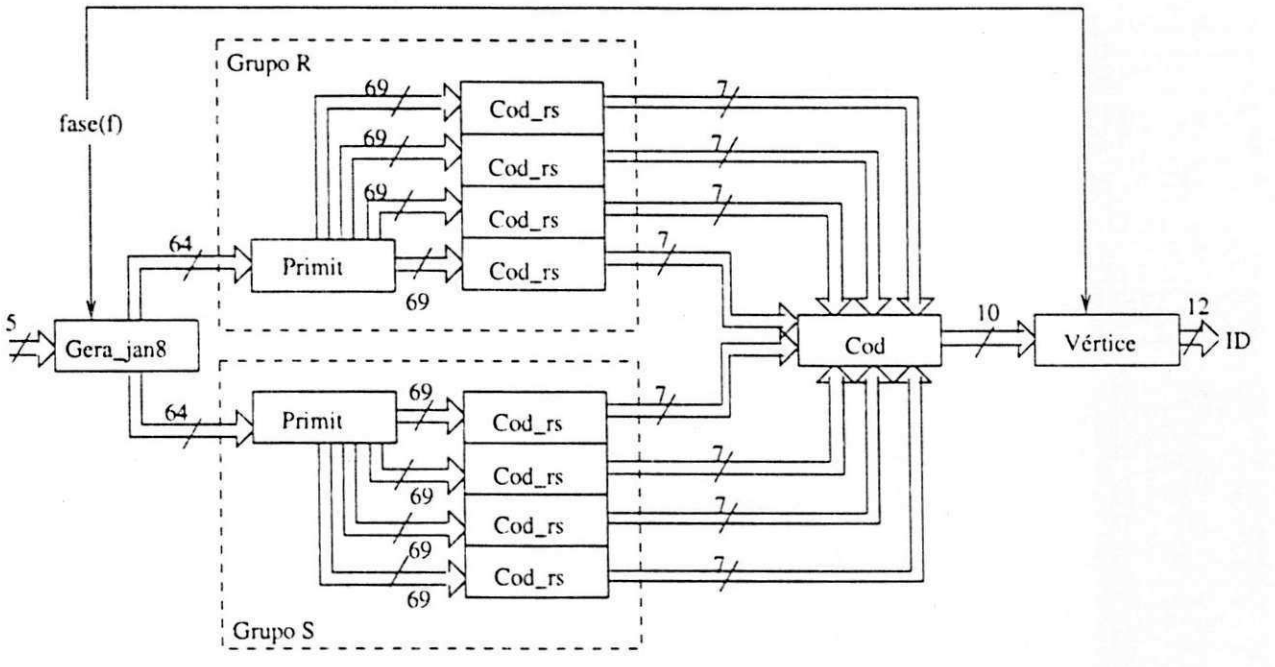


Figura 2.8: Arquitetura do CI para extração de vértices.

O princípio básico por traz desta implementação é o casamento de padrões. Foram geradas 273 máscaras primitivas, que quando rotacionadas e espelhadas, totalizam 2184 padrões binários para comparação. Nestas máscaras apenas os bits ativos são comparados. Desta forma, a comparação pode ser realizada através da operação lógica AND entre a máscara e a imagem. O resultado é um vetor de bits representando a cada máscara comparada. Este vetor pode ter mais de um bit ativo, significando que mais de uma máscara casou com a imagem. É realizado então uma codificação com prioridade. Estas comparações foram realizadas em 4 fases, onde a cada fase a imagem

Módulo	células padrão	transistores	atraso(ns)	área sem roteamento(mm^2)
gera_jan8	918	8568	10.57	0.62
primit	427	1696	2.44	0.16
cod_rs	323	1550	3.2	0.14
cod	74	470	4.71	0.04
vértice	82	1246	1.79	0.08

Tabela 2.4: Resultados da implementação do algorítmo de detecção de vértices em ASIC.

é rotacionada e duas janelas 8×8 são geradas para comparação. O relógio do circuito integrado é 5 vezes o relógio de pixel.

Na Figura 2.8, o bloco Gera_jan8 recebe uma coluna de 8 pixels, multiplexada no tempo no barramento de 5 vias de entrada, armazena esta coluna juntamente com as 7 últimas, formando internamente uma janela 8×8. Esta janela é apresentada em 2 saídas ao mesmo tempo, que são a representação da janela de forma espelhada e rotacionada. As 8 combinacoes possíveis de rotação e espelhamento são apresentadas em 4 fases distintas. Estas saídas são comparadas com primitivas armazenadas pelo bloco Primit. Este bloco ativa uma ou mais de suas 273 saídas quando detecta um padrão de bits ativos. Esse vetor passa por uma codificaç ao de prioridade realizada em duas etapas pelos blocos Cod_rs e Cod. O resultado é um valor de 10 bits, que juntamente com a fase (2 bits) indica de forma única o valor do vértice detectado. Este valor é usado como identificador da primitiva detectada. Este identificador, conjuntamente com a informaç ao da posição central da janela, determinam a existência de um vértice e das inclinações e direções das retas que passam pelo vértice.

Os resultados obtidos quanto ao número de transistores e de células padrão, os valores de área e de temporização para cada um dos módulos da Figura 2.8 são mostrados na tabela 2.4. A frequencia máxima de operação obtida foi cerca de 41MHz.

No total, o circuito utilizou 4512 células padrão, com 26076 transistores e um atraso de 22,71 ns. A área utilizada pelas células foi de $2.18mm^2$. A área total após a inserção dos PADs e roteamento foi de $12.09mm^2$.

Implementação em FPGA

A mesma descrição em Verilog utilizada por [Yuri97] foi sintetizada com a biblioteca de FPGA da Altera, utilizando o sintetizador Synergy do framework Cadence e aplicando o resultado da síntese na ferramenta Max+PlusII da Altera, versão 8.3, com mapeamento para o dispositivo 10K40, da linha FLEX10K, com capacidade de integrar 40 mil

portas lógicas. O resultado obtido para o dispositivo mais lento da linha 10K40 pode ser visto na tabela 2.5. Podemos observar nesta tabela que a performance de tempo real não foi obtida. No entanto, com pequena modificação da descrição original para a inclusão de registradores de pipeline, a performance de tempo real pode ser obtida, a custo de uma latência um pouco maior. A cada banco de registradores colocado é acrescentado um pulso de relógio na latência.

Algoritmos	Relógio(ns)	Frequência max(MHz)	Células lógicas	Recursos(%)
Vértice	227	4.4	2236	97

Tabela 2.5: Resultado da implementação do algoritmo de detecção de vértices em FPGA.

2.3.3 DCT

A seguir veremos a implementação da DCT bi-dimensional de blocos de tamanho 8×8, normalmente utilizada nos padrões de compressão de imagens. Para este processamento não se utiliza uma janela de pixels ao redor de cada pixel, mas é realizada em blocos separados. Desta forma, embora o número de operações por bloco seja alto, o número total de operações sobre a imagem é relativamente pequeno.

Implementação em DSP

Muitos algoritmos foram desenvolvidos para cálculo da DCT de forma eficiente em hardware e em software. Para o cálculo da quantidade de operações necessárias utilizamos o algoritmo adaptado de Chen e Smith [CSF77].

Para $N = 2^m$, $m > 2$, este algoritmo requer:

$$\left(\frac{3N}{2}\right)(\log_2 N - 1) + 2 \tag{2.13}$$

somas e

$$N \log_2 N - \left(\frac{3N}{2}\right) + 4 \tag{2.14}$$

multiplicações.

Para o caso de $N = 8 \rightarrow m = 3$, temos 26 somas e 16 multiplicações. A leitura e escrita dos valores contribui com 16 operações de transferência de dados. A DCT 2-D é calculada aplicando a DCT unidimensional sobre as linhas e aplicando novamente a

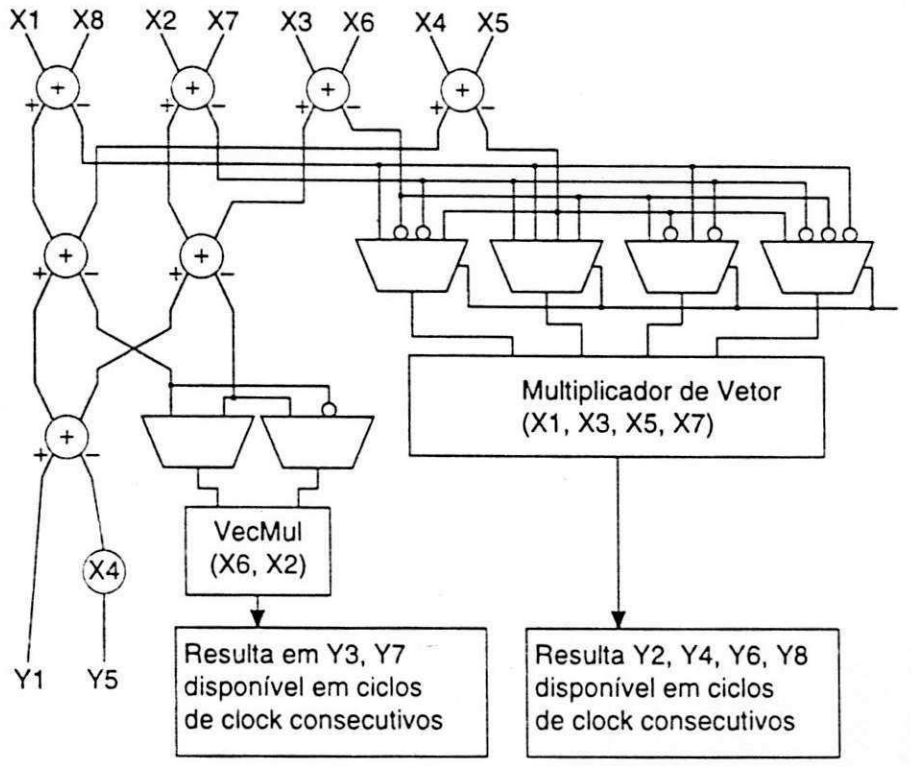


Figura 2.9: Implementação da DCT unidimensional de 8 pontos para FPGAs

DCT sobre o resultado obtido sobre as colunas. O total é 2N operações de DCT unidimensionais. Para um bloco de imagem isto resulta em 416 adições, 256 multiplicações e 256 transferências de dados, num total de 928 operações por janela 8×8.

Em uma imagem padrão de 512×512 pixels, existem 4096 blocos 8×8, e o algoritmo realiza 3.8 milhões de operações. Um DSP de 60 mips efetuaria este algoritmo a 15 quadros/s.

Implementação em ASIC

Devido a sua grande utilização em sistemas de compressão e descompressão de imagens, diversas implementações comerciais e e científicas foram realizadas. Já no início da década podemos encontrar implementações capazes de suportar as taxas de pixels iguais ou superiores às utilizadas em HDTV [CL].

Implementação em FPGA

Uma implementação da DCT bidimensional de blocos 8×8 otimizada para FPGAs Altera da linha FLEX10K pode ser visto em [Alt96]. O diagrama em blocos da DCT unidimensional de 8 pontos pode ser visto na figura 2.9.

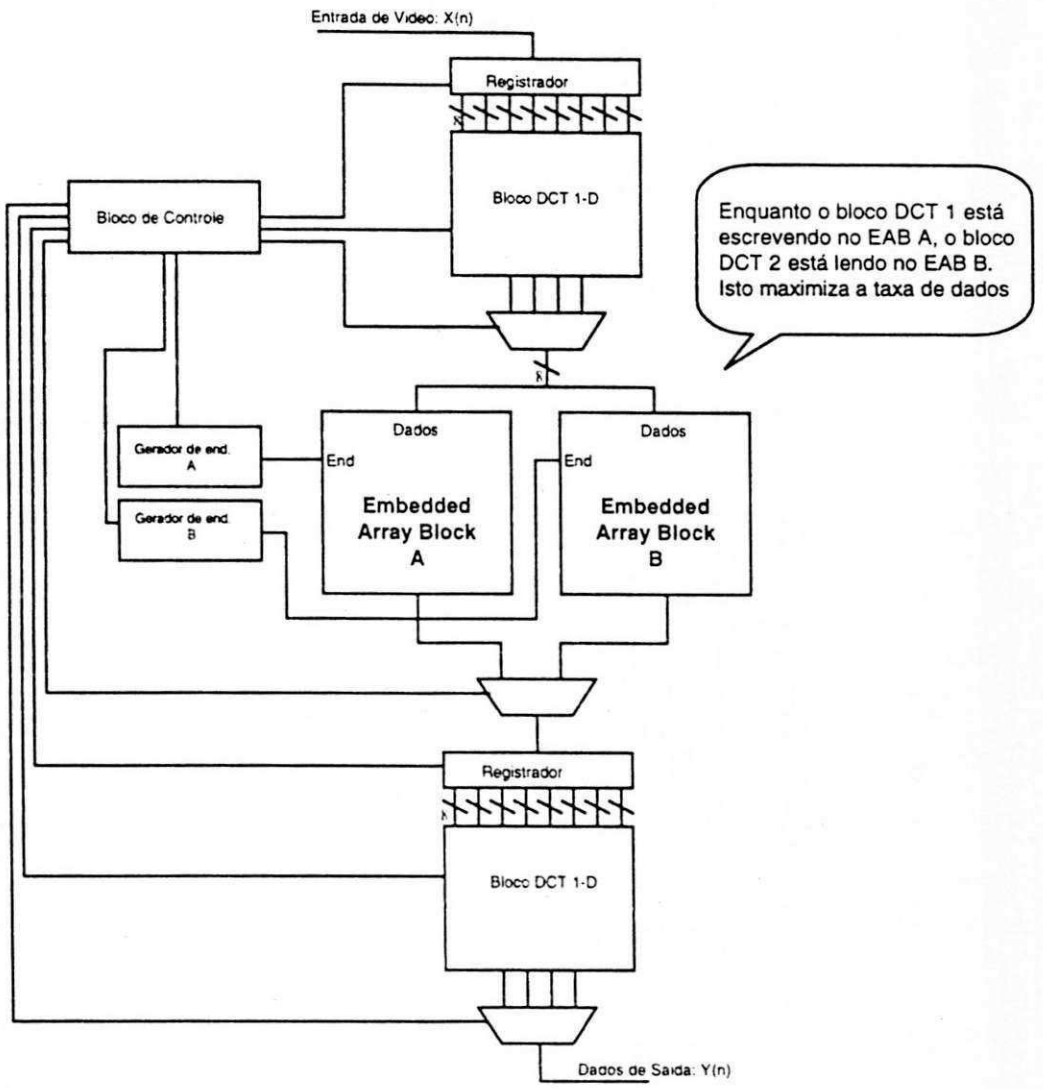


Figura 2.10: Implementacao da DCT de 8×8 pixels otimizada para FPGAs

Como a DCT é uma transformação separável, ela é implementada usando dois estágios idênticos de DCT 1-D, com valores intermediários armazenados em memória interna do FLEX 10K. O diagrama em bloco da DCT 2-D é mostrado na Figura 2.10.

A DCT 2-D inteira cabe em menos de 50% de um dispositivo Altera 10K50. Este circuito é capaz de processar vídeo de pixels a mais de 30 quadros por segundo no dispositivo mais lento desta linha.

2.4 Comparação entre as implementações

Na operação de extração de gradiente Spline, observamos que a performance de um DSP fica muito aquém do necessário para o processamento em tempo real, que só poderia ser realizado utilizando-se um conjunto de processadores interligados. Por

outro lado, as implementações em ASIC e FPGA analisadas ultrapassaram o nível mínimo de performance para o processamento em tempo real.

A extração de vértice leva centenas de milissegundos em processador genérico com pequenas imagens, mas pode ser realizado em tempo real em ASIC. O processamento em tempo real só pode ser obtido em FPGA através de uma descrição mais otimizada.

A operação de DCT pode ser realizada por DSP a taxas signifativas de quadros/s. No entanto, as implementações em ASIC e FPGA atingem performances capazes de lidar com sinais de vídeo de alta resolução, como em HDTV. A tabela 2.4 resume as principais características destes dispositivos processadores. Vemos nesta tabela que os dispositivos programáveis apresentam maior facilidade de implementação, normalmente porque o ciclo de projeto é mais curto e os resultados podem ser apreciados mais rapidamente. O DSP possui um ciclo de projeto mais curto do que o FPGA devido às ferramentas de software disponíveis.

Dispositivo	Performance	Implementação	Custos adicionais	Volume de Produção
DSP	Baixa	Fácil	pode necessitar de várias unidades	Pequeno e médio
ASIC	Muito alta	Difícil	desenvolvimento, NRE	Muito alto
FPGA	Alta	Médio	custo unitário alto	Pequeno e médio

Tabela 2.6: Comparação das possibilidades de implementação

Pelos exemplos anteriores, podemos observar que os processadores de sinais digitais (DSPs) e processadores de uso geral não são adequados a realização de algoritmos de processamento de imagens de baixo nível em tempo real. As vantagens da sua utilização é a facilidade de utilização, generalidade e a grande flexibilidade, uma vez que qualquer alteração no algoritmo é realizada através de uma modificação no software executado.

Por outro lado, os circuitos integrados de aplicação específica ultrapassam os níveis de performance exigidos para tempo real. Porém os altos custos envolvidos no projeto e na fabricação destes dispositivos só justificam sua escolha para grandes volumes de produção, onde são a solução mais eficiente em termo de custo.

Já os dispositivos programáveis como FPGA são uma solução mais viável quando temos pequenos a médios volumes de produção e também para prototipagem, porque permitem operações a velocidades relativamente altas, com custos menores de desenvolvimento do que um ASIC. As desvantagens de sua utilização está na dificuldade de

descrição dos algoritmos de forma a tirar vantagem de seu paralelismo e da utilização de linguagens de descrição de hardware e técnicas de projeto de hardware.

No próximo capítulo veremos com detalhes a utilização de dispositivos lógicos programáveis como FPGAs em sistemas de processamento de alta performance e em prototipagem, com ênfase no processamento de imagens em tempo real.

Capítulo 3

Prototipagem rápida e computação reconfigurável com FPGAs

Este capítulo inicia com uma discussão sobre a prototipagem de sistemas digitais complexos. A utilização de FPGAs como principal recurso para prototipagem rápida é então destacada. Em seguida, vemos a aplicação de FPGAs como elementos processadores em computação reconfigurável. Finalizando apresentamos alguns dos principais sistemas de prototipagem rápida e computação reconfigurável comerciais.

3.1 Prototipagem de sistemas digitais

Normalmente, um protótipo para um sistema digital complexo é construído para realizar a verificação funcional, avaliação de desempenho, validação de algoritmos, ou ainda, para permitir o desenvolvimento em paralelo dos demais subconjuntos que constituem o sistema, como software e hardware em um co-projeto hardware-software.

A verificação funcional é o processo na qual um modelo do sistema é exercitado com um conjunto de vetores e sua saída avaliada, levando a validação ou não do projeto. Esta verificação pode ser realizada por simulação em software ou com um protótipo. As ferramentas de simulação de linguagens de descrição de hardware (HDL) permitem que os projetos sejam completamente avaliados em software, sem a construção de um protótipo de hardware. No entanto, a complexidade e requisitos temporais dos sistemas atuais dificultam uma simulação a nível de sistema. Desta forma, a prototipagem de hardware torna-se um adjunto útil ao processo de desenvolvimento.

Da mesma forma, ao efetuarmos uma análise de desempenho através da construção de um protótipo é possível avaliarmos se os objetivos do projetos estão sendo atingidos, em tempo hábil para reavaliações desses objetivos.

A utilização de um protótipo para a validação de algoritmos pode ser justificada pela quantidade maciça de dados normalmente associada a implementações em hardware. Torna-se imprático simular um sistema completo descrito em uma HDL com um conjunto de vetores representativo dessa massa de dados, mesmo em estações de trabalho muito rápidas. Um exemplo deste conceito é um processamento complexo de imagens. Dias ou mesmo semanas são necessários para simular o processamento de uma única imagem de tamanho completo. E como algumas aplicações processam sequências de imagens, centenas de simulações podem ser necessárias para analisar adequadamente apenas alguns segundos de dados. A implementação destes algoritmos em um protótipo de hardware permite sua execução em tempo real, auxiliando de forma decisiva na validação de um novo algoritmo ou conceito.

Na construção de um protótipo de hardware, objetiva-se também permitir a cooperação no desenvolvimento das diversas etapas de um sistema e a utilização da metodologia de implementação *bottom-up*, realizando protótipos de módulos de nível mais baixo para a sua utilização em módulos de nível mais alto, de acordo com as relações de dependência. Em um co-projeto de hardware-software, o desenvolvimento do software é beneficiado com a existência de um protótipo do hardware. Nesse caso, o processo de prototipagem precisa incorporar métodos de alocar os custos e o tempo de desenvolvimento do software ao mesmo tempo que o hardware é desenvolvido. Ainda assim, a integração do protótipo é complicada devido a utilização de um hardware novo e um software novo, tudo sendo desenvolvido ao mesmo tempo.

3.2 Prototipagem rápida com FPGAs

Os dispositivos lógicos programáveis possuem características que os tornam adequados a serem utilizados como elementos constituintes de um protótipo de hardware digital. Inicialmente, foram utilizados as matrizes lógicas programável (PAL), baseadas na tecnologia de fusíveis. Estes foram seguidos pelos dispositivos lógicos programáveis apagáveis (EPLD), que utilizam células tipo EPROM para armazenar a configuração do circuito. Os dispositivos lógicos atuais são baseados em células de armazenamento tipo SRAM, FLASH e EEPROM. Devido a reconfigurabilidade e flexibilidade, os dispositivos baseados em SRAM se mostram os mais adequados a realização de protótipos e atualmente eles dominam o cenário de prototipagem rápida[].

A utilização de lógica programável para prototipar ASICs tem crescido a medida que a capacidade dos dispositivos programáveis tem aumentado. Com eles, é possível testar um projeto de ASIC diretamente no circuito final e verificar o seu funcionamento

com outras partes do sistema, algo que dificilmente pode ser realizado extensivamente com simulação. O teste diretamente no circuito evita reprojetos do ASIC, que são dispendiosos, tanto em gastos com NRE (*non-recurring engineering*, gastos em engenharia que não são reaproveitáveis) e tempo.

No entanto, para a construção de um protótipo utilizando FPGA é necessário que este esteja inserido em um sistema. Ou seja, além do dispositivo programável que constituirá todo ou parte do hardware desenvolvido, é necessário que existam dispositivos de entrada e saída, alimentação e um meio que permita a programação dos dispositivos.

Um dispositivo FPGA pode ser considerado um como um hardware reconfigurável apenas dentro de certos limites. A arquitetura interna destes dispositivos é micro e macroscopicamente reconfigurável, dando-lhes grande flexibilidade. No entanto, os elementos programáveis e roteamentos extra destas arquiteturas ocupam cerca de 90% da área de silício [Pet95] e também geram atrasos. Isto se traduz em limitações temporais e em número de portas utilizáveis. Desta forma, um circuito ao ser sintetizado para o protótipo normalmente apresentará níveis de performance inferiores ao seu par em ASIC. As limitações no número de portas dos FPGAs também podem fazer com que a síntese para o protótipo tenha de ser particionado e implementado em diversos dispositivos. Como as interconexões externas aos dispositivos apresentam atrasos bem maiores que as internas, o problema de performance se agrava. Todos estes fatores juntos fazem com que uma prototipagem em FPGA normalmente emule todas as funcionalidades do ASIC, sem no entanto atingir os mesmos níveis de performance. Assim, um protótipo de um ASIC implementado em FPGA normalmente deve operar com frequência de relógio e taxas de transferências inferiores às do circuito final.

Para superar limitações de área em protótipos e sistemas que utilizam FPGA, pode se utilizar o conceito de reconfigurabilidade em tempo de execução [Cor97][EH96], encontrado nos mais recentes dispositivos.

São utilizados dispositivos que podem ser reconfigurados por parte e muito rapidamente, ou ainda auto-reconfigurados. Desta forma, circuitos maiores do que caberiam em um único dispositivo podem ser implementados, desde que todas as partes não estejam operacionais ao mesmo tempo. Dependendo do algoritmo a ser implementado, resultados intermediários de um processamento anterior podem ser deixados em registradores para ser processado no novo hardware reconfigurado.

Apesar destas limitações, uma outra grande vantagem da realização do protótipo com FPGAs é a possibilidade de utilizar uma metodologia que se aproxima da utilizada para a realização do sistema final. No entanto, os efeitos secundários da utilização de um hardware diferente devem ser conhecidos e levados em conta. Por esses motivos, a

prototipagem de ASICs em lógica programável pode conter algumas dificuldades para projetistas iniciantes. Normalmente os projetos são criados em linguagens VHDL e/ou Verilog de forma independente da tecnologia. Após a simulação da funcionalidade do projeto, a síntese pode ser realizada com a biblioteca da lógica programável como alvo para utilização no protótipo. A mesma descrição deve ser sintetizável também para a biblioteca de ASIC para a produção.

No entanto, os algoritmos utilizados na síntese de um ASIC não forneciam resultados otimizados quando utilizados para mapear para um FPGA devido a diferenças arquiteturais. Isto levou a necessidade de algoritmos específicos para as arquiteturas deste dispositivos. Atualmente as ferramentas de síntese fornecem resultados aceitáveis quando mapeam sua saída para a biblioteca de FPGA. As técnicas de mapeamento e otimização têm se tornado mais sofisticadas, usando algoritmos adaptados para as arquiteturas dos principais vendedores de dispositivos lógicos programáveis, de forma que os projetos sejam sintetizados de forma otimizada para lógica programável ou para ASICs [Man95].

Como importante resultado desta evolução dos FPGAs, temos a utilização destes dispositivos como elementos processadores de algoritmos mais complexos, com vantagens significativas sobre elementos programáveis e circuitos dedicados, abrindo uma outra opção para processamento de alto desempenho, a computação reconfigurável, como visto em seguida.

3.3 Processamento de sinais com FPGAs

Como resultado dos benefícios potenciais e da vasta aplicação das técnicas de processamento digital de sinais (DSP), muito esforço tem sido empregado no desenvolvimento de sistemas de hardware especializados que permitam uma implementação eficiente do algoritmos de processamento digital. de sinais. Estes hardwares especializados buscam atingir as demandas computacionais e de E/S estritas normalmente associadas aos algoritmos de DSP. Estes sistemas de hardware especializados tomam a forma de processadores programáveis com unidades funcionais especializadas (processadores para DSP) e projetos dedicados com pouca ou nenhuma programabilidade (ASICs). Estas duas abordagens representam os extremos da curva de compromisso entre flexibilidade e performance. Cada qual se adequa melhor a determinados tipos de aplicação. Uma outra abordagem, o hardware reconfigurável, é capaz de combinar as características desejáveis de flexibilidade com altos níveis de performance.

O termo hardware reconfigurável refere-se ao uso de qualquer sistema eletrônico

capaz de mudar sua estrutura estaticamente (entre aplicações) ou dinamicamente (durante uma aplicação) sem o acréscimo de elementos físicos de hardware. São constituídos basicamente de dispositivos lógico programáveis como FPGAs e de dispositivos programáveis de interconexão, como FPIDs (Field Programmable Interconect Device).

Os FPGAs atuais de alta densidade e alta performance fornecem uma alternativa atrativa para implementações de processamento de sinais de alta velocidade e de outros tipos de algoritmos de alguma forma paralelizáveis. Estes dispositivos possuem grandes vantagens sobre a fabricação de ASICs, incluindo reduzido tempo de desenvolvimento, custo e risco com a mesma habilidade de adaptação do hardware as necessidades específicas da aplicação. Além disto, a utilização de FPGAs baseados em SRAM oferece a vantagem da reconfigurabilidade, a possibilidade de alterar o hardware do sistema para preencher as necessidades do momento através da reconfiguração dos dispositivos FPGAs. Isto garante a flexibilidade da aplicação ao permitir que o próprio projeto de hardware seja alterado em lugar de acrescentar hardware programável por software (registros de configuração) a um projeto de hardware. Ou seja, para mudar uma aplicação implementada com um sitema de hardware reconfigurável, é necessário apenas recarregar os bits SRAM de configuração. Isto permite que um projeto de hardware completamente novo seja implementado. Desta forma, uma mudança de parâmetros pode ser realizada reprogramando-se o FPGA em lugar de modificar registradores internos.

Os FPGAs geralmente consistem de uma matriz de células lógicas flexíveis relativamente simples, registradores e memórias, cujas funções e interconexões são controladas através do carregamento de bits de memória de configuração. Tipicamente, cada célula contém recursos lógicos para implementar uma operação com ao menos 2 ou 3 bits. Múltiplas células são usadas em paralelo para implementar operações de qualquer largura escolhida. A existência de memória SRAM permite que operações possam ser implementadas através de tabelas. A implementação de algoritmos de processamento de sinal em um FPGA é em essência conectar os vários operadores. Usando-se técnicas adequadas a arquitetura baseada em célula dos FPGA para implementar funções básicas de multiplicação-acumulação, tais como aritmética serial e paralela distribuída, sistemas baseados em FPGA podem superar processadores DSP convencionais em aplicações comuns, como visto no capítulo anterior e em [CME94]. A natureza da arquitetura FPGA rica em registros facilita técnicas de pipeline para aumentar a taxa global de processamento do sistema.

O uso de FPGA se estende desde a implementação de simples lógica de interface até grande sistemas de processamento reconfiguráveis. Aumentos na capacidade e perfor-

mance obtidos nas ultimas gerações de FPGAs têm expandido sua faixa de aplicações incluindo funções como ULAs, multiplicadores, filtros digitais e operadores complexos similares. Algoritmos inteiros de processamento de vídeo podem ser implementados em dispositivos FPGAs, tirando vantagem do paralelismo inerente de uma implementação de hardware para atingir altos níveis de performance.

Processadores reconfiguráveis baseados em FPGA combinam a versatilidade de uma solução programável com a performance de um hardware dedicado. Com FPGAs baseados em SRAM, mudanças podem ser feitas à lógica de um sistema simplesmente reconfigurando o FPGA residente no sistema. Logo, o potencial existe para o usuário desenvolver (através da reconfiguração do FPGA) exatamente o hardware especial necessário para uma dada tarefa sem ter que construir um novo hardware para cada aplicação. Erros podem ser corrigidos e diferentes abordagens algorítmicas exploradas, sem nenhum gasto extra com hardware. Por estas razões FPGAs são uma tecnologia viável para implementar tarefas de vídeo e outras tarefas que exigem alta performance. Em seguida veremos algumas implementações acadêmicas e comerciais que aplicam dispositivos programáveis em sistemas de processamento de sinais.

3.3.1 Exemplos de algoritmos implementados com FPGA

Diversas aplicações de processamento de sinais têm sido implementadas em FPGAs. A seguir são relacionados algumas destas implementações:

Peterson [Pet95] faz uma análise através de exemplos da competência de FPGAs como alternativa para implementação de sistemas de processamento de sinais. Entre as exigências computacionais comuns aos processamentos de sinais, está a necessidade de realizar cálculos, em especial multiplicação, rapidamente. Assim ele analisa diversas implementações de multiplicadores em FPGA, comparando a performance obtida com CIs multiplicadores dedicados. Estes multiplicadores são então utilizados na implementação de filtro FIR de 20 estágios, FFT complexa de 1024 pontos e filtro de convolução de imagem com núcleo 3x3 e 5x5. Estas aplicações são implementadas em dispositivos Xilinx e Altera. Os resultados são comparados com implementação em DSPs e com CIs dedicados.

Outra aplicação que exige grande poder computacional é o reconhecimento automático de alvos (ATR) em imagens de radar de abertura sintética (SAR). Em Rencher [Ren96], estes algoritmos foram usados como ferramentas para comparar duas plataformas de hardware reconfigurável, Splash-2, visto em detalhes na seção seguinte e Teramac. Os resultados de performance destas implementações é comparado com a implementação em software sobre uma workstation do mesmo algoritmo, mostrando

um ganho de performance de duas a três ordens de magnitude.

O mesmo algoritmo de ATR foi implementado por Ross [Ros97] em FPGAs Altera 10K50, que possuem memória RAM embutida. Ele estudou a utilização dessa memória RAM como elemento acelerador deste algoritmo. A grande capacidade deste dispositivo permitiu a realização de até oito elementos processadores em um só dispositivo, acelerando o processamento. A performance desta implementação é comparada com o trabalho visto acima e outras implementações e o resultado foi equivalente ou superior, mesmo com a utilização de um único dispositivo.

Grahan e Nelson [GN98] apresentam uma aplicação de conformação de feixe por atraso de tempo para sonar e discutem um sistema formado por vários FPGAs para realização deste método em tempo real. Como resultado obteve-se uma vantagem de performance de 6 a 12 vezes sobre um sistema equivalente construído com DSPs de alta performance projetados para sistemas de multiprocessamento (família SHARC da Analog Devices).

As aplicações de computação reconfigurável se estende às mais diversas áreas, incluindo implementações em hardware de redes neurais artificiais. Bade e Hutchings[BH94] apresenta uma arquitetura que torna praticável a implementação de grandes redes neurais, com milhares de sinapses, em FPGAs. São utilizadas técnicas de computação estocástica com uma arquitetura baseada em tabelas de busca. Com esta arquitetura é possível integrar 480 conexões sinápticas em um único Xilinx 4010.

Elredge [EH94] apresenta uma arquitetura que acelera a execução de redes neurais no algoritmo de backpropagation. Para aumentar o número de neurônios de hardware é utilizada a reconfiguração em tempo de execução, que permitiu a implementação de seis neurônios por FPGA. Esta arquitetura atinge grandes velocidades de aprendizagem (mudanças de pesos por segundo) e de operação (conexões por segundo).

Atualmente muito está sendo pesquisado sobre a utilização de computação reconfigurável com FPGAs para acelerar diversas aplicações que exijam grande poder de computação. Encontram-se aplicações em criptografia, compressão de dados, redes de alta performance, processamento de sinais e de imagem, prototipagem rápida, realidade virtual, multimídia, etc. A grande dificuldade do uso desta tecnologia reside na programação. Para se explorar todo este poder computacional é necessário conhecimento de técnicas de engenharia de hardware. O uso deste tipo de solução será mais difundido com o aparecimento de métodos de programação capazes de abstrair a arquitetura envolvida com conceitos puramente de software. Isto tornaria prática a utilização de computação reconfigurável por profissionais de informática.

3.4 Sistemas comerciais

Diversos sistemas comerciais de prototipagem rápida e computação reconfigurável baseados em FPGA existem no mercado. A seguir veremos umas descrição de três destes sistemas muito importantes: o SPLASH-2 desenvolvido no Centro de Pesquisa de Supercomputação em Bowie, Maryland, a plataforma de computação reconfigurável Spectrum da Giga Operations Corp. e Transmogrifier-2, desenvolvido no Departamento de Engenharia Elétrica e de Computação da Universidade de Toronto.

3.4.1 SPLASH-2 e VTSPLASH

O sistema Splash-2 é classificado como um processador auxiliar uma vez que para que possa operar ele deve ser agregado a uma máquina hospedeira através de um barramento de expansão. Ele difere de um coprocessador por que ele não reside diretamente no barramento do processador hospedeiro. O SPLASH-2 é composto de uma placa de interface (para formatação e buffers de entrada e saída), de uma a quinze placas processadoras e de uma Estação de trabalho Sun SparcStation-2 como hospedeira. Cada placa contém 16 PEs (Elementos Processadores), denominados como X1 a X16, arranjados linearmente e totalmente conectados através de uma matriz de interconexões de 16 x 16. Um décimo sétimo elemento de controle, X0, regula esta matriz. A Figura 3.1 mostra um diagrama em blocos do sistema.

Cada PE em uma placa Splash-2 (identificado com X1 a X16 na Figura 3.1 e expandido na Figura 3.2) consiste de um FPGA e uma memória rápida. O FPGA Xilinx XC4010 usado em cada PE consiste de um matriz 2D de blocos lógicos configuráveis que podem ser conectados internamente com recursos de interconexão reconfiguráveis. Tanto os blocos lógicos quanto os recursos de interconexão são programáveis através do computador hospedeiro. As operações computacionais são implementadas como circuitos lógicos construídos dentro do FPGA em blocos individuais e então reconectados da forma necessário usando as chaves programáveis. Uma RAM estática (SRAM) rápida de 256K x 16 é acrescentada a cada FPGA, que permite um acesso de leitura ou escrita em um ciclo de relógio. Cada PE tem 3 caminhos de dados bidirecionais de 36 bits; um para cada PE vizinho da esquerda e da direita e um para a rede de interconexões. Além disso, existe um caminho de 16 bits entre o FPGA e a SRAM.

O fluxo de dados de entrada para o sistema processador Splash-2 é fornecido pela placa de interface com um barramento SIMD de 36 bits para o X0 de cada placa processadora. O fluxo de dados de saída pode ser ligado a múltiplas placas processadoras ao se estender este fluxo do X16 de uma placa ao X1 da próxima.

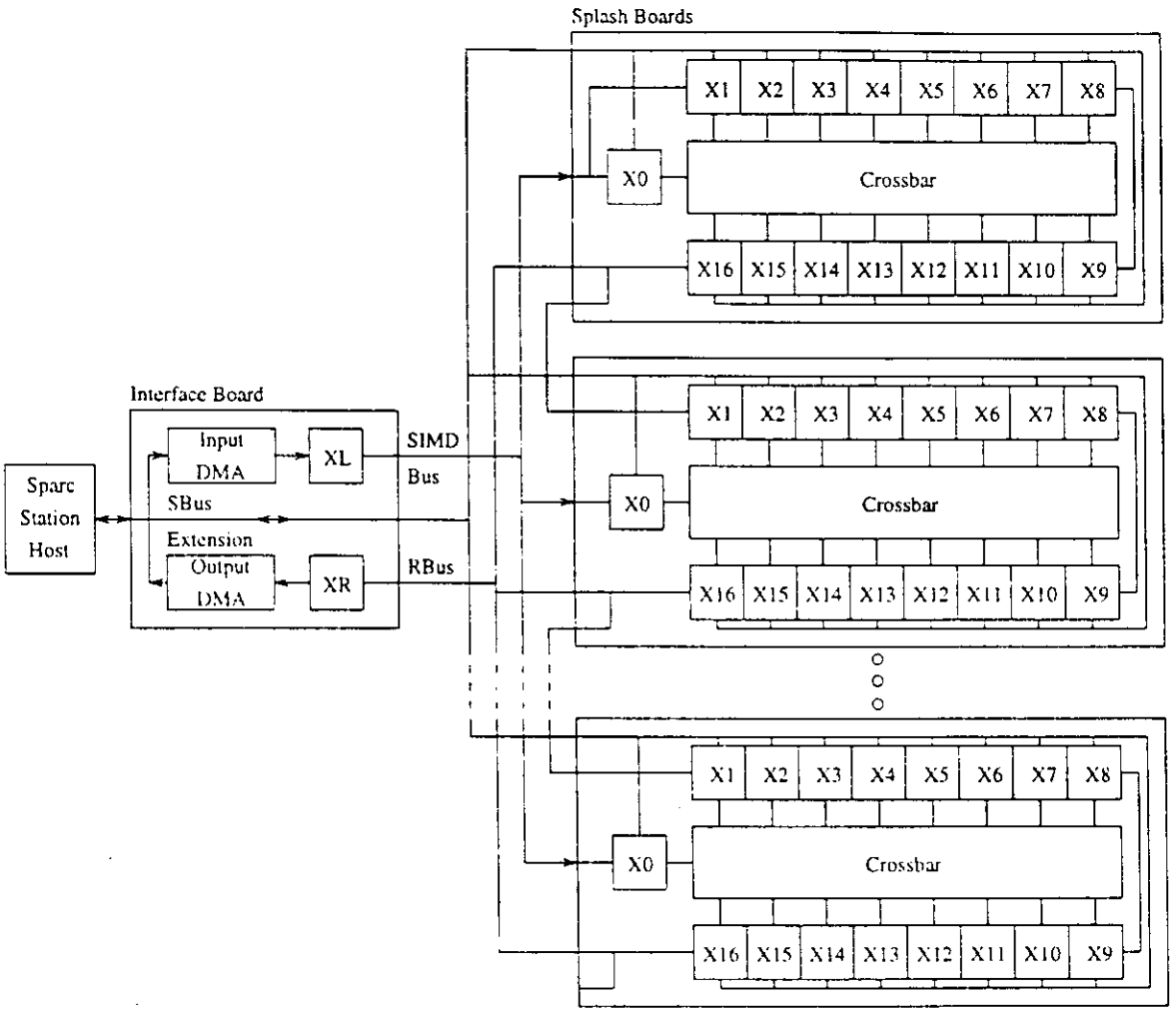


Figura 3.1: Arquitetura Splash-2

A rede de interconexões contém 16 portas bidirecionais de 36 bits para melhorar a comunicação entre processadores. As conexões programáveis do Splash-2 podem ser usadas para ajustes estáticos e dinâmicos da arquitetura. Os ajustes estáticos estabelecem os caminhos de dados para tarefas fixas tipo sistólicas, enquanto que os ajustes dinâmicos acomodam paradigmas mais complexos de movimento de dados. O sistema SPLASH-2 oferece uma alternativa atrativa a arquiteturas tradicionais. Com esta plataforma computacional, não apenas as operações podem ser projetadas de forma dedicada (por função e tamanho), como os caminhos de dados podem ser dedicado para aplicações individais.

Muito embora o Splash-2 não tenha sido projetado especificamente para processamento de imagem, suas propriedades arquiteturais são adequadas as taxas de transferência de dados e computação características dessa classe de problemas. Usando o SPLASH como base, Athanas [AA95] construiu o VTSPLASH, um sistema laboratório para processamento de imagem em tempo real . As imagens são adquiridas por uma

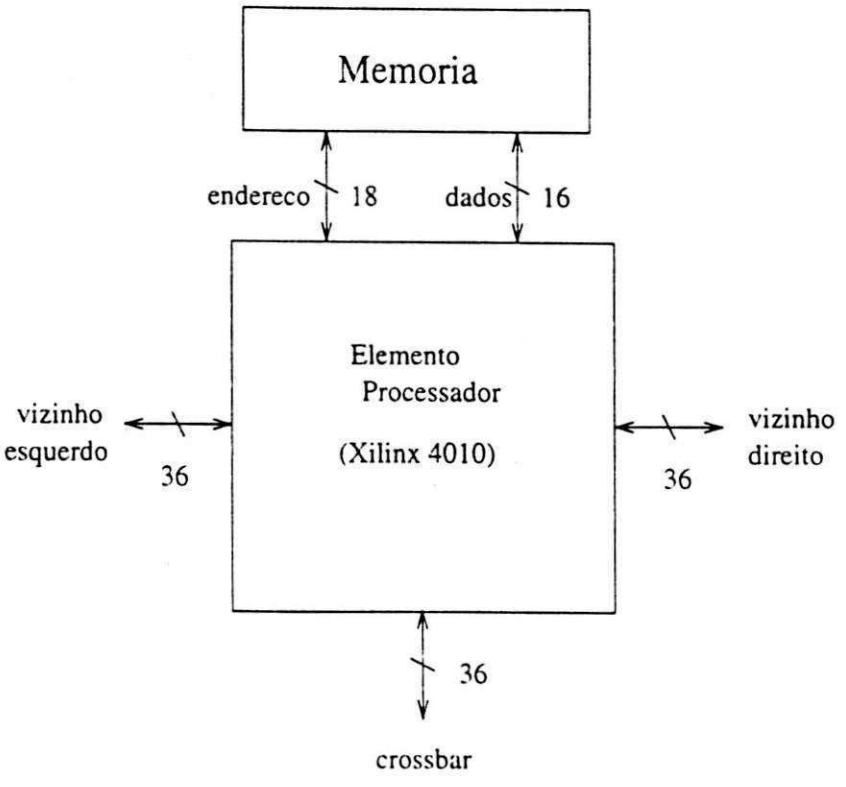


Figura 3.2: Elemento processador do Splash-2.

câmara de vídeo ou videocassete que cria um fluxo de vídeo padrão RS-170. O sinal produzido a partir da câmara é digitalizado com uma placa de captura de quadros (*frame-grabber*). Esta placa não apenas captura as imagens como também realiza algum sequenciamento ou operação simples com os pixels antes que os dados sejam apresentados ao SPLASH-2.

Os dados de saída produzidos pelo SPLASH-2, que podem ser um fluxo de dados de vídeo em tempo-real ,é primeiro direcionado a outra placa para converter os dados para outro formato, se necessário. Uma vez formatados, os dados são então apresentados em um monitor de vídeo.

3.4.2 Spectrum

A plataforma de computação reconfigurável Spectrum foi desenvolvida pela Giga Operations para promover uma arquitetura aberta e uma plataforma padrão para computação reconfigurável. Ela tem uma arquitetura de "hardware virtual" aberta. Os mesmos componentes de hardware podem ser programados para executar muitas arquiteturas de hardware. Esta plataforma é formada por 3 componentes de hardware:

- XMODs - contêm FPGAs e memória para realizar o processamento dos dados e gerenciamento de E/S. Cada XMOD fornece 130 pinos de E/S reprogramável.

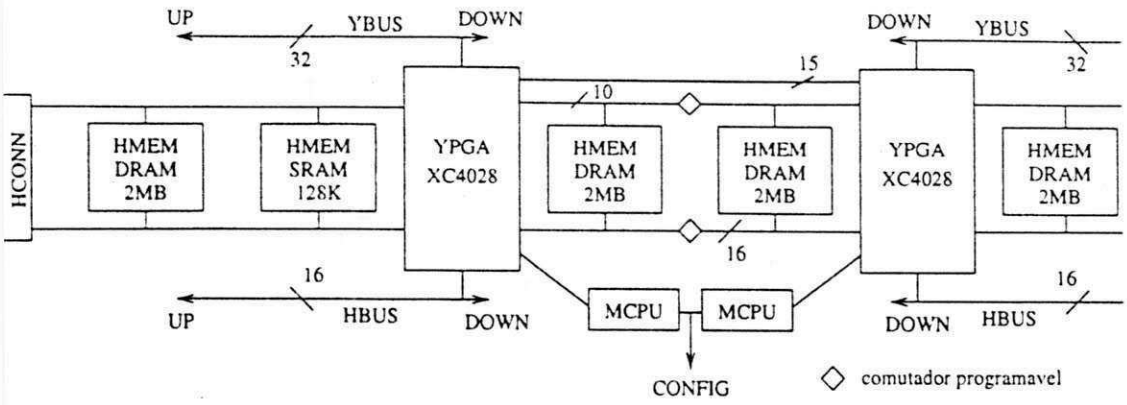


Figura 3.3: Diagrama em blocos de um XMOD

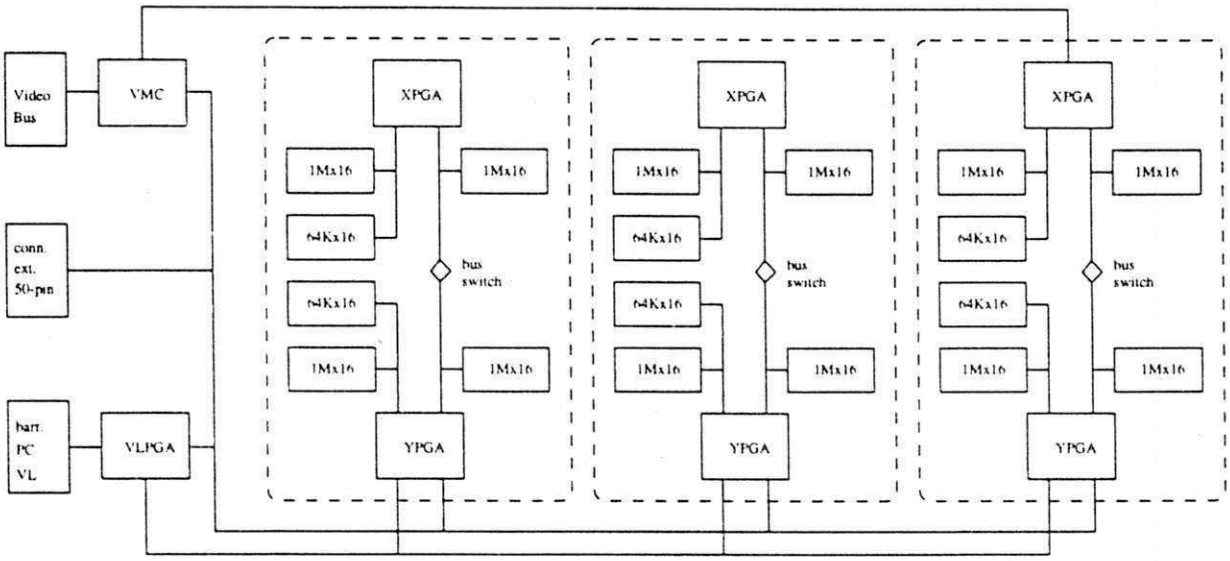


Figura 3.4: Diagrama em blocos da RIC-800

- Módulos de E/S - fornecem canais de E/S como vídeo, conversão A/D, etc.
- Placas de interface reconfiguráveis (RIC) que acomodam até 16 XMODs ou Módulos de E/S e fornecem interfaces para o barramento do computador hospedeiro e dispositivos periféricos.

Os XMODs são pequenas placas de processamento de dados, contendo 2 FPGAs XILINX. O diagrama em blocos de um XMOD pode ser visto na figura 3.3.

Estes módulos são conectados às RICs, que podem suportar 4 pilhas com 4 XMODs cada, perfazendo um total de 16 XMODs. Cada XMOD possui grande quantidade de memória local dinâmica (8MB) e estática rápida (256K), muitos pinos de E/S e geração de sinais de relógio. O diagrama em blocos de uma placa RIC para barramento Vesa pode ser visto na Figura 3.4.

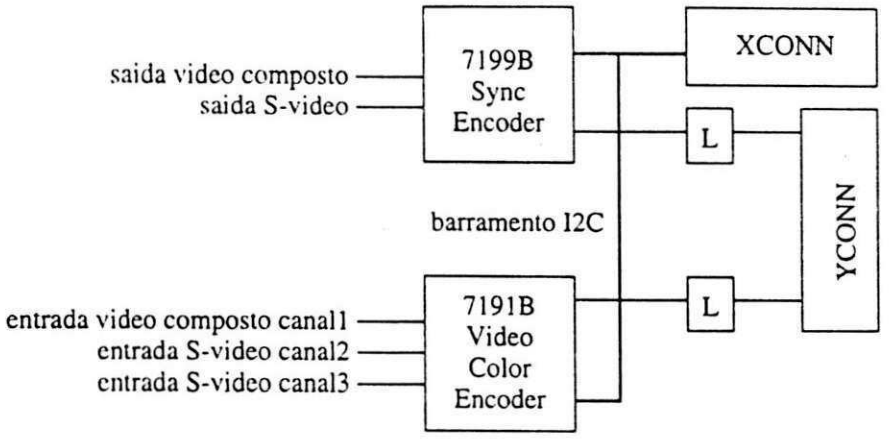


Figura 3.5: Diagrama em blocos do VID-MOD

O módulo de entrada e sáida de vídeo, chamado VIDMOD, é capaz de receber e gerar sinais de vídeo composto. Ele possui um circuito de captura de imagens com um decodificador de NTSC e PAL que gera um fluxo de dados sequencial. Possui também um codificador para NTSC e PAL que gera sinal composto analógico a partir do fluxo de dados. O diagrama em blocos desta placa pode ser visto na Figura 3.5.

A arquitetura Spectrum é aberta e altamente estruturada e escalonável. No entanto, possui deficiências de ordem prática. Uma de suas principais deficiências é a utilização de FPGAs da linha XILINX, que apesar de serem o maior vendedor mundial, seus dispositivos não apresentam as melhores arquiteturas e seus preços atualmente são muito elevados.

3.4.3 Transmoglifier-2

O transmogrifier-2[LGea97], ou TM-2 é um exemplo de um grande sistema de prototipagem rápida multi-FPGA para 1 milhão de portas. A maior versão do sistema é formado por 16 placas cada uma com dois FPGAs 10K50 da Altera, quatro dispositivos de interconexão da I-cube e até 8 Mbytes de memória. A arquitetura do TM-2 é modular e escalável. Isto significa que a arquitetura deve ser capaz de ser usado em sistemas de vários tamanhos, até um limite superior e que o sistema é construído a partir de alguns módulos idênticos. O sistema inclue um relógio programável que permite formas de onda de relógio arbitrárias com resolução de 10ns.

As principais vantagens deste sistema são a modularidade e escalabilidade, a grande capacidade (1 milhão de portas), o atraso predizível nas conexões entre dispositivos, a grande largura de faixa de acesso a RAM e a alta velocidade de operação.

3.5 Conclusão e motivação

Este capítulo enfatizou a grande importância da computação reconfigurável e da prototipagem com FPGAs. Para obtermos domínio nesta tecnologia de evolução tão rápida, optou-se pela construção de um sistema próprio em lugar de adquirirmos um já existente. O sistema desenvolvido é o tema dessa dissertação. Trata-se de uma placa de prototipagem rápida, com utilidade genérica, porém adequada ao processamento de imagens em tempo real.

No próximo capítulo veremos detalhes da implementação deste sistema.

Capítulo 4

Implementação do Sistema de Prototipagem

Com o objetivo de adquirirmos domínio da tecnologia envolvendo computação reconfigurável e prototipagem com FPGA, foi desenvolvido um sistema de prototipagem rápida dirigido ao processamento de imagens em tempo real, denominado SPRIT. Neste capítulo as especificações deste projeto são analisadas, a arquitetura proposta é apresentada juntamente com as decisões tecnológicas sobre os diversos elementos. Em seguida são apresentados os elementos constituintes a nível de bloco funcional e são relacionadas as características do layout da placa. Ao final do capítulo são formuladas algumas conclusões.

4.1 Especificações do SPRIT

O projeto do SPRIT foi guiado pelas seguintes especificações:

Modularidade e escalabilidade Por escalável se entende que a arquitetura pode ser configurada para ser utilizada em uma grande faixa de aplicações. O termo modular se refere a forma com que estes sistemas são construídos e significa que todos os sistemas são construídos com módulos idênticos.

Média capacidade, FPGAs modernos O sistema deve ser capaz de conter implementações de baixo nível de processamento de imagens, ou uma cadeia destes. Isto significa um número total de portas lógicas utilizáveis entre 20 e 200 mil.

Atraso entre chips O atraso devido a interconexão entre os dispositivos não deve limitar a performance do sistema, de forma a permitir que um projeto seja dividido em diversos dispositivos sem muita perda de velocidade.

Alta velocidade O sistema deve ser capaz de operar em altas frequências de relógio, possibilitando que o algoritmo possa ser implementado com vários pulsos de relógio por pixel de entrada.

Relógio programável O sistema deve ser capaz de gerar um número adequado de sinais de relógio, de várias frequências programáveis.

Memória RAM O sistema deve possuir grande quantidade de memória RAM de acesso rápido e uma quantidade menor de RAM de acesso muito rápido. Isto permite o armazenamento de mais de uma imagem inteira para processamento de algoritmos que utilizam sequências de imagens e também permite a utilização da memória muito rápida para os processamentos com vários acessos a memória por pixel de entrada e também como memória cache.

Alta taxa de E/S Ao fornecer grande quantidade de pinos de interface com o mundo exterior, o sistema possibilitará uma alta taxa de transferência de E/S. Isto é muito importante para garantir a escalabilidade do projeto.

Amigável com o usuário O sistema deve aproveitar as ferramentas disponíveis da melhor forma possível a fim de tornar o sistema de fácil aplicação.

Tempo de programação O tempo de carga do programa nos FPGAs deve ser pequeno, menor que 10 s.

Manufaturabilidade O circuito final deve ser de fabricação simples de baixo custo, usando tecnologia padrão de placas e componentes comerciais, de fácil aquisição.

O fluxo de projeto utilizando um sistema com essas características iniciaria com uma entrada de dados que descreva o hardware, normalmente através de uma linguagem de descrição de hardware ou entrada esquemática. Em seguida é realizada uma síntese que transforma esta descrição em informação de configuração dos dispositivos FPGA. O passo seguinte é descarregar esta informação na placa utilizando um programa desenvolvido para esta tarefa. Após a configuração o algoritmo pode ser verificado com entrada de valores e observação dos resultados. Uma descrição mais detalhada da metodologia de trabalho envolvida no desenvolvimento utilizando este sistema será visto no próximo capítulo. Diversos testes foram realizados com o sistema construído e os resultados podem ser vistos no capítulo 6.

4.2 Arquitetura do SPRIT

Durante a seleção de uma arquitetura adequada aos objetivos do SPRIT, foram consideradas como referência as arquiteturas comerciais citadas no capítulo 3. Podemos observar, que além do número máximo de portas implementáveis, estas arquiteturas também se destacam na forma com que os elementos processadores podem ser interconectados. Podemos interconectar os processadores FPGA em cadeia, barramento, malha unidimensional ou através de dispositivos crossbar como FPIDs, como visto nas Figuras 4.1, 4.2, 4.3 e 4.4. A escolha de um determinado tipo de interconexão depende do tipo de processamento que se deseja implementar, avaliando-se um compromisso entre a simplicidade e a flexibilidade. A alternativa mais flexível é o crossbar porque permite que todos os dispositivos se interconectem, e também tenham acesso a quaisquer pinos de E/S ligados a ele. Um crossbar programável dinamicamente também permite o acesso de sinais para depuração. O uso de um crossbar permite considerar um conjunto de FPGAs como um único FPGA maior, totalmente reconfigurável. As demais arquiteturas possuem a vantagem da simplicidade e podem ser completamente eficientes para determinados tipos de processamento.

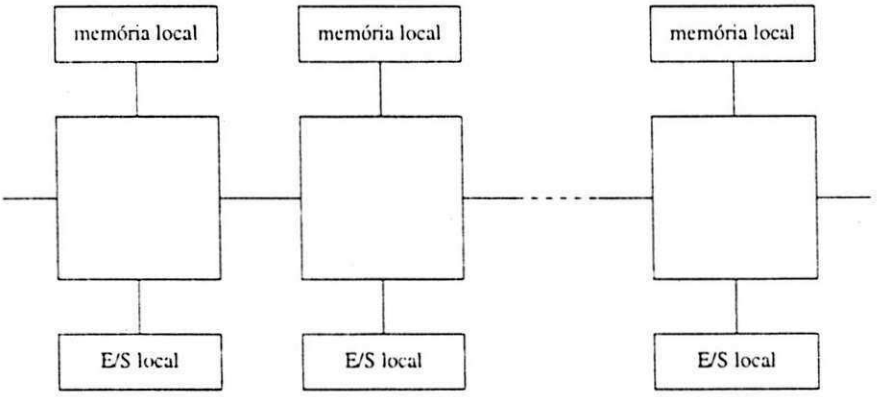


Figura 4.1: Interconexões em cadeia

A necessidade de um esquema de interconexão mais sofisticado se relaciona com o número de dispositivos necessários para realizar as implementações, ou seja, na relação entre as dimensões do circuito e dos dispositivos FPGA utilizados. Com o crescimento atual dos dispositivos, é possível realizarmos um número cada vez maior de circuitos de dimensões medianas com um ou dois FPGAs. Como as interconexões internas aos FPGAs são programáveis, um dispositivo maior substitui vários menores e um crossbar.

A decisão arquitetural então depende da abordagem utilizada para o processamento. O modelo arquitetural do SPRIT é otimizado para uma abordagem baseada no uso de pipeline com um modo de operação de fluxo de dados síncrono, composto de operadores

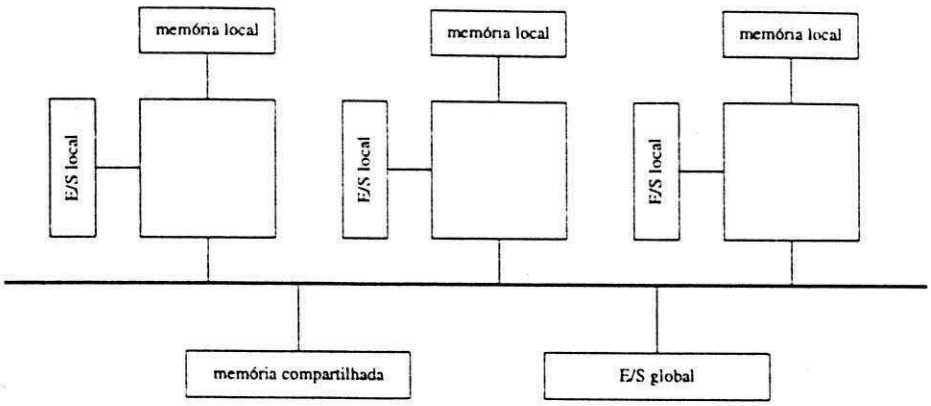


Figura 4.2: Interconexões em barramento

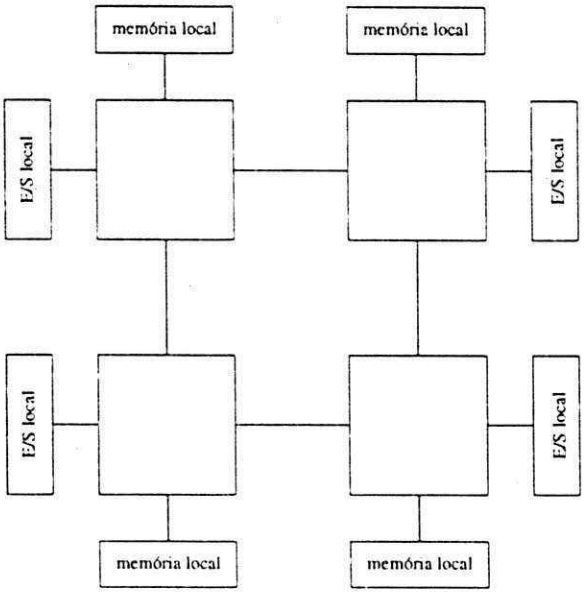


Figura 4.3: Interconexões em malha unidimensional

específicos básicos encadeados. Este modelo é direcionado para aplicações em que as exigências de performance são impostas primeiramente pelo *throughput* (operação em tempo real) em lugar do tempo de computação (latência). A estrutura proposta é justificada pelos seguintes aspectos:

1. Os operadores básicos que processam os dados a nível de pixel podem ser decompostos de forma sucessiva em sub-operadores até o nível de operadores aritméticos e lógicos elementares, pela inserção de registradores adicionais de pipeline.

2. O gerenciamento do nível de pipeline entre os operadores internos, torna possível fracionar os atrasos críticos dos circuitos em seções menores de forma que os circuitos possam ser adaptados para serem clocados a taxas de acordo com as exigências do sistema.

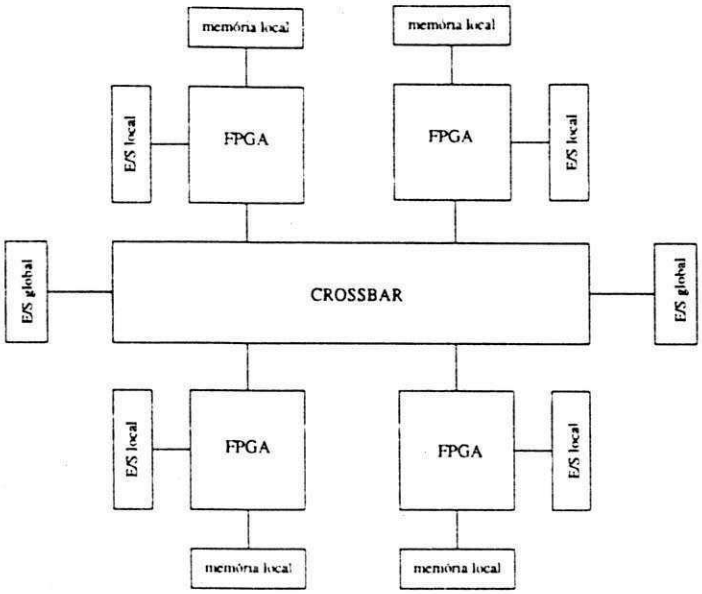


Figura 4.4: Interconexões com um crossbar

3. Para as tarefas de processamento de imagens de baixo nível é útil usar sinais de vídeo diretamente no formato de varredura. Este formato também é usado na saída dos operadores. O modelo arquitetural proposto mantém a uniformidade dos fluxos de dados.

4. O controle nesta arquitetura é muito simples. Poucos ou mesmo nenhum laço são executados, de forma que as mesmas operações são executadas em cada amostra de dados. Este controle pode ser facilmente gerado a partir dos sinais de sincronismo de vídeo.

5. O uso de FPGAs modernos, de grande capacidade, permite que mais de um operador básico seja implementado em um único dispositivo.

A arquitetura de uma placa SPRIT pode ser visto no diagrama de blocos apresentado na Figura 4.5. Os dois FPGAs estão conectados de forma que possam processar em cadeia, com diversos pinos de interconexão e também possuem barramentos que permitem a comunicação com o host a uma taxa de transferência relativamente alta. Cada FPGA possui bastante recursos de memória local e de E/S. Este diagrama de blocos será analisado mais detalhadamente na seção 4.4.

O modelo arquitetural do SPRIT consiste de uma cascata de operadores básicos colocados em uma matriz linear de módulos físicos composto de um bloco de circuitos FPGA e recursos de memória interconectados. Os circuitos FPGA são programáveis pela tecnologia SRAM. Cada bloco de circuito representa o principal componente de um módulo de cadeia e integra um operador e um conjunto de operadores arranjados

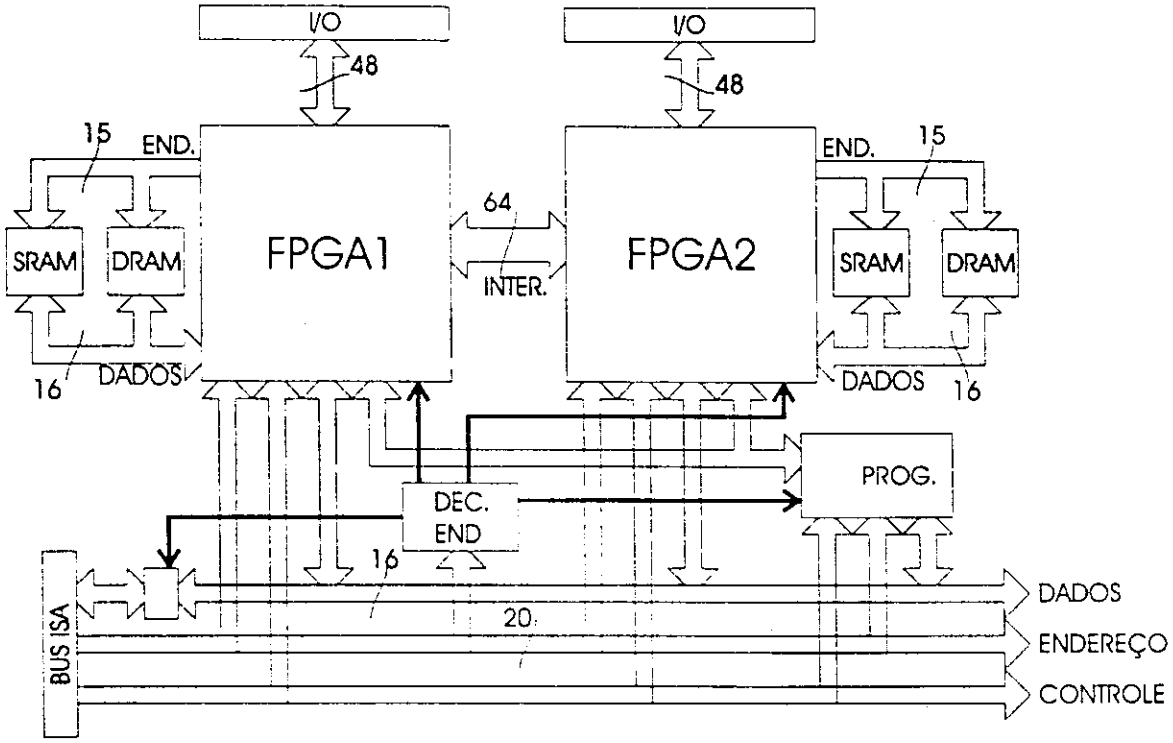


Figura 4.5: Diagrama de blocos do sistema desenvolvido

em pipeline ou em paralelo. O sistema como um todo tem um modo de operação de fluxo de dados e é controlado pela amostragem de pixels na cadeia de entrada (sistema de aquisição) e/ou pelos sinais de controle do processador hospedeiro. Cada módulo tem dois tipos de recursos de memória. As memórias FIFO (implementadas logicamente) são associadas a funções de linha de atraso nos operadores locais (operadores de vizinhança). As memórias RAM de maiores dimensões são associadas para intermediar dados de imagem em funções globais (operadores globais e transformações de imagem).

O tamanho da matriz é determinado pelo compromisso entre o número de estágios de pipeline da arquitetura e a área efetiva disponível em cada módulo. O número de estágios de pipeline é determinado pelas exigências de velocidade da aplicação. A flexibilidade é atingida ao se explorar a programabilidade dos circuitos FPGA utilizados.

Desta forma, com o uso de 2 FPGAs de 20 mil a 70 mil portas implementáveis por dispositivos, é possível implementar vários operadores básicos por placa, utilizando-se a metodologia descrita acima. Estes FPGAs de alta densidade substituem diversos menores e um sistema complexo de interconexões. No entanto, se circuitos maiores forem desejados, o sistema construído pode ser escalado com o uso de mais de uma placa. A modularidade é obtida através do uso dos operadores básicos a nível de descrição do projeto.

4.3 Decisões tecnológicas

Diversas decisões relativas a implementação do SPRIT tiveram de ser tomadas, sempre levando em consideração as metas citadas na seção 4.1. As decisões mais importantes são relativas a família de FPGA empregada, o barramento para comunicação com o host e os demais dispositivos periféricos.

4.3.1 Família de FPGA empregada

As qualidades tecnológicas nem sempre são o principal motivo de decisão na escolha da família de dispositivos empregada em um projeto eletrônico. Certas considerações de ordem prática se somam as de ordem tecnológica e influenciam na escolha.

Na escolha da família de FPGAs para o projeto SPRIT, as seguintes características foram analisadas:

Fabricante Uma das principais características buscadas foi utilizar uma família de FPGAs cujo fabricante esteja a algum tempo neste mercado, que possua as características de modernidade, solidez e confiabilidade.

As empresas que mais se destacam neste mercado são a Xilinx e Altera. Outros fabricantes analisados foram a Actel, Quicklogic, AT&T e Lattice.

Representante no Brasil A existência de um representante no Brasil é muito importante devido a possibilidade de aquisição de componentes em pequena quantidade e suporte técnico.

Na época da aquisição dos componentes, apenas a Xilinx e a Altera possuíam representante no país.

Disponibilidade de software Além da disponibilidade de um bom hardware, o sucesso na utilização de dispositivos FPGA depende muito de um software que explore suas capacidades. Este software deve ser uma solução completa baseada em PC capaz de realizar síntese de HDL e entrada esquemática, chegando até a geração do bitstream (arquivo de configuração) e a programação do dispositivo. Além disso, o software deve possuir uma boa interface com os outros ambientes de desenvolvimento existentes na UFPB, ou seja, os ambientes de trabalho Cadence e Synopsys.

Através dos programas universitários das empresas, foram cedidas à UFPB cópias dos softwares da Xilinx e da Altera. Destes, apenas o Max+plusII, solução completa da Altera, não possuía limitações de licença. Este software possui vantagens

significativas sobre o XACT da Xilinx. Ele apresenta uma boa interface com o usuário, com todos os módulos integrados em um único ambiente. Também possui diversas possibilidades de entrada de projeto, através de linguagem de descrição de hardware com AHDL, VHDL e Verilog, entrada esquemática e híbrido. Realiza gerenciamento da hierarquia, projeto multi-dispositivos, simulação lógica e análise temporal, *floorplanning* e roteamento automático e manual e síntese dirigida pela temporização.

Além destas vantagens, o Maxplus-II possui uma boa interface com Cadence e Synopsys. Isto permite que projetos maiores e simulações mais intensivas sejam realizados nestes sistemas.

Tendência do mercado Apesar da Xilinx ainda dominar o mercado de FPGAs, avaliando-se os projetos de sistemas de prototipagem rápida e computação reconfigurável mais recentes, como o Transmoglifier-2, visto na seção 3.4.3, nota-se uma tendência de utilização dos dispositivos Altera, mais densos e com algumas vantagens tecnológicas.

Preço Os dispositivos Xilinx da mesma faixa de densidade e de velocidade, na época da decisão, custavam de duas a cinco vezes o preço do equivalente da Altera.

Além das vantagens de ordem prática acima citadas, os dispositivos Altera possuem diversas vantagens tecnológicas que devem ser destacadas. Um diagrama simplificado da estrutura dos dispositivos Altera da linha Flex 10K pode ser visto na Figura4.6. A estrutura de um dispositivo da linha XC4000 da Xilinx é mostrado na Figura 4.7 [Xil94].

Interconexões As interconexões nos dispositivos da linha 10K da Altera estão distribuídos em dois níveis, com conexões locais a LABs e grande quantidade de linhas contínuas que travessam toda uma linha ou coluna, denominado de *Fast-Track*. Este tipo de interconexão de roteamento contínuo possui atraso pequeno e previsível. Além das estruturas locais e globais para roteamento, os dispositivos possuem linhas de cadeia de carry dedicadas para implementação de somadores, contadores e comparadores rápidos, e de cascateamento lógico, para implementação de funções lógicas de alta velocidade. A linha 10K também possui linhas dedicadas para o roteamento de sinais globais como relógio(*clock*) e *clear*.

Por outro lado, uma das principais desvantagens dos dispositivos da linha 4000 da Xilinx é que os atrasos gerados pelas interconexões não são completamente previsíveis, por serem segmentadas em vários níveis, através de matrizes de

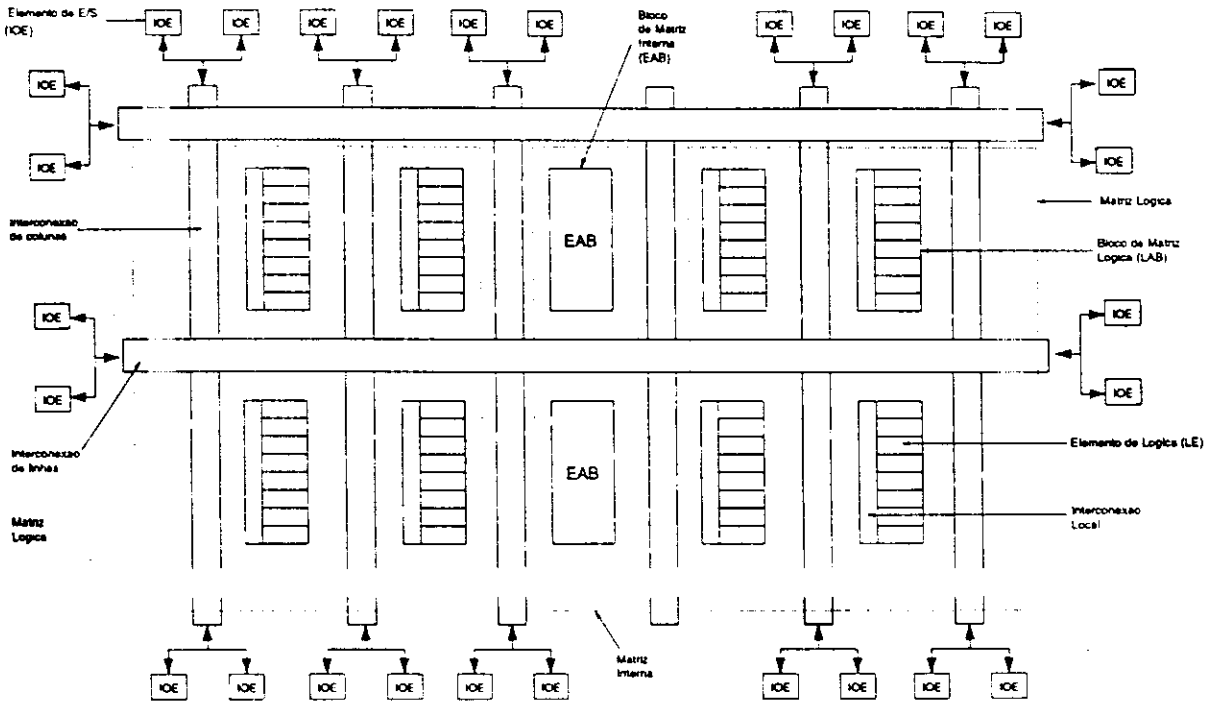


Figura 4.6: Estrutura interna dos dispositivos Altera linha Flex 10K

chaveamento. Os dispositivos Xilinx utilizam uma estrutura hierárquica de interconexões em três níveis, com conexões de tamanhos simples, duplo e linhas longas. As conexões entre estes elementos é realizada através de matrizes de chaveamento, que são conjuntos de elementos de passagem. Estas matrizes de chaveamento introduzem atrasos não previsíveis que são dependentes do tipo de conexão utilizada e da carga. Estes blocos são configurados através de células SRAM e são responsáveis por grande parte do atraso nas linhas locais.

Memória interna Uma outra característica da família Flex 10K da Altera é a existência de memória RAM embutida, de tamanho razoável, denominado de EAB (Embedded Array Block). Este bloco pode ser usado como memória de grande velocidade ou para implementar lógica. A utilização de um EAB não diminui a capacidade de implementação lógica básica do dispositivo. Os EABs podem ser utilizados para implementar lógica complexa, como multiplicadores, conversores de dados e funções de DSP.

Nos dispositivos Xilinx, cada CLB (Configurable Logic Block), que é a menor unidade configurável de cada chip, pode implementar qualquer lógica com 4 variáveis binárias através de uma tabela de busca. Esta mesma tabela pode ser configurada como uma pequena memória RAM de alta velocidade. Esta memória pode ser expandida utilizando-se vários CLBs. Quando a tabela de um CLB é utilizado

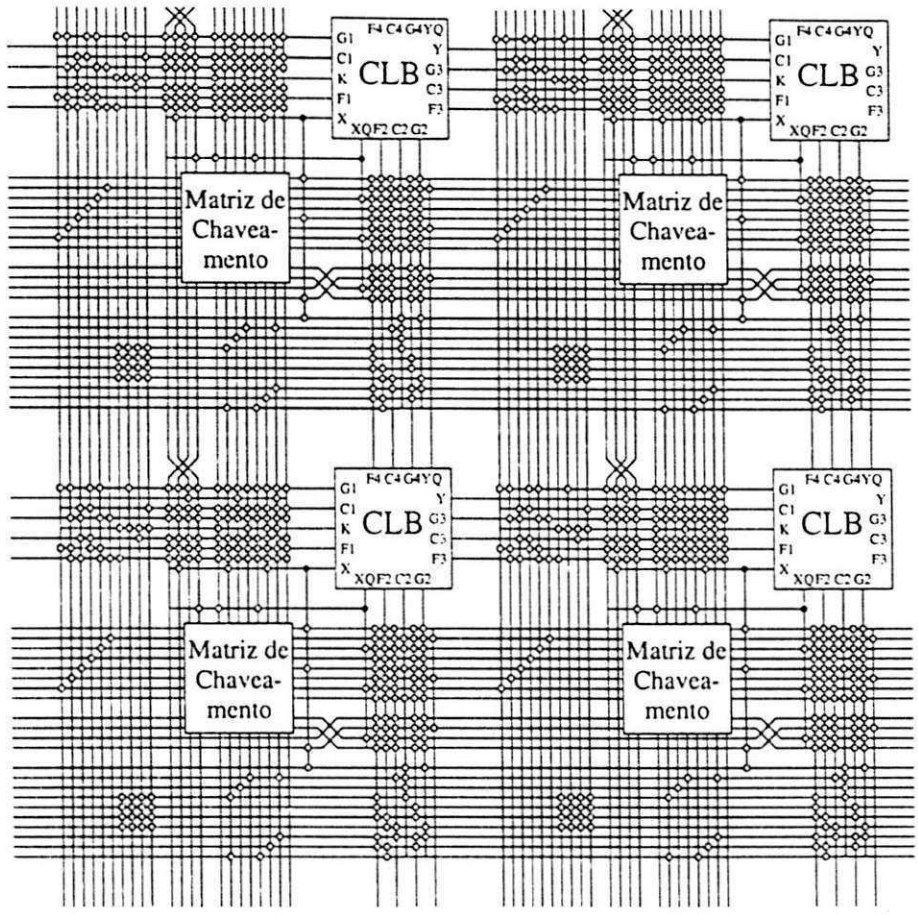


Figura 4.7: Estrutura interna de dispositivos da linha Xilinx 4000

como memória, este não pode ser usado para implementar lógica.

Comparando-se estas duas possibilidade de implementação de memória embutida, percebemos desvantagens na abordagem da Xilinx, devido a grande necessidade de interconexão, um recurso escasso em FPGAs, e a diminuição dos recursos para implementação lógica.

Levando-se em conta estes diversos fatores, decidiu-se por utilizar dispositivos da família Flex 10K da Altera. Analisando-se as possibilidades de encapsulamento existentes, optamos por utilizar SMD de 240 pinos (RC240), haja visto que com este encapsulamento dispomos de uma gama mais ampla de opções, com dispositivos entre 20 mil e 70 mil portas equivalentes. Devido as dificuldades de soldagem destes componentes, foram analisados inicialmente os encapsulamentos de montagem em orifícios, como PGA e os soquetes de adaptação de SMD para PGA. No entanto, estes dispositivos e adaptadores são muito caros. A solução encontrada foi o domínio da tecnologia de soldagem de dispositivos SMD.

4.3.2 Barramento

Ao decidir-se que tipo de barramento será usado para comunicação entre a placa SPRIT e o seu microcomputador hospedeiro, foram avaliados diversos aspectos. Os mais importantes são a taxa de transferência, a existência de padrão, a complexidade de implementação e a duração prevista.

Os principais barramentos encontrados atualmente nos microcomputadores pessoais são ISA (Industry Standard Architecture) e PCI (Peripheral Component Interconect).

As principais características do barramento PCI são:

- Acoplado ao barramento local da CPU através de uma ponte;
- Permite alta taxa de transferência - 133Mbytes/s (em rajadas);
- Suporta Plug-and-Play;
- Multi-master/Bus-Master;
- É praticamente o novo padrão entre microcomputadores pessoais;
- Implementação muito complexa, com requisitos lógicos e temporais muito estritos;
- Exigência de driver de software específico.

As características do barramento ISA são:

- Barramento de E/S;
- Permite apenas baixas taxas de transferências (8 Mbytes/s);
- Suporta Plug-and-Play;É o padrão de fato e industrial, existente em praticamente todo computador pessoal atual;
- Implementação muito simples;
- Nenhuma exigência de software específico.

A decisão deve então ser baseada no compromisso entre a taxa de transferência necessária à aplicação e a simplicidade de implementação. Na arquitetura proposta, o fluxo de dados de alta velocidade promovido pelo entrada e saída de sinais de vídeo não trafega no barramento da máquina hospedeira. Esse fluxo deve ocorrer exclusivamente nos pinos de E/S do sistema. O hospedeiro tem acesso aos dispositivos programáveis

apenas para realizar a sua configuração, para submeter parâmetros e para receber resultados. A taxa de transferência é então muito resumida. Mesmo assim, para padrões de vídeo cuja taxa de transferência total seja inferior a 8MBytes/s, é possível realizar a transferência dos dados dos pixels em tempo real entre o processador e o sistema dedicado, embora este não seja o objetivo deste sistema. Esta metodologia permite que o processamento das imagens possa ser realizado inteiramente pelos dispositivos programáveis, ou em cooperação com o processador da máquina hospedeira, numa estrutura de co-projeto hardware-software.

Desta forma, optou-se pela implementação do sistema sobre o barramento ISA, de constituição mais simples. As pequenas exigências temporais do barramento simplificou o projeto elétrico e físico da placa, a escolha dos componentes periféricos, e a realização do software de programação.

4.3.3 Componentes periféricos

A escolha dos componentes auxiliares do sistema é baseada principalmente na possibilidade de aquisição e na simplicidade de utilização. Por isso, nos circuitos auxiliares foram utilizados dispositivos lógicos MSI TTL, soquetes de memória SIMM e de cache e gerador de relógio programável dos tipos utilizados em placas-mãe de microcomputadores pessoais.

4.4 Esquema elétrico e layout do SPRIT

Aqui será analisado com um pouco mais de detalhes o diagrama de blocos da Figura 4.5. O diagrama esquemático correspondente pode ser encontrado no Apêndice A, juntamente com o desenho do layout da placa. Como explicado anteriormente, os dois FPGAs possuem muitos pinos interconectados de forma a permitir uma grande interação para a realização de circuitos maiores. Além da conexão direta, os FPGAs estão conectados a quatro barramentos:

- Barramento de programação - interliga os pinos dedicados a programação;
- Barramento de endereços, conectado ao barramento ISA diretamente;
- Barramento de dados, com buffers e conectado ao barramento ISA. Permite a comunicação entre os processadores (entre si) e o host;

- Barramento de controle, são os sinais de controle do barramento ISA. Através deles é possível determinar que tipo de transação está sendo realizada no barramento.

Parte do controle da interface do sistema com o barramento ISA é realizado por dispositivos MSI TTL e parte deve ser programada nos FPGAs. Os dispositivos são responsáveis pela decodificação da parte mais significatica dos endereços, selecionando previamente uma região de memória de 32K bytes. O barramento de endereços permite que os FPGAs selecionem qualquer endereço dentro desta faixa. Isto é feito para controlar os buffers dos sinais de dados e também para permitir que aplicações que não se comuniquem com o barramento possam ser construídas, sem a inclusão de lógica de controle do barramento nos FPGAs e para que mais pinos de E/S fiquem disponíveis. As faixas de endereços são selecionáveis através de estrapes.

Os buffers do barramento de dados são necessários para suportar a carga do barramento e para isolar os FPGAs. A ativação dos buffers é realizado por circuito externo aos FPGAs para garantir seu estado e direção mesmo quando os FPGAs ainda não foram inicializados.

A programação dos FPGAs é realizada através de um circuito que está mapeado como E/S. O endereço de E/S pode ser configurado através de oito estrapes, selecionando oito linhas de endereçamento das dez disponíveis no barramento. Neste circuito encontramos uma interface paralela programável composta de três portas de oito bits (A,B e C). A porta C é programada como saída pelo software de controle e é responsável pelos sinais de programação DCLK, DATA e NCONFIG. Além disso, existe um sinal de saída de teste que controla apenas um LED. Os demais pinos desta porta controlam a frequencia do gerador de relógio programável. A porta B é programada como entrada e utilizada para receber os pinos de resultados da programação. A porta A é interligada a 8 sinais comuns aos FPGAs e pode ser utilizado para passagem de parâmetros, controle assíncrono dos FPGAs e também para receber resultados.

Os principais sinais de programação estão ligados a acionadores de LEDs, para permitir a visualização rápida do processo de programação. No circuito de geração de relógio temos três circuitos. Um gerador de relógio fixo e dois geradores com PLL programáveis. Estes geradores são os mais comuns encontrados em microcomputadores pessoais e podem gerar diversas frequências predefinidas. Apenas um desses dois geradores pode ser soldado na placa. A seleção entre a utilização do gerador fixo ou dos geradores programáveis é feita por estrape. A saída de relógio possui um casamento com a linha através do método de resistência série [Cor98].

Cada FPGA possui três barramentos para acesso a memória local, com 17 pinos de endereço, 16 pinos de dados e 6 sinais de controle. Neste barramento são colocados até 4 dispositivos de memória rápida (10 a 20 ns) de 32Kbytes, totalizando 64K × 16 por FPGA, para armazenamento local e cache. Ainda existem dois soquetes de memória tipo SIMM, que permitem a colocação de pentes de 1M × 8, totalizando 1M × 16, por FPGA.

Os sinais de E/S de cada FPGA são ligados a 2 conectores de 34 e 40 pinos respectivamente. Outro conector é utilizado para a interface JTAG, que permite o teste dos dispositivos e das interconexões e também a programação. Cada FPGA pode gerar um sinal de interrupção para o processador hospedeiro e a seleção de qual interrupção será sinalizada é realizada por estrapes.

Por tratar-se de um circuito complexo, que trabalha a altas frequências, alguns cuidados adicionais foram tomados durante a concepção do layout de circuito impresso, de acordo com as recomendações de [Cor98]. Foram utilizadas quatro camadas, sendo duas camadas de sinais e dois planos internos de alimentação. A tecnologia utilizada permitiu linhas de 8 mils de espesssura, com vias de 30 mil de diâmetro.

O ponto mais importante na concepção do layout consistiu da decisão de que encapsulamento utilizar para os FPGAs. Pelos motivos citados anteriormente, optou-se por utilizar o encapsulamento SMD de 240 pinos tipo RQFP, com pinos dos quatro lados espaçados de 0.65mm. Esta decisão se identifica com a tendência industrial de utilização de apenas SMD nas suas linhas de produção. A dificuldade residual de soldagem destes componentes foi solucionada com a aprendizagem de técnicas de soldagem em pequena escala de dispositivos SMD.

4.5 Conclusão

Neste capítulo foram apresentadas as especificações que direcionaram o desenvolvimento do sistema de processamento sugerido, a sua arquitetura, seu esquema elétrico e layout da placa. As decisões tecnológicas mais importantes, como a escolha do barramento ISA e da família de FPGAs da Altera foram destacadas.

O sistema implementado pode ser utilizado para o desenvolvimento circuitos de 20 a 140 mil portas com uma única placa. Circuitos maiores podem ainda serem implementados com a utilização de mais placas, interligadas pelos conectores de E/S.

A metodologia geral de concepção de circuitos digitais utilizando as ferramentas disponíveis de simulação e síntese será visto no capítulo seguinte, juntamente com a metodologia específica para estes sistema.

Capítulo 5

Metodologia de utilização do sistema desenvolvido

Iniciamos este capítulo com uma breve discussão das metodologias gerais de projeto de circuitos integrados de aplicação específica (ASICs), apresentando as técnicas de projeto, as ferramentas de software necessárias e que qualidades e conhecimentos que são exigidos dos profissionais. Traçamos algumas considerações sobre as ferramentas que existem atualmente e que características são necessárias para que estas ferramentas permitam gerenciar a complexidade dos sistemas a serem implementados. Relacionamos então as ferramentas analisadas com a metodologia empregada no projeto de circuitos com FPGAs que é analisada em seguida com mais detalhes. As características específicas do sistema desenvolvido são evidenciadas. Tratamos também da utilização do sistema de prototipagem no ensino de técnicas de projeto de ASICs e sistemas de computação reconfigurável. Ao final apresentamos algumas conclusões.

5.1 Metodologia de projeto de circuito de aplicação específica (ASIC)

5.1.1 Objetivos

Devido ao crescimento tecnológico da microeletrônica, os circuitos integrados desenvolvidos estão cada vez maiores, mais rápidos e mais complexos. Este crescimento se dá numa taxa muito elevada dobrando a complexidade a cada 18 meses [BP97]. Um dos fatores que limitam o crescimento é o fato de que o aumento de produtividade dos projetistas não cresce no mesmo ritmo que a tecnologia de fabricação. No gráfico da Figura 5.1[BP97], que mostra o número de transistores por chip e o número e uma

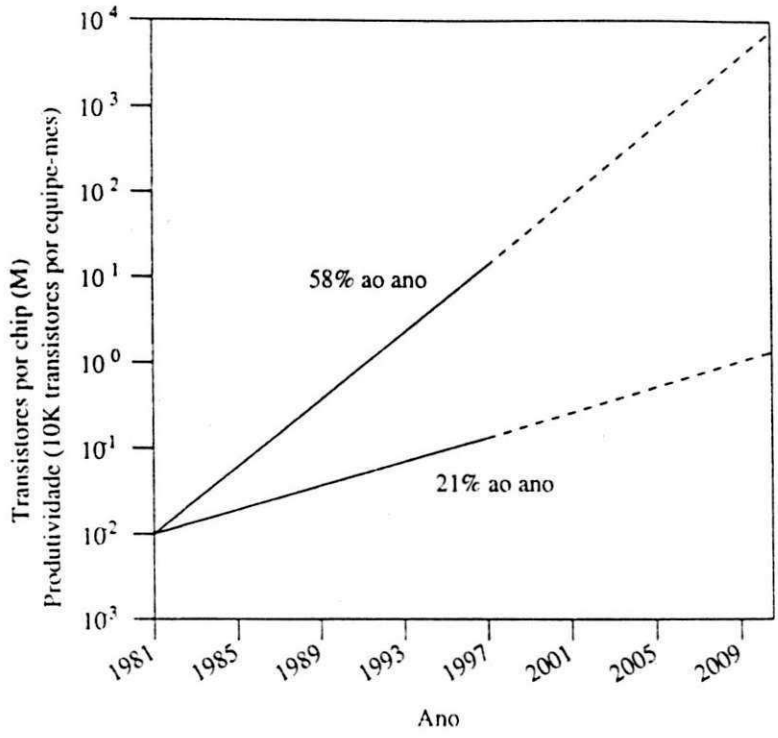


Figura 5.1: Comparação entre crescimento tecnológico e produtividade do projetista

medida da capacidade dos projetistas no tempo. Vemos que a discrepância tende a aumentar nos próximos anos. Como decorrência destes fatos, podemos ter a capacidade humana de gerenciamento como um fator limitante deste crescimento. Para aumentar a produtividade dos projetistas é necessário uma metodologia bem disciplinada, ferramentas mais poderosas e melhoria na capacitação dos profissionais.

Os objetivos de qualquer metodologia de projeto é maior eficiência, seja em velocidade, consumo, funcionalidade, flexibilidade, área de silício, custos de engenharia e de cronograma de projeto. As decisões sobre as diversas possibilidades de implementação devem considerar todos estes fatores e assumir uma solução de compromisso.

5.1.2 Metodologia

A definição de uma boa metodologia de projeto para circuitos integrados é essencial para reduzir o tempo necessário para o desenvolvimento do projeto e aumentar a probabilidade de sucesso na primeira fabricação do circuito. Diversos fatores têm de ser considerados na elaboração de uma metodologia de trabalho:

- Escolha de um estilo de projeto *top-down*, com o particionamento *hardware/software* bem planejado;
- Uso de uma linguagem de descrição de hardware, que atualmente permite uma

produtividade maior e uma maior taxa de acerto para grandes projetos, além da maior portabilidade, devido a possibilidade de descrições independentes da tecnologia;

- Escolha cuidadosa das ferramentas de projeto, haja visto o alto custo do investimento em ferramentas de projeto e sobretudo no treinamento dos profissionais para aquela determinada ferramenta;
- Avaliação contínua do projeto;
- Interação e comunicação intensiva com fornecedores e fabricantes;
- Verificação efetiva das regras de projeto.

Alguns princípios básicos se aplicam ao gerenciamento da complexidade no desenvolvimento de qualquer sistema e são muito utilizados no projeto de circuito integrados. São eles a **hierarquização**, obtida com a divisão do circuito em sub-módulos; a **regularidade**, através do uso de estruturas regulares, como memórias, *datapaths*, matrizes e células padrão; a **modularidade**, obtida através de interfaces com parâmetros bem definidos, como a temporização dos sinais e o efeito de carga aplicada; e a **localidade**, com a otimização das interconexões locais e o uso de sinais de interface com registros. Estes princípios devem ser apoiados pelas ferramentas e são mais facilmente obtidos com o uso de linguagens de descrição de hardware.

Além de uma metodologia bem definida, para gerenciar a complexidade cada vez maior são necessários novos desenvolvimentos em ferramentas, tanto com relação a algoritmos quanto a interface com o usuário. Um caminho sendo seguido é a entrada de descrição a níveis de abstração cada vez mais altos, com o desenvolvimento de ferramentas que sejam capazes de receber descrições comportamentais a níveis de sistema e cheguem até um sistema funcionando de forma automática. Os níveis de abstração são discutidos na seção seguinte.

5.1.3 Níveis de abstração

Em um projeto de ASIC as ferramentas de CAD desempenham um papel fundamental no fluxo de projeto. Um fluxo de projeto simplificado pode ser visto na Figura 5.2. Ferramentas de CAD são usadas em quase todos os pontos do fluxo e muitas interações são necessárias entre eles. Durante o projeto, são utilizados vários níveis de abstração, de forma a permitir o gerenciamento da complexidade mantendo os parâmetros relevantes e suprimindo os detalhes desnecessários. A abstração dos detalhes permite que

o projetista se concentre em uma representação de alto nível simplificada do projeto. O fluxo de projeto vai a cada passo aprofundando-se passando para níveis mais baixos de abstração. A conversão de uma descrição em um nível de abstração para outro inferior é denominado síntese. Os níveis de abstração encontrados em projetos são nível de sistema, nível comportamental, nível de transferência entre registros, nível lógico e nível de layout. O **nível de sistema** se concentra no projeto como um todo. A representação é um conjunto de especificações e definições das funções do sistema. O **nível comportamental** se concentra na descrição das operações básicas utilizadas para implementar as funções do sistema. Um modelo comportamental é usado para avaliar diversas opções de projeto. O **nível funcional ou de transferência entre registros** consiste de um modelo mais detalhado do sistema levando em conta a temporização. É formado com registradores, memória e funções lógicas e aritméticas. As ferramentas atuais são capazes de síntese automática a partir de descrições funcionais. No **nível lógico** o projeto é representado como uma coleção de circuitos como flip-flops, portas lógicas, etc. Ferramentas automáticas podem sintetizar o nível lógico diretamente para um layout. O **nível de circuito** descreve como os elementos de circuito como transistores, resistores, capacitores, estão conectados. É utilizado para analisar o comportamento elétrico do sistema. O nível mais baixo, **nível de layout**, é primariamente geométrico, com primitivas utilizadas para a fabricação dos circuitos integrados.

5.1.4 Síntese

A **síntese comportamental** ainda é objeto de pesquisa, mas alguns resultados são obtidos para uma faixa estreita de problemas [DER97]. Por outro lado, a **síntese a partir do nível de registros** já está bem consolidada e permite bons resultados com aumento da produtividade. Uma entrada de descrição neste nível geralmente é realizada em linguagem de descrição de hardware. Utiliza-se um subconjunto da linguagem com construções de fluxo de controle, condições e laços, operadores aritméticos e lógicos e alocação de registros. O resultado é um conjunto de registros e lógica combinatória. A **síntese de layout** atualmente é completamente automática. Nela a ferramenta realiza o posicionamento dos elementos e o roteamento das interconexões.

5.1.5 Fluxo de projeto

O processo de desenvolvimento utilizando FPGAs é muito similar a produção de um ASIC. Apenas as etapas finais, quando se realiza o mapeamento da descrição na tecno-

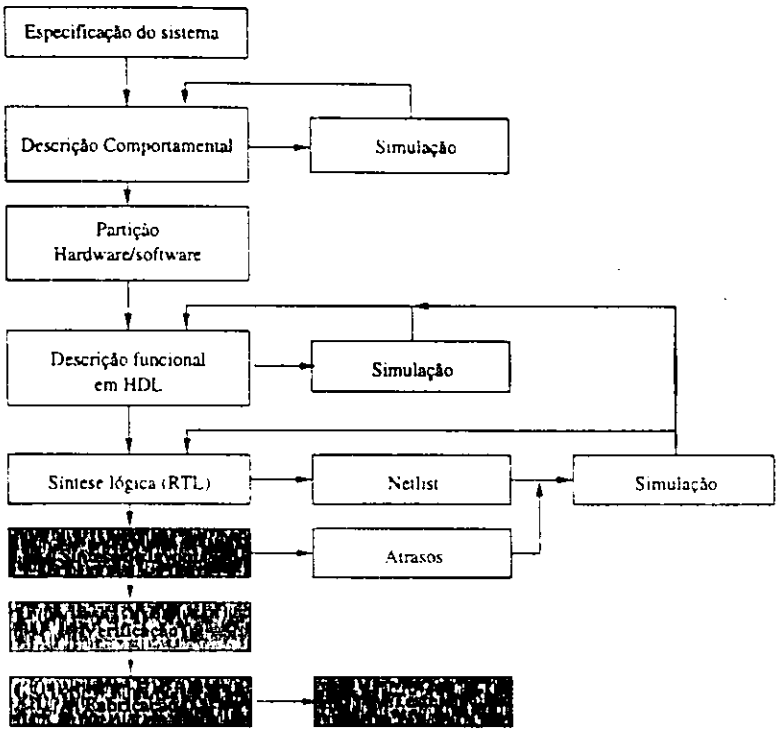


Figura 5.2: Fluxo de projeto de ASIC ou FPGA simplificado

logia é diferente. A Figura 5.2 apresenta um fluxo de projetos de ASICs simplificado. O projeto com FPGAs diferencia apenas nas áreas sombreadas, de são substituidas pelas etapas vista na Figura 5.3. As principais etapas são comentadas abaixo:

Especificação do sistema - Nesta fase estuda-se a performance global do sistema, o custo, a testabilidade e a interação de sub-sistemas, abstraindo-se completamente como o sistema será implementado, focalizando-se apenas na sua funcionalidade.

Codificação em linguagem de alto nível - realiza-se um modelo comportamental do sistema para avaliação, adaptação e validação de algoritmos e para explorar opções de projeto como conjunto de instruções, tamanho de palavra, interfaces, organização de memórias, entre outras. Como o comportamento do sistema é estudado com descrições comportamentais, muitas opções de projeto podem ser testadas rapidamente, através de simulações.

Partição hardware/software - Em sistemas mais complexos que utilizem a metodologia de co-projeto de hardware/software, é necessário realizar a decisão de que funções se adequam melhor a realização em hardware ou devem ser programados em software, segundo alguns critérios de otimização, como custo ou consumo, e sujeitos às restricões de projeto, como velocidade de operação.

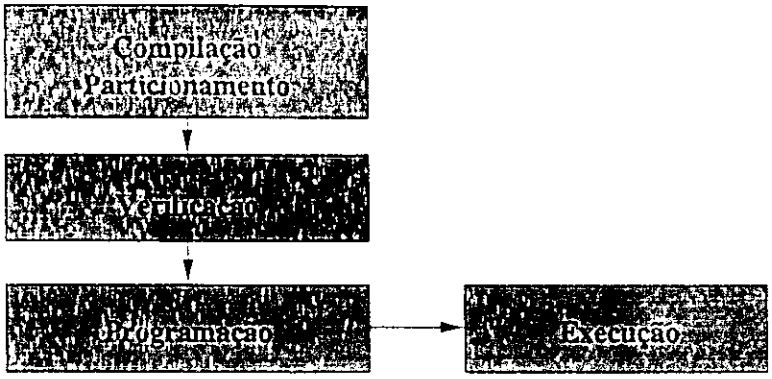


Figura 5.3: Modificação no fluxo de projeto da Figura 5.2 para FPGA

Descrição funcional - Definidas as partes que deverão ser integradas, realiza-se um modelo do sistema em linguagem de descrição de hardware. Neste modelo, o algoritmo é implementado com operações elementares entre os pulsos de relógio. Através de simulações verifica-se o comportamento do modelo funcional comparando-o com o modelo comportamental para assegurar que está correto. O modelo funcional já possui conectividade associada e pode ser usado para verificar que os modelos subsequentes tenham o mesmo conjunto de sinais.

Síntese lógica - A síntese a nível de registro, realizada a partir de uma descrição funcional, produz uma lista de elementos lógicos e interconexões (*netlist*). É um processo semi-automático que nem sempre produz resultados satisfatórios. Diversos parâmetros devem ser configurados para dirigir o processo. Dentre eles, os mais importantes são a tecnologia na qual o circuito será fabricado e as imposições de desempenho. O *netlist* obtido é usado para análise de velocidade, consumo e área. O netlist é simulado usando o mesmo conjunto de vetores usado nas simulações anteriores e os resultados são comparados.

Neste ponto são incluídas as facilidades de teste, normalmente de forma automática pela ferramenta de síntese.

Síntese de layout - Com o uso de ferramentas automáticas e interativas, são realizados os processos de posicionamento e roteamento dos elementos de layout. De posse de um layout, utiliza-se ferramentas de extração de características, que geram netlists acrescidos de elementos de circuito, que descrevem atrasos adicionais devido às interconexões. Novas simulações e alterações devem ser realizadas para assegurar que o circuito ainda atende às especificações.

Fabricação - Envia-se à fábrica um um banco de dados compilado com a descrição do layout e pelo conjunto de vetores de teste. Após a fabricação são realizados

testes de falha de fabricação usando estes vetores.

Teste - Após a chegada do circuito são realizados testes funcionais.

As etapas sombreadas no fluxo da Figura 5.2 são substituídas pelas mostradas na Figura 5.3 quando se realiza um mapeamento em FPGAs.

As etapas são as seguintes:

Compilação - A compilação na realidade engloba vários passos realizados automaticamente pela ferramenta. São eles síntese logica, o particionamento em varios dispositivos, o *floorplanning* e finalmente o posicionamento e roteamento. Estas etapas podem ser realizadas automaticamente ou pode ser dirigidas pelo projetista, levando normalmente a melhores resultados. O resultado da compilação é um arquivo de bitstream, que possui os bits de configuração para cada elemento programável dentro do FPGA.

Programação - Enquanto que na fabricação de um CI leva semanas ou meses para se receber o produto final, a configuração do FPGA é realizada em segundos. Após a programação o dispositivo está pronto para ser utilizado.

Execução - O teste final de funcionamento normalmente é realizado dentro do próprio sistema.

5.1.6 Linguagens de descrição de hardware

O projeto de hardware com linguagens de descrição de hardware está se tornando similar aos métodos utilizados para desenvolver software convencional. É difícil saber até onde esta tendência levará. Uma vez que os projetistas trabalham mais efetivamente a níveis de abstração mais altos (onde esquemáticos e diagramas de estado têm menos valor), eles provavelmente tirarão vantagem das representações mais baseadas em texto para acelerar a entrada e facilitar o uso. Por outro lado, as interfaces gráficas estão continuamente melhorando e logo elas permitirão que os projetistas representem idéias muito complexas com objetos gráficos relativamente simples.

As linguagens de descrição mais comuns são VHDL [IEE88] e Verilog [INC94]. Com elas é possível representar o hardware em alto nível, nível funcional, nível lógico e de forma insipiente nível de circuito, como também de forma heterogênea, com módulos em níveis diferentes. Estas HDL permitem descrever fluxo de controle (if-then-else e case), iteração, hierarquia, comprimento de palavras, vetores e campos de bits, operações sequenciais e paralelas, alocação e especificação de registros e operações lógicas e aritméticas.

As descrições em HDL podem ser simuladas, através da construção de *test-benches* definidos diretamente na linguagem, para a geração de estímulos e a obtenção de resultados, que podem ser visualizados de forma textual, gráfica, ou processados de alguma maneira para melhorar a compreensão. O mesmo *test-bench* pode ser usado posteriormente para verificação do resultado da síntese lógica.

Assim, descreve-se um sistema em HDL para capturar a especificação comportamental de alto nível, fornecer descrições simuláveis de circuito e sistema, capturar vetores de verificação e permitir sintése lógica automática a partir da descrição.

Os benefícios obtidos são o aumento a produtividade do projetista e da taxa de sucesso na primeira fabricação e a captura do projeto em um formato que permite o remapeamento de projetos em cima de tecnologias mais densas, mais econômicas ou mais rápidas. Também é possível realizar projetos maiores mais eficientemente. As desvantagens estão nos custos das ferramentas de CAD.

5.2 Metodologia de desenvolvimento com FPGAs

O projeto de sistemas usando FPGAs segue os mesmos princípios, utiliza ferramentas similares e as linguagens utilizados em projetos de ASIC. No entanto, devido a acessibilidade e facilidade de reprogramação de FPGAs em contraste com a necessidade de simulações e verificações de projetos ASIC, as condições são diferentes. Isto leva a diferenças na metodologia de trabalho. A necessidade de simulações é reduzida em FPGAs devido a possibilidade de verificação de funcionamento em real.

A metodologia utilizada para projetos de sistemas utilizando FPGAs é praticamente o mesmo mostrado no fluxo de projeto da Figura 5.2. As principais diferenças residem no fato de que os FPGAs são circuitos prontos, testados contra defeitos de fabricação e podem ser programados em segundos. Assim, para o ASIC, muitas iterações nos ciclos iniciais devem ser realizadas, com diversas simulações, para garantir que o primeiro circuito fabricado funcione. Já para os FPGAs, normalmente chega-se até o circuito programado para realizar os testes funcionais.

Outra diferença importante existe sobre a formação dos profissionais que trabalham com ASIC e FPGAs. As exigências sobre um projetista de ASIC exigem um especialista de hardware, com formação em microeletrônica. Em um projeto com FPGA as exigências sobre o projetista são menores, recaindo sobre as ferramentas, que precisam realizar mais etapas automaticamente, de forma eficiente. Com a melhoria das ferramentas, pretende-se que profissionais da área de informática, sem conhecimento profundo de hardware, possam realizar projetos de computação reconfigurável de forma

similar a programação.

As ferramentas utilizadas para projeto de ASICs permitem o projeto de FPGAs com a instalação de bibliotecas e design kits específicos. Desta forma, toda a metodologia desenvolvida e trabalhos anteriores podem ser reaproveitados. A implementação em ASIC ou FPGA no processo de síntese é definida apenas no mapeamento para a tecnologia. Assim, os circuitos podem ser mapeados em FPGA para prototipagem de ASICs ou também podem ser utilizados para ensino de microeletrônica.

Durante os testes realizados com o sistema desenvolvido foram utilizadas várias ferramentas de CAD disponíveis na UFPB. A linguagem de descrição utilizada é Verilog, que entre outros motivos, é melhor gerenciada pelas ferramentas existentes. Para simulação de código Verilog foi utilizado o Verilog-XL do Framework Opus da Cadence [INC94], o VeriWell, da Wellspring e o Max+PlusII da Altera. Para síntese Verilog foram utilizados o Synergy da Cadence, o FPGA Express da Synopsys e também o Max+PlusII. Após a síntese, os netlists obtidos devem ser submetidos à ferramenta completa Max+PlusII para a geração do bitstream necessário a programação dos dispositivos. A programação é realizada pelo software desenvolvido para este sistema, PROG_ALT.

Após uma síntese (compilação) bem sucedida, o Max+PlusII produz o *bitstream* de programação codificado em arquivos de diversos formatos. Para o sistema utilizamos o formato .rbf (*raw binary file*). Em um sistema com múltiplos FPGAs deve ser gerado um *bitstream* formado pela concatenação de vários *bitstreams*. Isto é realizado automaticamente pela ferramenta quando o projeto é multiparte, e manualmente quando os circuitos são descritos independentemente (no menu de programação, concatenação de bitstream). O arquivo final deve ser aplicado ao programa PROG_ALT, que realiza a programação e verificação efetiva nos FPGAs do sistema.

O Max+PlusII é a principal ferramenta no desenvolvimento com FPGAs da Altera. Ela é uma solução completa com entrada do projeto, compilação, verificação e programação dos dispositivos. Além disso, esta ferramenta possui a possibilidade de interface com diversos outros sistemas CAD. A captura de dados pode ser via entrada esquemática, com linguagens de descrição de hardware AHDL, VHDL e Verilog e entrada com forma de onda. O sistema pode ser definido de forma hierárquica e híbrida. Durante a compilação, a ferramenta realiza a verificação de regras de projeto, síntese lógica, particionamento automático em vários componentes e compilação com restrições de temporização. A interface do Max+Plus-II com outras ferramentas é realizada através de arquivos no formato padrão EDIF [] [] e netlists em Verilog e arquivos de atraso SDF.

Para o desenvolvimento com o SPRIT, é necessário definir como os pinos dos FPGAs serão utilizados. Na ferramenta Max+Plus-II a configuração dos pinos pode ser armazenada em arquivo para reutilização. A forma mais fácil de reutilizar um arquivo de configuração é abrir um projeto já existente e salvar com outro nome (*save as...*). O arquivo de descrição utilizado nos testes pode ser visto no apêndice XXX.

5.3 Conclusão

A elevada taxa de crescimento tecnológico na área de microeletrônica exige um avanço nas ferramentas de desenvolvimento, na metodologia de trabalho e na formação dos profissionais. A tendência atual é o uso de linguagens de descrição de hardware e entrada a níveis de abstração mais altos e ferramentas com melhores interfaces com os usuários.

Projetos com FPGAs têm as necessidades similares e seguem a mesma metodologia de projetos em ASIC. Assim, sistemas de prototipagem com FPGAs podem ser utilizados no ensino das técnicas de projeto de microeletrônica.

Espera-se que o avanço nas ferramentas de projeto permitam que sistemas de computação reconfigurável sejam programados por profissionais sem profundo conhecimento de microeletrônica.

Capítulo 6

Testes e resultados

Foram realizados testes para verificação de funcionamento de cada sub-módulo existente no sistema desenvolvido. Em seguida foi analisado a implementação de um circuito complexo descrito em Verilog. São mostrados a velocidade e a ocupação de recursos lógicos obtidos.

6.1 Testes dos sub-módulos

Os sub-módulos do sistema desenvolvido foram testados individualmente. O primeiro sub-módulo testado foi a programação do FPGA A1 (ver esquemático em anexo).

6.1.1 Circuito de programação

Inicialmente foi testada a interface paralela utilizada para a programação dos dispositivos. Este circuito é formado pela PPI 8255, decodificador e drivers para os LEDs indicadores. O circuito pode ser testado através do LED de teste, colocado para este fim.

Em seguida foi realizado o teste de programação dos dispositivos com a criação de um circuito mínimo, formado apenas por uma porta lógica e a programação utilizando o programa desenvolvido PROG_ALT. Este teste serviu para depurar tanto o circuito quanto o programa PROG_ALT. O dispositivo verifica o bitstream com uma soma de verificação e indica quando a programação é realizada com sucesso através dos sinais CONF_DONE e nSTATUS, apresentados nos LEDs.

Antes da programação todos os pinos dos dispositivos ficam em alta impedância e não provocam nenhuma influência sobre o restante do circuito nem no barramento ISA. Após a programação, os dispositivos passam por uma etapa de inicialização e os pinos assumem a direção programada. No entanto, o software Max+Plus II assume

```
module conta1(saida, clk);
       output [9:0] saida;
       input clk;

   lpm_counter contador(.saida(q),.clk(clock))
      defparam contador.lpm_width = 10;
endmodule
```

Figura 6.1: Descrição em Verilog de um contador módulo 1024.

por default que os pinos não utilizados na descrição e não reservados são saídas fixas em nível lógico 0. Isto ocorre para que os pinos não utilizados não fiquem flutuando e insiram ruído no dispositivo. Devido a isto, os pinos não utilizados em determinado projeto mas que estejam conectados aos circuitos auxiliares, como o barramento ISA e memórias devem ser reservados. Além disto, se os dois dispositivos forem soldados na placa, ambos devem ser programados. Isto é obtido com a geração de um bitstream formado pela concatenação de dois outros bitstreams. Este processo é realizada pelo usuário na ferramenta Max+Plus II.

6.1.2 Teste funcional - Contador módulo 1024

O teste realizado em seguida verificou o funcionamento dos dispositivos FPGA e do gerador de relógio. O circuito foi programado através da descrição vista na Figura 6.1, escrita em Verilog, um contador binário módulo 1024. A definição da direção dos pinos é realizada diretamente na ferramenta e armazenada em um arquivo do tipo ACF. Este arquivo pode ser reutilizado em outros projetos.

O pino de entrada de relógio foi mapeado para o pino dedicado pelo dispositivo para este fim. O pino de saída foi mapeado para um dos pinos dos conectores de E/S. O resultado, uma forma de onda quadrada de frequência aproximada de 1KHz (20 MHz/1024), pôde ser visualizada no osciloscópio.

6.1.3 Interface ISA

O próximo sub-circuito testado foi a interface com o barramento ISA. Este sub-circuito é formado pela decodificação de endereços, buffers do barramento de dados e sinais de seleção. Parte do controle do barramento ISA é realizado internamente ao FPGA e a correta utilização do barramento depende da programação dos dispositivos. Como

todos os sinais do barramento ISA estão conectados aos dois dispositivos, os sinais de controle podem ser gerenciados por qualquer um destes.

O barramento ISA possui acessos de 8 bits de 16 bits. O tipo de acesso é diferenciado através dos sinais SBHE e M16. Quando estes sinais não são utilizados (permanecem em nível lógico 1), assume-se um acesso de 8 bits. Os sinais MEMR (PCRD) e MEMW (PCWR) indicam ciclo de leitura ou de escrita.

O primeiro teste realizado compreendeu acesso de leitura de 8 bits. O circuito implementado apresenta um valor predefinido (AAH em hexadecimal) quando o microcomputador executa uma leitura em qualquer endereço dentro do segmento escolhido nos estrapes.

Na descrição temos buffers com capacidade de alta impedância, cujas entradas estão ligados a níveis lógicos constantes e as saídas ligadas ao barramento de dados ISA. A habilitação dos buffers ocorre quando há requisição de leitura (PCRD = 0) e habilitação pelo circuito decodificador de endereços (ENF1 = 0).

Durante este teste foi constatado um erro no layout da placa. Devido a uma inconsistência na nomenclatura, os sinais PCRD e PCWR não possuíam conexão entre o barramento e os dispositivos FPGA.

O próximo passo realizado foi o teste de acesso de 16 bits. A descrição utilizada pode ser vista na Figura 6.2.

O sinal SBHE sinaliza uma requisição de transferência de 16 bits. O sinal M16 quando ativo indica ao barramento que o circuito é capaz de realizar transferências de 16 bits.

A interface ISA funcionou a contento, tanto para leitura quanto para escrita, com 8 e 16 bits.

6.1.4 Memória estática

O teste de acesso a memória estática foi realizado com um circuito formado apenas por interconexões entre os sinais do barramento ISA e os pinos ligados a memória. Para o barramento de dados foram utilizado buffers bidirecionais de três estados com habilitação e direção definidos pelos sinais de controle do barramento ISA.

6.2 Circuitos mapeados

Após os teste funcionais, alguns circuitos desenvolvidos anteriormente por outros alunos [Vil97] foram sintetizados para dispositivos Altera 10K40. As descrições originais escritas em Verilog foram alteradas para acomodar as pequenas diferenças existentes

```
module isa16(clk, PCA, PCRD, PCWR, SBHE, ENF1, PCD, M16, PA, J2, J4);
input clk, PCRD, PCWR, SBHE, ENF1;
input [14:0] PCA;
output M16;
inout [15:0] PCD;
reg [15:0]mem;
tri [15:0]PCD;
wire [9:0] q;

lpm_counter contador (.clock(clk), .q(q));
defparam contador.lpm_width = 10;
defparam contador.lpm_modulus = 1024;
wire ena = (!PCWR) && (!ENF1) && (PCA==0);
wire M16 = SBHE | ENF1;

always @(posedge clk)begin
if (ena)
      mem <= PCD;
end
lpm_bustri tribus (.data(mem), .enabledt((!PCRD) && (!ENF1)), .tridata(PCD));
defparam tribus.lpm_width = 16;
endmodule
```

Figura 6.2: Acesso de 16 bits ao barramento ISA.

nos formatos aceitos nos sintetizadores Synergy e Max+plus-II. O Max+plus-II possui um mecanismo de inferência de latches e registros mais simples que o Synergy e as descrições são mais sucintas.

Os resultados destes mapeamentos puderam ser observados no capítulo 2 quando foram comparadas as implementações de algoritmos em DSP, ASIC e FPGA. A tabela 6.1 resume estes resultados.

Podemos observar que o circuito de gradiente Spline pode operar com taxas de pixels superiores ao mínimo necessário para execução em tempo real e a taxa de ocupação foi pequena. Trata-se de um algoritmo que se adequa facilmente a uma implementação em FPGA.

O circuito de extração de vértices ocupou 97% dos recursos de um FPGA e apresen-

Algoritmo	Frequência Máx (MHz)	Células Lógicas	Recursos(%)
Vértice	4.4	2236	97
Spline	22.7	415	18

Tabela 6.1: Resultados do mapeamento em FPGA

tou baixa performance, não atingindo a velocidade de operação em tempo real. Este circuito é um exemplo de descrições pouco adequadas ao mapeamento em FPGAs. O circuito é quase que completamente combinacional, com barramentos com muitos sinais (centenas) que precisam ser roteados. A velocidade deste circuito pode ser aumentada consideravelmente através do uso da técnica de pipeline. Como os FPGAs possuem registradores em cada célula básica reconfigurável, o uso de pipeline é altamente encorajado como forma de melhoria no desempenho porque normalmente não aumenta a taxa de ocupação. Porém, a cada nível de pipeline acrescentado, aumenta em um ciclo de relógio na latência do circuito. A operação a uma taxa de pixels de 8 MHz, que corresponde a 30 quadros/s de 512×512 pixels, pode ser obtida acrescentando-se apenas um nível de pipeline.

A arquitetura do circuito integrado para extração de vértices pode ser visto na figura 2.8. Podemos observar a grande quantidade de barramentos com grande número de fios. Neste circuito apenas o módulo Gera_jan8 é sequencial. Todos os demais circuitos são combinacionais, o que permite a inclusão de níveis de pipeline com facilidade.

Como explicado no capítulo 2, a extração de vértice é um passo de pré-processamento do modelamento poligonal. Para este modelamento é necessário uma lista ordenada de vértices. Para a obtenção desta lista a partir de uma imagem, são realizadas várias etapas. Realiza-se a detecção de bordas através do gradiente spline e esta borda é afinada através das técnicas de contorno I e contorno II. O resultado é aplicado ao circuito de extração de vértices, obtendo-se uma lista não ordenada. Todas estas etapas foram implementadas anteriormente em Verilog. Os passos de afinamento de borda caminho I e II não foram implementados em FPGA e são deixados como trabalhos futuros.

Os circuitos citados acima utilizam uma interface de entrada de pixels simplificada, denominada Gera_Jan. Esta interface recebe um conjunto de pixels correspondente a uma coluna de n linhas, onde n é o tamanho da janela, e gera uma janela cujo pixel inferior direito é o último pixel recebido. Além desta janela, dependendo da operação, cópias rotacionadas e em espelho podem ser geradas. Estas cópias são geradas apenas com fiações. A taxa com que o conjunto de pixels é apresentado é a mesma do sinal de vídeo. Para tal é utilizado uma memória organizada como registrador de deslocamento.

Este circuito auxiliar não é integrado e deve ser fornecido. Para a redução do número de pinos nos circuitos integrados. esta entrada é multiplexada no tempo usando um relógio auxiliar cinco vezes maior que o relógio de pixel. Com este esquema, a taxa de transferência é cinco vezes maior do que a necessária e cada pixel é apresentado ao circuito integrado n vezes, onde n é o tamanho da janela.

A implementação real deste circuito requer a utilização de memórias RAM convencionais, contadores e multiplexadores, haja visto que não há registradores de deslocamento com as dimensões necessárias. Este circuito pode ser implementado diretamente no sistema desenvolvido, utilizando as memórias SRAM rápidas existentes. Para cada etapa de processamento implementada é necessário uma cópia deste circuito, porque ele permite o encadeamento das operações.

6.3 Módulos de Entrada/Saída

Para a obtenção de uma sequencia de imagens em tempo real, é necessário uma interface que receba sinais de vídeo de forma analógica e apresente-os digitalmente ao circuito. Duas interfaces estão sendo desenvolvidas. A primeira interface utiliza um decodificador (SAA7111A) de sinal de vídeo padrão em formato CVBS ou Y/C para diversos sistemas (NTSC, PAL e SECAM). A saída digital é apresentada em diversos formatos programáveis. Além dos dados de vídeo, sinais de sincronismo e de relógio (travado em linha) são obtidos. A segunda interface utilizará uma câmara de vídeo de baixo custo utilizada em microcomputadores. Esta câmara apresenta um dispositivo CCD e circuito de alimentação e drivers. Será realizada um acesso direto ao dispositivo CCD. Desta forma, a geração de relógio e sinais de sincronismo será realizado pelo sistema. Os dados de vídeo serão digitalizados por um A/D *flash* (TDA8708).

Os resultados em forma de vídeo poderão ser visualizados no monitor do microcomputador hospedeiro ou convertidos para os formatos padrão. Para visualização no monitor os dados podem trafegar pelo barramento do microcomputador ou enviados diretamente à placa de vídeo. A primeira opção é viável apenas quando algum processamento deve ser realizado pelo processador e a quantidade de dados é pequena, como uma lista de pontos. A segunda opção utiliza o conector padrão VMI existente em placas de vídeo. Este padrão [Jac97b] fornece meios de comunicação entre diversos dispositivos de geração e apresentação de sinais de vídeo digitais e de multimídia. A placa de vídeo apresenta os dados de vídeo obtidos no conector em uma janela diretamente no monitor em tempo real.

A geração de sinais de vídeo em formato padrão será realizada através de uma

circuito codificador de vídeo (SAA7199) que recebe dados digitais e gere um sinal de vídeo composto CVBS ou Y/C para apresentação em TV. Um circuito deste tipo está sendo implementado. O circuito será capaz de apresentar vídeo em diversos formatos (NTSC, PAL e SECAM).

Estes circuitos auxiliares permitirão a realização de diversos projetos práticos utilizando o sistema desenvolvido.

6.4 Conclusões

Os resultados obtidos com os testes foram satisfatórios. Através do mapeamento sobre o sistema desenvolvido, verificou-se que ele suporta implementações de algoritmos de processamento de imagens em tempo real de baixo nível, como o circuito de extração de vértices. Além disso, vários processamentos mais simples podem ser implementados em cadeia no sistema.

Devido a possibilidade de utilização de FPGAs de 20 mil a 70 mil portas equivalentes, o sistema pode implementar de pequenos a relativamente complexos algoritmos de processamentos de imagens.

Com a conclusão dos módulos de entrada e saída, implementações práticas mais significativas poderão ser realizadas.

Capítulo 7

Conclusões e trabalhos futuros

Este trabalho objetivou a realização de um sistema capaz de realizar operações de processamento de imagens de baixo nível em tempo real. As exigências temporais determinam a utilização de hardware específico, mas por outro lado a ampla gama de aplicações requer um sistema flexível, capaz de ter seu comportamento reconfigurado para a necessidade de cada momento. Uma solução adequada é o uso de lógica reconfigurável.

O resultado final é um sistema onde se espera poder executar operações de processamento digital de sinais com um desempenho uma ordem de magnitude superior a sistemas com DSPs. A partir dos resultados dos mapeamentos realizados, diversas conclusões podem ser formadas com relação a viabilidade do uso de FPGAs para processamento de imagens. Estes resultados revelam também sugestões para possíveis trabalhos futuros nesta área. Estas conclusões e trabalhos futuros são discutidos em seguida.

7.1 Conclusões da dissertação

7.1.1 Estado atual do sistema

O sistema proposto foi construído e depurado. Alguns algoritmos de processamento de imagens desenvolvidos na instituição foram mapeados e os resultados obtidos demonstram a possibilidade de implementação de operações de relativa complexidade.

Atualmente o sistema está sendo utilizado como estrutura básica de desenvolvimento para realização de diversos algoritmos, entre eles o cálculo da mediana para remoção de ruído em imagens multi-espectrais de satélite.

7.1.2 FPGA como elemento processador de DSP

Os resultados do estudo de alguns algoritmos de processamento de imagens realizado neste trabalho indicam que dispositivos FPGA são adequados para implementação deste tipo de operação. A arquitetura sistólica, ou seja, com elementos dispostos em matriz, adequada a um fluxo de dados altamente regular, se beneficia com algoritmos localizados e em fluxo de dados. Mais especificamente, os processamentos operando com janelas como o gradiente Spline e em blocos como a DCT são mapeados com sucesso. Circuitos com lógica combinacional de muitos níveis e sinais não são adequados à implementação em FPGAs devido a escassez de recursos para roteamento. Estes circuitos podem ser melhor mapeados com alterações na descrição para execução em etapas, ou seja, a inclusão de níveis de *pipeline*. Com isso se consegue aumento de desempenho sem perda significativa de área ocupada, haja visto que, os FPGAs são ricos em registros. Um exemplo deste tipo de operação é a detecção de vértices, visto nos capítulos 2 e 6. Um desempenho adequado para processamento em tempo real é obtido com um circuito cuidadosamente projetado, levando em consideração as características específicas da arquitetura dos FPGAs utilizados.

7.1.3 Processo de desenvolvimento

A utilização mais ampla de FPGAs nas áreas de processamento de alto desempenho e computação reconfigurável depende de um maior desenvolvimento dos dispositivos, das ferramentas e da metodologia utilizada, de forma a reduzir o ciclo de projeto associado. Esperam-se melhorias que permitam a utilização de 100% dos recursos dos dispositivos, com roteamento e particionamento em múltiplos dispositivos de forma completamente automática.

Com um melhor suporte a projeto em alto nível, os projetos poderão ser implementados em um nível mais próximo do algoritmo real. Parte deste objetivo está sendo conseguido com o uso de linguagens de descrição de hardware trabalhando a níveis de abstração mais altos e a inclusão de macros e módulos parametrizáveis.

7.1.4 Desenvolvimento tecnológico

Trata-se de um sistema dotado de um grande desenvolvimento tecnológico graças a aplicação de técnicas modernas como circuitos impresso de várias camadas e manipulação de encapsulamento SMD com centenas de pinos.

7.2 Trabalhos futuros

Citamos os seguintes trabalhos que podem ser desenvolvidos futuramente:

- Mapeamento dos circuitos de afinamento de bordas Caminho I e II
- Implementação de um sistema completo de reconhecimento de objetos baseado no modelamento poligonal. O referido sistema consiste de um co-projeto em hardware/software com as etapas de baixo nível, ou seja, extração de gradiente Spline, Caminhos I e II e detecção de vértices, realizadas em hardware no sistema de prototipagem proposto, e as etapas de ordenação de vértices e casamento de padrões realizada por software no processador hospedeiro.
- Verificação algorítmica em tempo real de métodos de compressão de imagens utilizando a DCT, wavelets e segmentação poligonal (baseada em modelos).
- Implementação dos circuitos de entrada e saída de vídeo.
- Implementação de circuito de geração de sinais para motores de indução trifásicos.

Bibliografia

[AA95] P. M. Athanas and A. L. Abbott. Real-time image processing on a custom computing platform. *IEEE Computer*, 28(2):16–24, February 1995.

[Alt96] Image processing in altera flex 10k devices. Technical report, Altera Corp., 1996.

[BH94] S. Bade and B. L. Hutchings. FPGA-based stochastic neural networks: Implementation. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 189–198, Napa, CA, April 1994.

[BP97] William F. Brinkman and Mark R. Pinto. The future of solid-state electronics. *Bell Labs Technical Journal*, 2(4), autumn 1997.

[CL] Chiu and Liu. Real-time parallel and fully pipelined 2-d dct lattice structures with application to hdtv systems. *Transactions on Circuits and Systems for Video Tech.*, 2.

[CME94] Chi-Jui Chou, Mohanakrishnan, and Joseph B. Evans. Fpga implementation of digital filters. In *Proceedings of the Fourth International Conference on Signal Processing Applications and Technology*, pages 80–88, 1994.

[CNH93] S.C. Chan, H.O. Ngai, and K.L. HO. A programmable image processing system using FPGAs. *International Journal of Electronics*, 75(4):725–730, October 1993.

[Cor95a] Cortez, P. C. e Carvalho, J. M. "A 2D efficient Algorithm for Polygonal Modeling". $10^{\underline{o}}$ *International Congress on Industrial and Applied Mathematics, Hamburg*, Julho 1995.

[Cor95b] Cortez, P. C. e Carvalho, J. M. "Look-ahead Boundary Tracking Segmentation of 2D Contours". *XI Congreso Chileno de Ingenieria Electrica, Punta Arenas-Chile*, Nov. 1995.

[Cor95c] Cortez, P. C. e Carvalho, J. M. "Modelagem Poligonal de Contornos 2d Usando a Transformada de Hough". 13º *Simpósio Brasileiro de Telecomunicações, Águas de Lindóia-SP*, Setembro 1995.

[Cor97] Atmel Corporation. *Configurable Logic Data Book.* march 1997.

[Cor98] Altera Corp. An-75 high-speed board designs. Technical report, january 1998.

[CSF77] W.-H Chen, C. H. Smith, and S. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE transactions on Communications*, COM-25, september 1977.

[DER97] Alfred E. Dunlop, William J. Evans, and Lawrence A. Rigge. Managing complexity in ic design - past, present and future. *Bell Labs Technical Journal*, 2(4), autumn 1997.

[EH94] J. G. Eldredge and B. L. Hutchings. RRANN: The run-time reconfiguration artificial neural network. In *Custom Integrated Circuits Conference*, pages 77-80, San Diego, CA, May 1994.

[EH96] J.G. Elredge and B.L. Huthings. Run-time reconfiguration: A method for enhancing the functional density of sram-based fpgas. *Jornal of VLSI Signal Processing*, 12, 1996.

[eMS96] T. Y. Tian e M. Shah. "Motion estimation and segmentation". *Machine Vision and Applications*, 9:32–42, 1996.

[GFGT95] M. Gheari, S. De Faria, I. N. Goh, and K. T. Tan. "Motion compensation for very low bitrate video". *Signal Processing: Image Communication*, 7:567–580, 1995.

[GN98] Paul Grahan and Brent Nelson. Fpga-based sonar processing. *ACM/SIGDA International Symposyum of Field Programmable Gate Arrays*, february 1998.

[GW77] R. C. Gonzalez and P. Wintz. *"Digital Image Processing"*. Addison-Wesley Publishing Company, 1977.

[IEE88] IEEE. *"Standard VHDL Language Reference Manual"*. IEEE Std 1076-1987, 1988.

[Inc92] Texas Instrument Inc. *TMS320C4X Usersś Guide.* 1992.

[INC94] CADENCE DESIGN SYSTEMS INC. *"Verilog-XL Reference Manual 2.0".* 1994.

[Jac97a] Keith Jack. Bt.656 video interface for ics. Technical report, Harris Semiconductor, June 1997.

[Jac97b] Keith Jack. Video module interface (vmi) for ics. Technical report, Harris Semiconductor, September 1997.

[LGea97] David M. Lewis, David R. Galloway, and et al. The transmoglifier-2: A 1 million gate rapid prototyping system. *FPGA '97, ACM Symposyum on FPGAs*, february 1997.

[Man95] R.T. Maniwa. Putting it together: Prototyping methods. *integrated system design*, september 1995.

[MCCea96] E. Melcher, J.M. Carvalho, P.C. Cortez, and et al. A vertex detection algorithm for vlsi implementation. In *Anais do SIBGRAPI XI*, Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pages 367–368, 1996. Published in Portuguese.

[MCea96] E. Melcher, J.M. Carvalho, and et al. A novel gradient operator suited for vlsi implmentation of 2d shape recognition. In *Anais do SBCCI 96*, March 1996. Published in Portuguese.

[MNdC+97] Elmar Melcher, Lírida Naviner, Joao Marques de Carvalho, Jean François Naviner, Ricardo A. S. Moreira, Yuri de Mello Villar, and Marcos R. Morais. "VLSI Implementation of Contour Extraction from Real Time Image Sequences". *VLSI*, 1997.

[Pag96] Ian Page. Towards a common framework for hardware and software. In *Anais do IX SBCCI*, pages 3–22, Recife - PE, março 1996.

[Pet95] R. J. Petersen. An assessment of the suitability of reconfigurable systems for digital signal processing. Master's thesis, Brigham Young University, 1995. p. 60.

[Ren96] Michael Rencher. A comparison of FPGA platforms through SAR/ATR algorithm implementation. Master's thesis, Department of Electrical and Computer Engineering, Brigham Young University, Provo, Utah, 1996.

[Ros97] Richard Ross. An FPGA implementation of ATR using embedded RAM for control. Master's thesis, Department of Electrical and Computer Engineering, Brigham Young University, Provo, Utah, 1997.

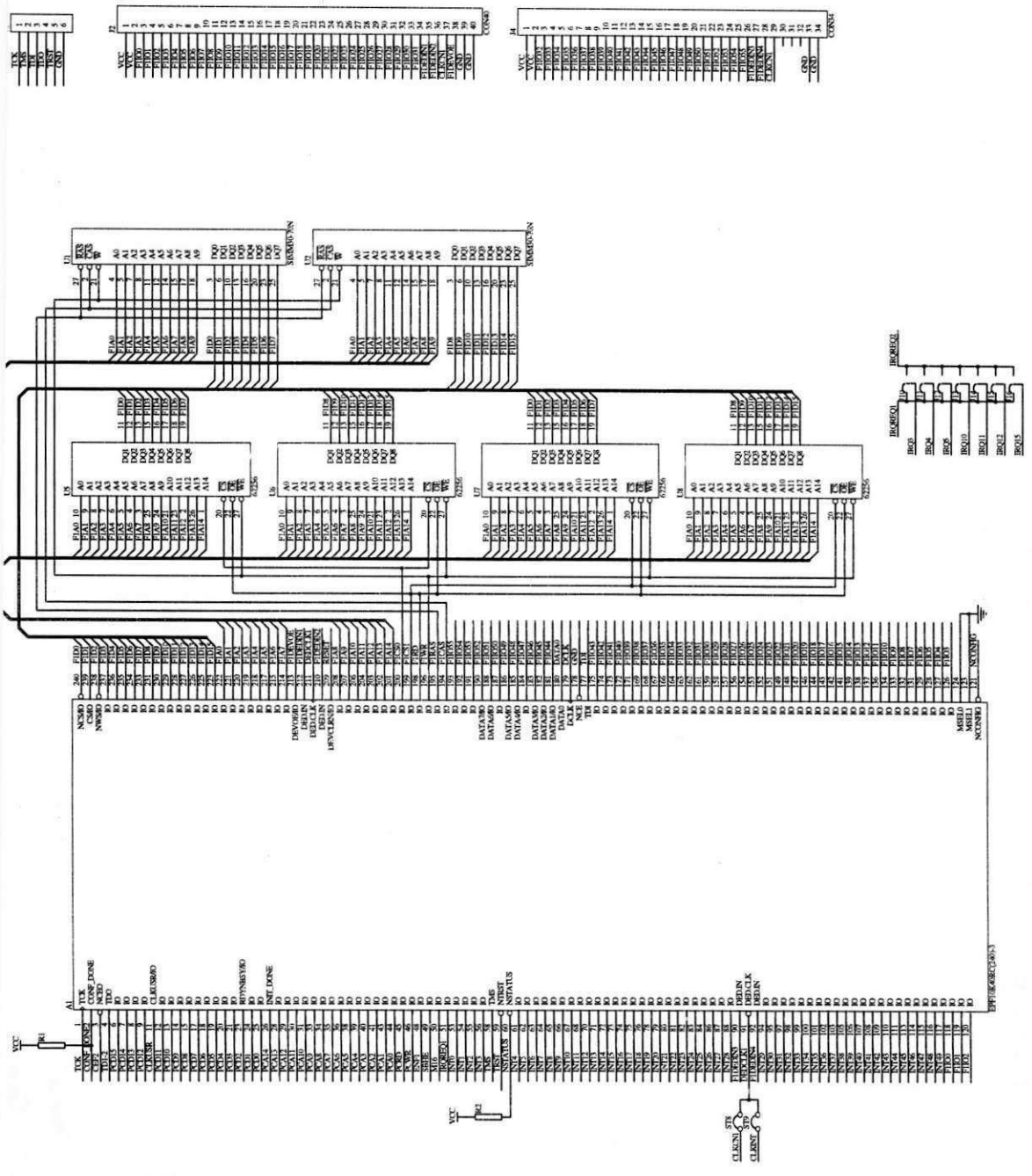
[Vil97] Yuri de Melo Vilar. Projeto de um asic para extração de vértices de imagens em tempo real. Master's thesis, UFPB, 1997.

[Xil94] Xilinx. *"The XC4000 Programmable Gate Array Data Book"*. San Jose, CA, USA, 1994.

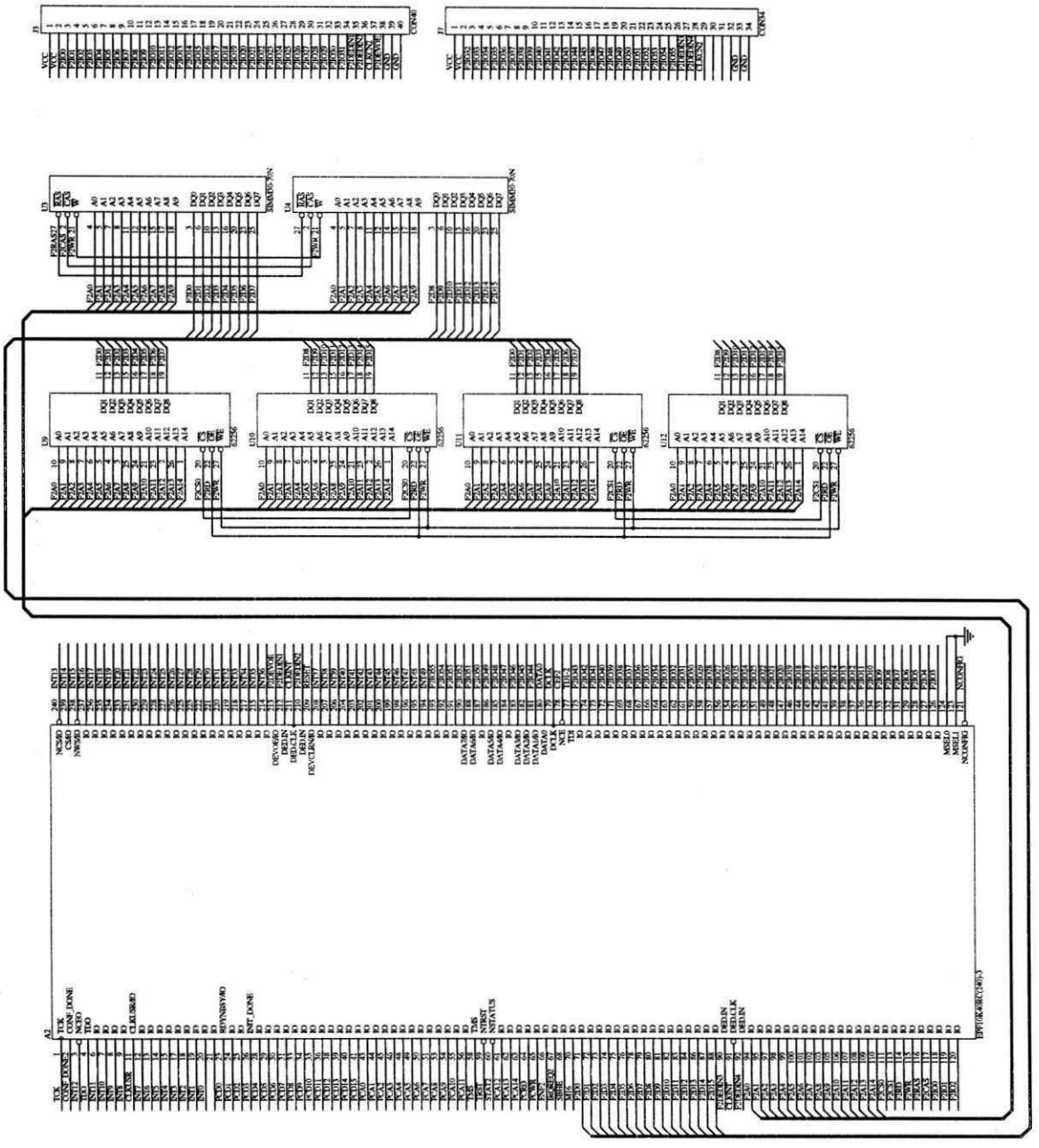
Apêndice A

Esquemas Elétricos

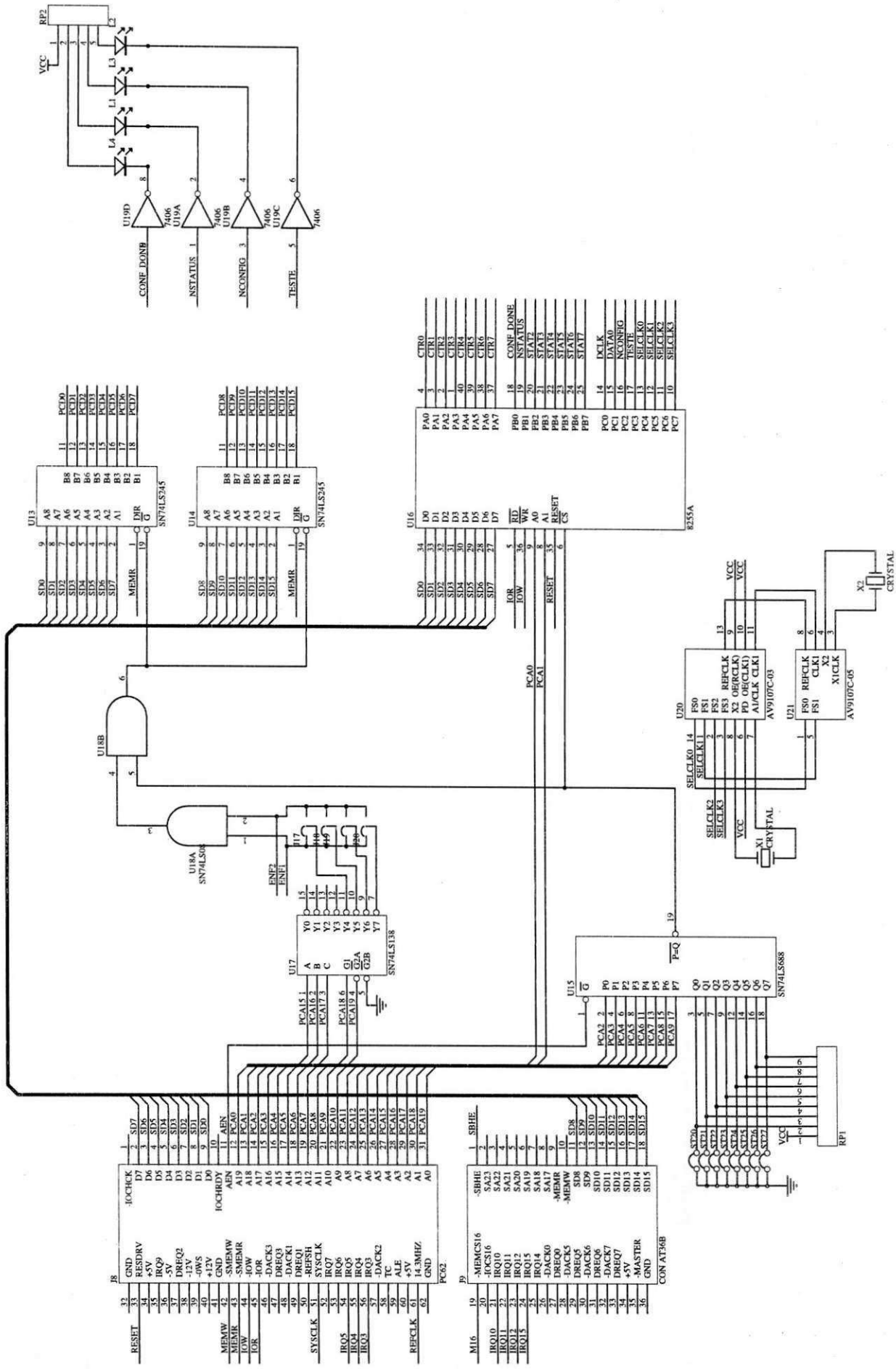
A.1 Esquema elétrico - página 1/3



A.2 Esquema elétrico - página 2/3



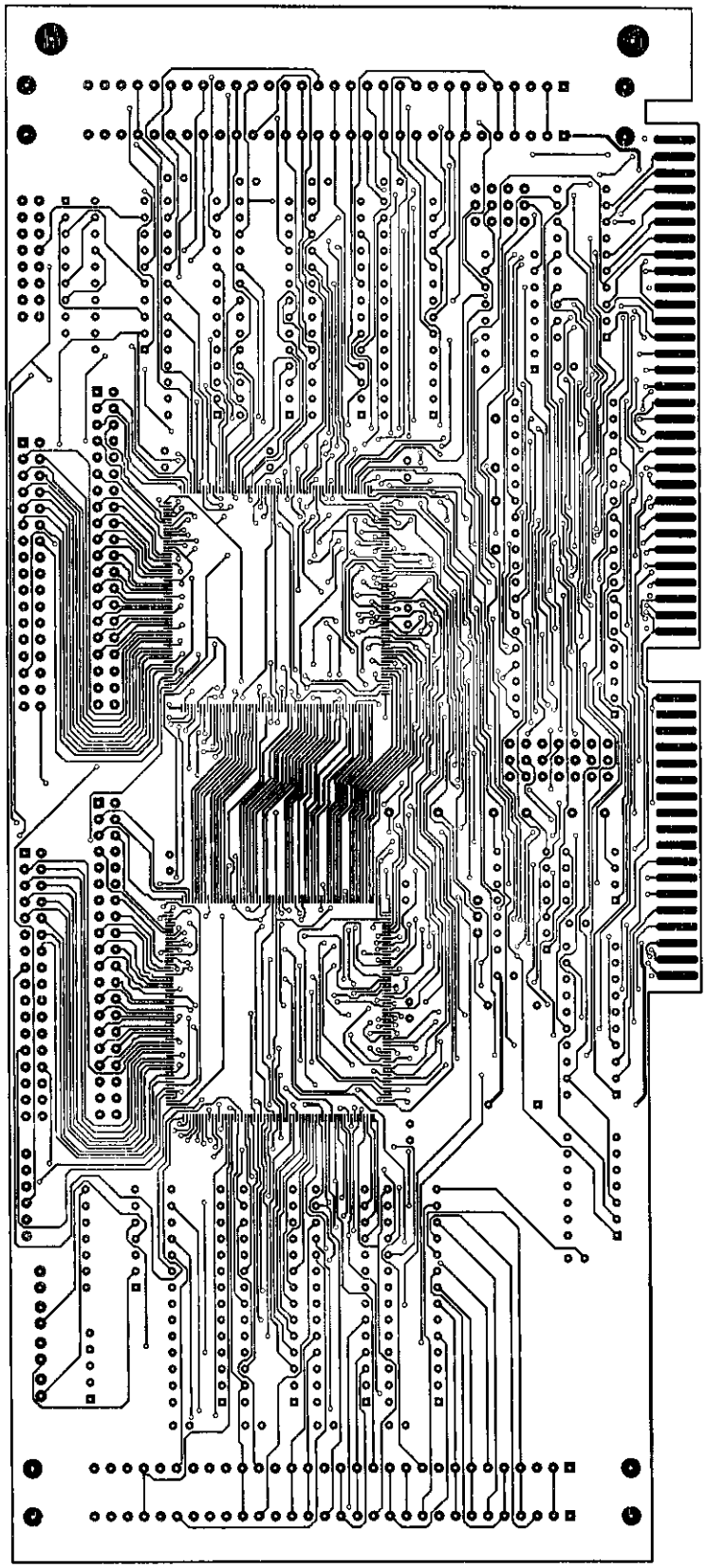
A.3 Esquema elétrico - página 3/3



Apêndice B

Layouts da placa

B.1 Lado dos componentes



Apêndice C

Programa Prog_Alt

```
/*+-------------------------------------------------------------------------+*/
/*|                              Includes                                   |*/
/*+-------------------------------------------------------------------------+*/

#include <stdio.h>
#include <dos.h>
#include <time.h>

/*+-------------------------------------------------------------------------+*/
/*|                            Definicoes                                   |*/
/*+-------------------------------------------------------------------------+*/

#define  BYTE   unsigned char
#define  WORD   unsigned int
#define  P_A    (base)
#define  P_B    (base+1)
#define  P_C    (base+2)
#define  CTL (base+3)
#define  outb    outportb
#define  in     inportb
#define  low(n) (outb(CTL, n<<1))
#define  high(n) (outb(CTL, (n<<1)+1))
#define  DCLK 0
#define  DATA0 1
#define  CONFIG 2
#define  TESTE  3
/* mascaras de bits */
#define  STATUS 2
#define  CONF_DONE 1

/*+-------------------------------------------------------------------------+*/
/*|                            Prototipos                                   |*/
/*+-------------------------------------------------------------------------+*/

void serial(BYTE  dado);

/*+-------------------------------------------------------------------------+*/
/*|                         Variaveis Globais                               |*/
/*+-------------------------------------------------------------------------+*/

WORD base;

/*+-------------------------------------------------------------------------+*/
/*|                         Rotina Principal                                |*/
/*+-------------------------------------------------------------------------+*/

void main(int argc, char *argv[])
{
  register BYTE dado;
  FILE *fp;
```

```
 unsigned long cont;
 time_t inicial;

 if (argc < 2) {
   printf("uso: prog_alt arquivo [porta]\n");
   exit(1);
 }
 if (argc == 2) base = 0xe4;
 if (argc >= 3) {
   base = sscanf (argv[2], "%3x", &base);
   if (!base) {
     printf("Porta errada\n");
     exit(2);
   }
 }

 inicial = time (NULL);
 outb(CTL, 0x92);  /* A entrada, B entrada, C saida */
 fp = fopen (argv[1], "rb");
 if (fp == NULL) {
   printf("Nao consigo abrir arquivo: %s\n", argv[1]);
   exit(3);
 }

 low(CONFIG);
 low(CONFIG);
 while(in(P_B) & STATUS != 0);  /* Espera status descer - sem timeout */
 high(CONFIG);
 high(CONFIG);
 while(in(P_B) & STATUS != 1);  /* Espera status subir - sem timeout  */
 delay(1);
 high(TESTE);
 cont = 0; //124564;
 while (!feof(fp)) {
   dado = fgetc(fp);
   serial(dado);
   cont ++;
   if ((in(P_B) & STATUS) == 0) {
     printf("Erro na programacao - em %ld\n", cont);
     exit(1);
   }
 }
 printf("configurou - %ld bytes\n", cont-1);
 for (dado=0; dado < 100; dado++) {
   high(DCLK);
   low(DCLK);
 }
 printf("terminou\n");
 printf("%ld", time(NULL) - inicial);
}

void serial (BYTE dado)
{
  register BYTE a;

  a = 8;
  while (a --) {
    low(DCLK);
    if (dado & 1)
      high(DATA0);
    else
      low(DATA0);
    dado >>= 1;
    high(DCLK);
  }
  low(DCLK);
}
```

Apêndice D

Arquivo de configuração

```
CHIP chip_name
BEGIN
|M16 : INPUT_PIN = 70;
|PCD0 : BIDIR_PIN = 23;
|PCD1 : BIDIR_PIN = 24;
|PCD2 : BIDIR_PIN = 25;
|PCD3 : BIDIR_PIN = 26;
|PCD4 : BIDIR_PIN = 28;
|PCD5 : BIDIR_PIN = 29;
|PCD6 : BIDIR_PIN = 30;
|PCD7 : BIDIR_PIN = 31;
|PCD8 : BIDIR_PIN = 33;
|PCD9 : BIDIR_PIN = 34;
|PCD10 : BIDIR_PIN = 35;
|PCD11 : BIDIR_PIN = 36;
|PCD12 : BIDIR_PIN = 38;
|PCD13 : BIDIR_PIN = 39;
|PCD14 : BIDIR_PIN = 40;
|PCD15 : BIDIR_PIN = 41;
|ENF2 : INPUT_PIN = 66;
|SBHE : INPUT_PIN = 68;
|PCWR : INPUT_PIN = 65;
|PCRD : INPUT_PIN = 64;
|PCA14 : INPUT_PIN = 63;
|PCA13 : INPUT_PIN = 62;
|PCA12 : INPUT_PIN = 61;
|PCA11 : INPUT_PIN = 56;
|PCA10 : INPUT_PIN = 55;
|PCA9 : INPUT_PIN = 54;
|PCA8 : INPUT_PIN = 53;
|PCA7 : INPUT_PIN = 51;
|PCA6 : INPUT_PIN = 50;
|PCA5 : INPUT_PIN = 49;
|PCA4 : INPUT_PIN = 48;
|PCA3 : INPUT_PIN = 46;
|PCA2 : INPUT_PIN = 45;
|PCA1 : INPUT_PIN = 44;
|PCA0 : INPUT_PIN = 43;
|saida14 : OUTPUT_PIN = 152;
|saida15 : OUTPUT_PIN = 153;
|saida13 : OUTPUT_PIN = 151;
|saida12 : OUTPUT_PIN = 149;
|saida11 : OUTPUT_PIN = 148;
|saida10 : OUTPUT_PIN = 147;
|saida9 : OUTPUT_PIN = 146;
|saida8 : OUTPUT_PIN = 144;
|saida7 : OUTPUT_PIN = 143;
|saida6 : OUTPUT_PIN = 142;
|saida5 : OUTPUT_PIN = 141;
```

```
|saida4 : OUTPUT_PIN = 139;
|saida3 : OUTPUT_PIN = 138;
|saida2 : OUTPUT_PIN = 137;
|saida1 : OUTPUT_PIN = 136;
|saida0 : OUTPUT_PIN = 134;
|relogio : INPUT_PIN = 211;
DEVICE = EPF10K40RC240-4;
END;
```