Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica
Departamento de Sistemas e Controle de Energia
Laboratório de Sistemas Modulares Robóticos

Arquitetura de Hardware para a Extração em Tempo Real de Características de Múltiplos Objetos em Imagens de Vídeo: Classificação de Cores e Localização de Centróides

Fabrício Nicolato

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica

Banca Examinadora:
Prof. Dr. Marconi Kolm Madrid (Orientador) - DSCE/FEEC/UNICAMP
Prof. Dr. José Raphael Bokehi - DEPEB/FUNREI
Prof. Dr. José Raimundo de Oliveira - DCA/FEEC/UNICAMP
Prof. Dr. Roberto de Alencar Lotufo - DCA/FEEC/UNICAMP

Campinas, 13 de março de 2002

À minha noiva Mônica

e aos meus pais Tercílio e Lourdes,

por todo amor, carinho e dedicação.

Agradecimentos

Agradeço a todas as pessoas que de alguma forma contribuíram para a realização deste trabalho. Em especial agradeço:

- Ao meu orientador, professor Marconi Kolm Madrid, pela excelente orientação, amizade, companheirismo e pela crença na minha capacidade intelectual.
- Aos professores da graduação, José Raphael e Oriane, pelo incentivo na continuação da carreira acadêmica.
- Ao professor José Raimundo de Oliveira pela disponibilização do programa Quartus.
- Aos amigos Mário, Jim e Bonani, cujas contribuições foram de suma importância para a finalização deste trabalho.
- A todos os amigos dos laboratórios: LCEE, LADIME, LAT e LABSIM, pela amizade e companheirismo durante estes anos.
- À minha irmã, Deise, e seu marido, Ubaldino, pela amizade, apoio e constante incentivo na realização de meus sonhos.
- Este trabalho foi financiado pelo convênio CNPq e pela FAEP.

Resumo

Esta tese de mestrado foi o início de um projeto de pesquisa que visa a construção de um sistema de *hardware* capaz de extrair características relevantes em imagens de vídeo, tais como cor, posição, orientação, momentos invariantes etc, presentes nas imagens. Foi dada uma abordagem usando técnicas de processamento paralelo e eletrônica reconfigurável com uso de dispositivos lógicos programáveis. Os resultados demonstraram eficiência na determinação de características de interesse quando é necessário se ter informações de medidas de objetos presentes em imagens de vídeo. O principal resultado prático foi a constatação das altas freqüências de operação conseguidas, mesmo quando tratou-se da determinação da localização de múltiplos objetos na cena; justificando sue emprego em diversas aplicações que requerem operação em tempo real, tais como monitoramento de tráfego, controle servo visual etc.

Abstract

This thesis was the beginning of a research project that intend to design a hardware system to extract important characteristics of objects presented by video images, such as color, position, orientation, invariant moments, etc. An approach with a parallel processing technique and programmable electronics was given using programmable logic devices. The results demonstrated great efficiency in the determination of some characteristics of general interest when it is necessary to have information of measures about objects in video images. The main practical result was the high operation frequencies gotten when the system was programmed to determine the location of multiple objects in the scene, justifying its application in real time for a great amount of applications, such as traffic surveilance, visual servoing, etc.

Sumário

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução, Motivação e Organização

As operações de processamento de imagens e visão computacional têm sido consumidoras insaciáveis de desempenho computacional. Aplicações práticas, geralmente, são limitadas pela resolução empregada ou necessária, que pode variar dependendo da taxa de captura de vídeo alcançada pelos algoritmos empregados e da qualidade das imagens, uma vez que o *hardware* geralmente não é rápido o suficiente (Crookes, 1999). Algumas aplicações, tais como as que utilizam imagens provenientes de satélites, equipamentos de raios-X, microscopia, aerofotogrametria e imagens produzidas por equipamentos médicos de alta definição lidam com milhões de *pixels* por imagem. Mesmo tratando-se de operações simples de processamento de imagens, como detecção de bordas, que requerem apenas operações numa vizinhança local dos *pixels* [1], uma enorme quantidade de operações elementares é necessária para processar a imagem inteira. Outras operações, tais como as de monitoramento de tráfego, rastreamento de objetos e controle por visão, requerem processamento em tempo-real. Na tentativa de satisfazer restrições como a de tempo-real, geralmente adota-se algum tipo de relaxação no sistema de processamento, por exemplo: na taxa de vídeo, na resolução das imagens ou na complexidade dos algoritmos empregados.

Duas abordagens são freqüentemente empregadas para melhorar o desempenho dos sistemas de processamento de imagens e visão computacional: uma baseada em *software* e outra em *hardware*. Na primeira, busca-se o desenvolvimento de algoritmos mais eficientes para processamento em microprocessadores padrão. Um exemplo interessante dessa abordagem pode ser encontrado em Hager e Toyama (1998), onde um alto desem-

[1] pixel, segundo o dicionário *Webster*, é o menor elemento de uma imagem que pode ser individualmente processado em um sistema de apresentação de vídeo

penho é alcançado usando uma técnica de "janelamento"na imagem, isto é, somente uma pequena região da imagem é processada por vez, reduzindo-se, dessa forma, a quantidade de dados envolvida durante cada procedimento. A segunda abordagem baseia-se no desenvolvimento de arquiteturas especiais de *hardware* de forma a acelerar certas operações de processamento. O processamento paralelo (Choudray e Patel, 1990), os circuitos dedicados ou de aplicação específica ($ASIC$) (Andreadis *et al.*, 1996) e os processadores de sinais digitais (DSP) (Illgner, 2000) pertencem às abordagens baseadas em *hardware*.

A decisão por qual abordagem adotar depende da tarefa, das habilidades do projetista, se em *hardware*, *software* ou ambos e, obviamente, dos recursos financeiros disponíveis. Em geral, abordagens baseadas em *hardware* oferecem maior desempenho com menor flexibilidade, enquanto que abordagens baseadas em *software* oferecem menor desempenho com maior flexibilidade. Outro fator que influi bastante nesta decisão, e que deve ser levado em consideração, é o tempo gasto na realização do projeto. Esse tempo, tradicionalmente, tem sido bem maior no *hardware* do que no *software*, fazendo com que arquiteturas de *hardware* sejam desenvolvidas para funções que aparecem freqüentemente nas tarefas de processamento, deixando as tarefas mais complexas e menos freqüentes para implementação em *software*.

Na segunda metade da década de 80 surgiram os primeiros dispositivos lógicos programáveis (*Programmable Logic Devices - PLD*) como resultado do aprimoramento dos precursores PLA (*Programmable Logic Arrays*). Tais dispositivos permitem o projeto de circuitos digitais com auxílio por programação e também que, durante as etapas de desenvolvimento, os projetos sejam testados e modificados quantas vezes forem necessárias, com acréscimo muito reduzido de custo. Isso dá aos PLD uma grande versatilidade, abrindo possibilidades de emprego num vasto campo de aplicações, tais como: comunicações, processamento de sinais e imagens, sistemas robóticos etc (Bouridane *et al.*, 1999; Lee e Salcic, 1997).

Os sistemas robóticos são constituídos por sub-sistemas que podem trabalhar paralelamente. O paralelismo existe em diversos níveis. Tornando tais sistemas candidatos a serem beneficiados pelas estratégias da computação paralela. O desenvolvimento de dispositivos de lógica programável possibilita a criação de sistemas digitais complexos em um único componente, utilizando um único ambiente de desenvolvimento. Possibilita também grande integração entre *hardware* e *software*, permitindo que o projetista escolha quais funções devem ser realizadas em *hardware* e quais devem ser realizadas em *software*.

Embora os PLD atuais operem com freqüências de relógio (*clock*) inferiores às dos microprocessadores, microcontroladores e DSP, eles são superiores devido à característica inerentemente paralela de suas arquiteturas. O projetista pode construir o próprio conjunto de instruções, que seja o mais adequado a uma determinada tarefa, e, posteriormente,

substituí-lo por outro sempre que se fizer necessário ou desejável, tornando tais sistemas altamente flexíveis.

Objetivos

Os principais objetivos deste trabalho foram:

1. Ampliar os conhecimentos na tecnologia de implementação de circuitos lógicos em dispositivos lógicos programáveis;
2. Desenvolver uma arquitetura de *hardware*, a ser aplicada em um dispositivo de lógica programável, que faça a localização de múltiplos objetos de uma mesma cor numa seqüência de imagens de vídeo, baseada nos momentos dos objetos na imagem;;
3. Realizar o processamento em tempo-real, levando em conta características inerentes dos sinais de vídeo padrão;
4. Projetar a arquitetura de forma modular para que novas funções possam ser adicionadas ou retiradas do sistema sem afetar o funcionamento das outras funções.

Organização

De modo a fornecer ao leitor os conceitos básicos para o entendimento do trabalho desenvolvido, este texto está organizado da seguinte maneira:

O **capítulo** 2 apresenta conceitos básicos da lógica programável, bem como as principais características do componente e do ambiente de desenvolvimento usados no trabalho.

O **capítulo** 3 apresenta as partes básicas de um sistema de visão computadorizada, as principais características de um sinal de vídeo e os conceitos de processamento de imagens e visão computacional necessários para o entendimento do sistema desenvolvido.

O **capítulo** 4 apresenta o algoritmo do sistema de localização, sua implementação em *software* e também a descrição da arquitetura de *hardware* desenvolvida.

O **capítulo** 5 apresenta alguns resultados importantes obtidos nas simulações funcional e temporizada dos circuitos implementados, bem como uma análise destes resultados, mostrando a eficiência do projeto implementado, tanto no aspecto de rapidez de

processamento como na utilização eficiente dos recursos lógicos disponíveis no componente usado.

O **capítulo** 6 apresenta as conclusões do trabalho, as perspectivas, assim como sugestões para trabalhos.

O **apêndice A** apresenta alguns dos circuitos mais importantes usados na implementação dos blocos que constituem a arquitetura para a extração das características dos objetos presentes nas imagens.

O **apêndice B** apresenta alguns fundamentos da aritmética de ponto fixo, que são relevantes em projetos que utilizam lógica programável.

Capítulo 2

Dispositivos Lógicos Programáveis

Este capítulo apresenta características importantes dos Dispositivos Lógicos Programáveis, direcionado-se para o dispositivo específico usado neste trabalho de Tese.

2.1 Introdução

O termo dispositivo lógico programável (*Programmable Logic Device* - *PLD*) refere-se a qualquer tipo de circuito usado para implementar circuitos digitais, onde o dispositivo pode ser configurado pelo usuário para realizar uma vasta gama de projetos (Brown e Rose, 1996). A configuração de um *PLD* freqüentemente envolve a colocação do componente numa unidade especial de programação, mas alguns componentes também podem ser configurados no próprio sistema (*in-system*).

O primeiro tipo de componente programado pelo usuário apto a implementar funções lógicas foi a *Programmable Read-Only Memory* (*PROM*), na qual as linhas de endereço podem ser usadas como entradas do circuito lógico e as linhas de dados como saídas. Como funções lógicas raramente requerem mais que alguns termos-produto, e uma *PROM* contém um decodificador completo para os seus endereços de entrada, ela é um tipo de arquitetura ineficiente para a implementação de circuitos lógicos e, portanto, dificilmente usada na prática para este propósito. Posteriormente, outros dispositivos desenvolvidos para implementar circuitos lógicos foram os *Programmable Logic Arrays* (*PLA*) e os *Programmable Array Logic* (*PAL*), que são dispositivos com dimensões físicas relativamente pequenas constituídos por dois níveis de lógica, um plano *E* (*AND*) seguido por um plano *OU* (*OR*). Os *PLA*, *PAL* e vários pequenos *PLD* estão agrupados em uma

categoria denominada de PLD simples ($SPLD$), cujas características mais importantes são o baixo custo e o alto desempenho em velocidade de processamento. Com o avanço da tecnologia, tornou-se possível construir dispositivos com capacidades mais elevadas que os $SPLD$. A dificuldade de se aumentar a capacidade de uma arquitetura $SPLD$ rígida é que a estrutura dos planos lógicos programáveis cresce muito rapidamente em função do aumento do número de entradas (Brown e Rose, 1996). O único modo factível de se prover uma maior capacidade para os dispositivos baseados na arquitetura $SPLD$ é através da integração de múltiplos $SPLD$ em um único componente, de modo a conectar seus blocos via programação. Atualmente, existem vários tipos de PLD no mercado baseados nesse tipo de arquitetura (Atmel, 2001; Lattice, 2002; Altera-Max, 2001; Xilinx, 2002). Eles são coletivamente referenciados como PLD complexos ($CPLD$).

Os PLD de propósitos gerais com maior capacidade disponíveis atualmente são os conhecidos como *Programmable Gate Arrays*($FPGA$). Os $FPGA$ são constituídos por um conjunto de elementos de circuito, chamados elementos lógicos (*logic elements* - LE), circundados por blocos de entrada e saída (E/S), figura 2.1. Os LE implementam funções lógicas simples, e seus blocos de E/S são conectados entre si por meio de interconexões programáveis. Essas interligações programáveis permitem que diversos LE possam ser conectados para implementar circuitos digitais combinacionais e/ou seqüenciais complexos.

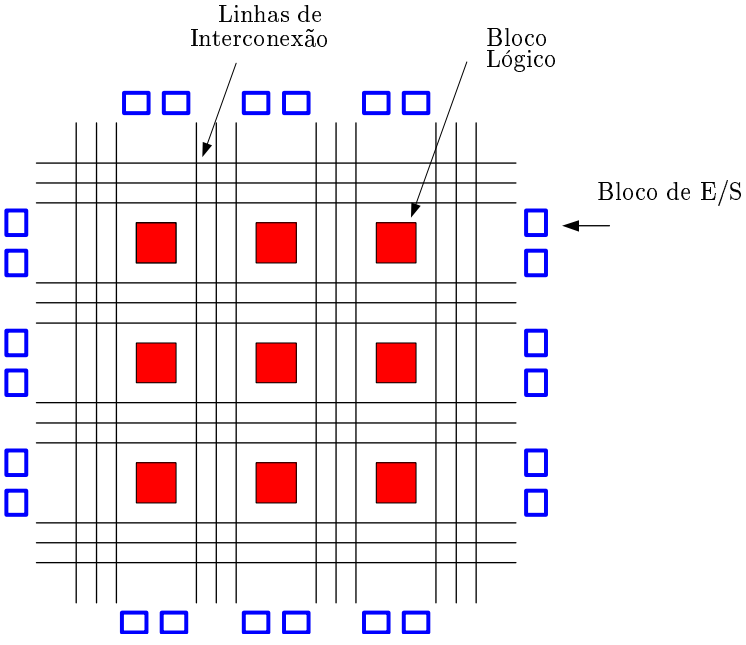

Figura 2.1: Estrutura básica de um $FPGA$.

2.1.1 Métodos de Configuração

Dependendo de como são construídos, os PLD dividem-se em duas categorias: os programáveis apenas uma vez (*one-time programming* - OTP) e os reprogramáveis. Esses últimos, por sua vez, dividem-se em: reconfiguráveis estaticamente e reconfiguráveis dinamicamente. Os reconfiguráveis estaticamente são configurados antes ou depois da realização de uma tarefa e não permitem configuração parcial. Na reconfiguração dinâmica, também chamada de *run-time reconfiguration* (RTR), *on-the-fly reconfiguration* ou *in-circuit reconfiguration*, as mudanças estruturais na configuração do PLD podem ser feitas durante a realização de uma tarefa, isto é, o sistema digital é dividido em sub-circuitos que são configurados no componente somente quando usados.

Embora a reconfiguração dinâmica seja muito atrativa e desejada em diversas áreas, como é o caso dos sistemas com *hardware* evolutivo (*Evolvable hardware*), sua aplicação ainda carece de maiores avanços tecnológicos e de ferramentas de projeto mais adequadas (Mesquita, 2002), sendo este um assunto de intensas pesquisas científicas na atualidade (Thompson, 1997; Thompson, 2002) . O fator que determina se um dispositivo é OTP, estaticamente programável ou RTP é a técnica de programação. Algumas das tecnologias de programação mais utilizadas são: a fuse, a anti-fuse, a $SRAM$ e a E^2PROM ($EEPROM$). A tecnologia *fuse* foi a usada inicialmente para a lógica programável e ainda é utilizada para construir $SPLD$ bipolares, ela é constituída por conexões de metal que podem ser programadas (conexão aberta) passando-se corrente elétrica através delas. A tecnologia E^2PROM é similar àquela usada em dispositivos $EPROM$ padrão. Ela é empregada comumente em $SPLD$ e $CPLD$. Uma célula de memória $EEPROM$ é fisicamente maior que uma célula $EPROM$, mas oferece a vantagem de poder ter seu conteúdo apagado eletricamente.

Os dispositivos baseados em $SRAM$ possuem tecnologia similar a daqueles de RAM estática, porém com algumas poucas modificações. As células de RAM num dispositivo programável são geralmente projetadas para priorizarem estabilidade ao desempenho de leitura e escrita. Conseqüentemente, as células de RAM num dispositivo programável apresentam baixas impedâncias conectadas à tensão de alimentação (VCC) e terra (GND) para proporcionarem maior estabilidade frente às flutuações de tensão. As memórias estáticas são voláteis, isto é, perdem o seu conteúdo quando a tensão de alimentação é retirada. Por isso, os dispositivos baseados em $SRAM$ devem ser configurados toda vez que o sistema entrar em operação. Como resultado, os dispositivos baseados em $SRAM$ são comuns em aplicações onde a função do dispositivo é mudada dinamicamente.

A seguir são apresentados alguns aspectos importantes sobre o projeto de circuitos que utilizam dispositivos lógicos programáveis.

2.1.2 Projeto com Dispositivos Lógicos Programáveis

O projeto de sistemas digitais destinados a implementação em dispositivos lógicos programáveis é é realizado por ferramentas denominadas *Eletronic Design Automation* - EDA). Uma ferramenta de *EDA* para *PLD* típica inclui programas para as seguintes tarefas: entrada inicial para o projeto,otimização, adequação ao dispositivo, simulação e configuração, figura 2.2.

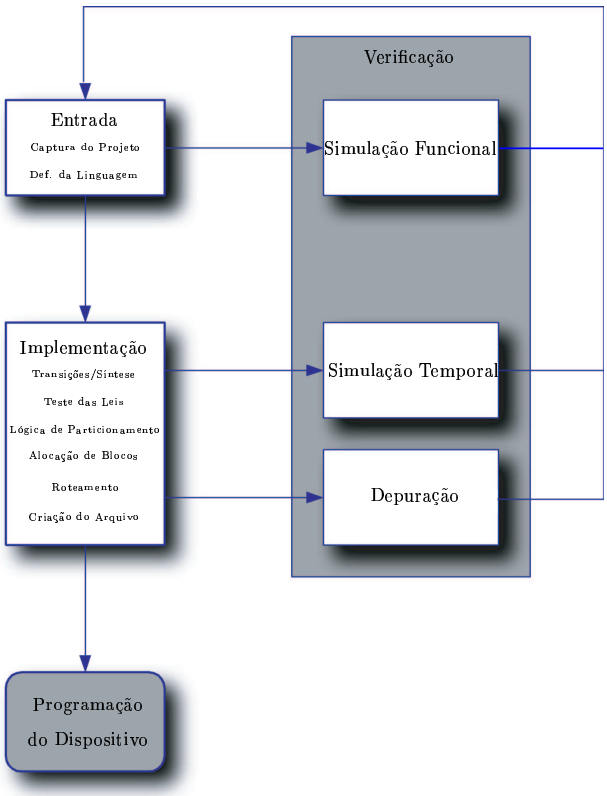

Figura 2.2: Fluxo de projeto em lógica programável.

Existem algumas ferramentas computacionais usuais e apropriadas para realizar a entrada do projeto. Alguns projetistas preferem a entrada através de esquemáticos, enquanto outros preferem o uso das linguagens de descrição de *h*ardware (*Hardware Description Languages* - *HDL*), tais como *Verilog*, *VHDL*, *Handle-C* entre outras, havendo ainda aqueles que preferem combinações entre as duas.

Tradicionalmente, as ferramentas baseadas em diagramas esquemáticos propiciam maior controle sobre o particionamento e posicionamento da lógica no dispositivo. Porém, tal benefício vem em detrimento do tempo gasto na realização do projeto, uma vez que o mesmo é executado no nível de portas lógicas. Já, as ferramentas baseadas em

linguagens permitem que o projeto seja realizado num nível de abstração mais elevado (por ex. descrição comportamental em $VHDL$), possibilitando maior rapidez no processo de desenvolvimento. As desvantagens comuns das HDL são a diminuição do desempenho e a densidade lógica necessária (quantidade de lógica implementada por unidade de área do componente) (OptiMagic, 2000).

Atualmente, os principais fabricantes de PLD disponibilizam bibliotecas de funções (em alguns casos, parametrizadas) básicas (Altera-LPM, 1996); tais como somadores, multiplicadores, multiplexadores, memórias etc, que podem ser "chamadas" a partir de editores de esquemáticos ou instanciadas por meio de HDL. Adicionalmente, há uma ampla pesquisa sobre tradutores que fazem a conversão de programas escritos em linguagens de alto nível (Sankaran e Haggard, 2001; Rinker *et al.*, 2001).

A verificação ocorre em vários níveis, e durante vários passos do procedimento, dependendo dos critérios adotados pelo projetista. Há alguns tipos fundamentais de verificação quando emprega-se dispositivos de lógica programável. A simulação funcional é realizada em conjunto com a entrada do projeto, mas antes do posicionamento e roteamento, isso intenciona a verificação da sua funcionalidade lógica. A simulação considerando as características de temporização é realizada após a etapa de posicionamento e roteamento. Neste passo, o programa de desenvolvimento determina os atrasos do circuito, possibilitando a verificação completa de sua temporização.

Uma boa técnica para projetos com lógica programável é, primeiramente, realizar uma simulação funcional para determinar o funcionamento correto do circuito, verificar a temporização e, finalmente, verificar a funcionalidade completa testando-o no sistema, incluíndo dispositivos físicos e as exigências ambientais da aplicação.

Muitos dispositivos de lógica programável têm a grande vantagem de poderem ser programados, ou re-programados, no sistema, isto é, não necessitarem de uma unidade especial de programação. Assim, o projeto pode ser facilmente verificado no sistema real, reduzindo-se a necessidade da criação de vetores de simulação muito complexos.

Depois que o arquivo de programação foi criado, o dispositivo é configurado e estará pronto para a operação. O método de programação depende da tecnologia dos componentes alvo. Algumas tecnologias, incluindo as $PROM$ para dispositivos baseados em $SRAM$, requerem algum tipo de dispositivo de programação. Dispositivos que podem ser programados no sistema nem sempre necessitam de um programador físico, mas requerem algum tipo de recurso para carregar o arquivo de programação no *chip*. Isto pode ser realizado com auxílio de um microprocessador, microcontrolador ou via portas $JTAG$.

Finalmente, é importante salientar que, seja qual for a tecnologia ou fabricante escolhido para implementar o projeto, o conhecimento da arquitetura do componente, bem

como da ferramenta de projeto ajuda bastante na obtenção de melhores resultados.

2.2 Síntese Utilizando VHDL

Um forte atrativo das linguagens de descrição de *hardware*, e mais especificamente a $VHDL$, é a possibilidade de empregá-las na síntese de dispositivos $ASIC$ e PLD. A síntese é um método automático de conversão de uma linguagem de alto nível para outra de mais baixo nível de abstração. Há várias ferramentas de síntese disponíveis atualmente, entre as quais pode-se citar: *LeonardoSpectrun*, *FPGAExpress*, *Symplify*, *Synopsys* etc. Essas ferramentas convertem descrições apresentadas em formato *Register Transfer Level* (RTL) para *netlists* no nível de portas lógicas. Estes *netlists* consistem de macro-células inter-conectadas, e os modelos para as células no nível de portas lógicas ficam contidos em bibliotecas específicas para cada tecnologia utilizada ou suportada. Um *netlist*, geralmente, pode ser otimizado por área e por velocidade. As entradas para o processo de síntese são as descrições RTL em $VHDL$, as restrições e atributos do circuito, e a(s) biblioteca(s) da tecnologia alvo.

Uma descrição RTL é caracterizada por um estilo que especifica todos os registradores num projeto e a lógica combinacional existente entre eles (Perry, 1998). Os registradores são descritos tanto pela instanciação de componentes quanto implicitamente através de inferências. O circuito combinacional pode ser descrito por equações lógicas, estruturas de controle de fluxo ($CASE, IF/THEN/ELSE$ etc), sub-programas ou comandos concorrentes, e os *loops* geralmente são implementados usando máquinas de estado (Bezerra e Gough, 2000).

Para converter uma descrição RTL para portas lógicas básicas, três passos são necessários em geral. Primeiramente, a descrição RTL é traduzida para uma descrição booleana não otimizada com uso de portas lógicas, geralmente, dos tipos E e OU, *flip-flops* e *latches*. Ainda que correta, normalmente esta tradução não visa fazer otimizações. Em seguida, algoritmos de otimização booleana são aplicados sobre a descrição obtida no passo anterior no sentido de procurar por uma descrição equivalente e otimizada. Finalmente, a descrição booleana otimizada resultante é mapeada para as portas lógicas fazendo-se uso da biblioteca da tecnologia do componente alvo.

A tradução da descrição RTL para a descrição booleana equivalente, usualmente, não é controlada pelo usuário. Por outro lado, a descrição intermediária gerada, geralmente, tem um formato otimizado por ferramentas particulares e pode ficar transparente para o usuário. Todas as estruturas IF, $CASE$ e $LOOP$, associações condicionais de sinais, e associações de sinais selecionados, são convertidas para suas formas booleanas

equivalentes durante o processo intermediário. Os *flip-flops* e *latches* também poder ser instanciados ou inferidos; em ambos os casos mantêm-se os mesmos *flip-flops* ou *latches* na descrição intermediária.

A seguir são apresentados algumas características de uma família de dispositivos lógicos programáveis denominada *Apex* $20K$.

2.3 Arquitetura Apex 20K

Freqüentemente, muitos projetistas de sistemas usam múltiplos PLD num mesmo circuito, pois existem arquiteturas mais adequadas para cada tipo de aplicação. Os dispositivos da família *Apex* combinam características de três famílias de PLD diferentes. Isso permite realizar a implementação do projeto completo em um único componente. A característica mais importante dos dispositivos *Apex* é sua arquitetura embutida, denominada *MultiCore*, que incorpora estruturas especiais como: tabelas de procura (*look-up tables* - LUT), blocos de termos-produto e blocos de memória embutida.

A arquitetura *MultiCore* é constituída por conjuntos de blocos lógicos (*logic array blocks* - LAB) que, por sua vez, são constituídos por elementos lógicos (*logic elements* - LE). Esses blocos são combinados em estruturas maiores chamadas $MegaLAB$, figura 2.3, que são "LAB de LAB". Cada estrutura $MegaLAB$ contém 16 LAB e uma estrutura adicional chamada de bloco embutido do sistema (*embedded system block* - ESB). Este componente possui também uma estrutura de roteamento global responsável pela interconexão das linhas e colunas dos $MegaLAB$. Além disso, cada $MegaLAB$ possui uma estrutura de interconexão local que conecta todos os LAB e o ESB. Isso permite melhorias no desempenho global a partir da utilização dos recursos de roteamento locais ao invés dos globais.

Na arquitetura *Apex* $20K$, as conexões entre LE, ESB e pinos de E/S são feitas com o uso de interconexões do tipo *FastTrack*. Esses tipos de interconexões consistem de conjuntos de canais rápidos de interconexão dispostos na forma de linhas e colunas. As linhas fazem o roteamento dos sinais através de linhas de estruturas $MegaLAB$. Similarmente, as colunas fazem o roteamento através das colunas dos small $MegaLAB$. Qualquer LE, IOE ou ESB pode ser comandado por outro LE, IOE ou ESB através destas interconexões em linhas e colunas. As interconexões locais também são usadas para fazer as conexões de todos os LE dentro de um mesmo LAB e com os LAB adjacentes.

O ESB é o principal componente da arquitetura *MultiCore*. Cada ESB contém 2048 *bits* programáveis que podem ser formados com lógica de termos-produto, lógica baseada em LUT ou três tipos de memória: *dual-port* RAM, ROM ou memória endereçável

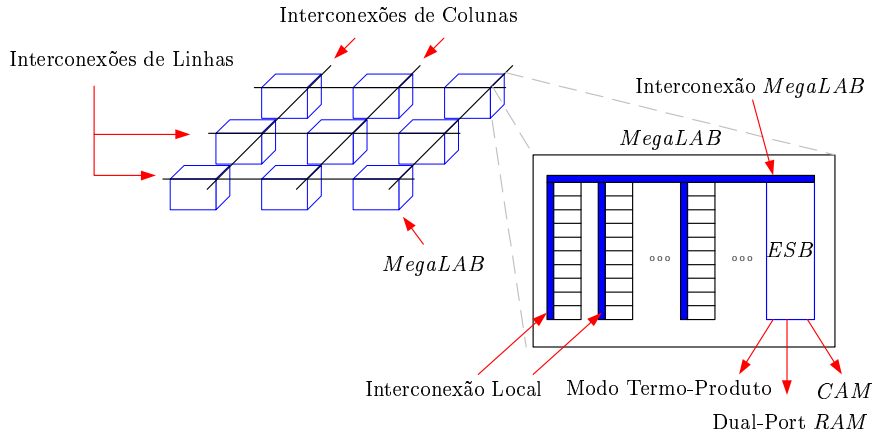

Figura 2.3: Estruturas de roteamento *MegaLAB* e *FastTrack*.

por conteúdo (*content addressable memory* - *CAM*).

O elemento lógico (*LE*) é a menor unidade de lógica presente numa arquitetura *Apex* 20*K*, figura 2.4. Cada *LE* contém uma *LUT* de quatro entradas, que pode implementar funções de até quatro variáveis. Além disso, cada *LE* contém um registrador programável, com conexões de alta velocidade (*carry e cascade chains*); sendo ainda responsável pelo comando das estruturas de roteamento local, *MegaLAB* e *FastTrack*.

Cada registrador programável do *LE* pode ser configurado para operar como um *flip-flop* tipo: *D*, *T*, *JK*, ou *SR*, sendo que seu relógio e seu sinal de controle *clear* podem ser comandados por sinais globais, por pinos de entrada e saída (*E*/*S*) de propósito geral ou por qualquer outro tipo de lógica interna. Para funções combinacionais, o registrador é ignorado e a saída da *LUT* comanda as saídas do *LE*. Cada *LE* possui duas saídas que comandam as estruturas de interconexão local, *MegaLAB* ou *FastTrack*, sendo que cada saída pode ser comandada independentemente pela *LUT* ou pelo registrador. Por exemplo, a *LUT* pode comandar uma saída, enquanto o registrador comanda a outra. Isso, dependendo do caso, possibilita uma melhor utilização do componente, pois, o registrador e a *LUT* podem ser usados simultaneamente para implementar diferentes funções. As conexões entre *LE* adjacentes podem ser feitas através de duas linhas de dados de alta velocidade: *carry chains* e *cascade chains*. Um *carry chain* suporta funções aritméticas calculadas em alta velocidade como contadores e somadores, enquanto um *cascade chain* implementa funções de muitas entradas, tais como comparadores de igualdade com atraso ínfimo.

Na arquitetura *Apex* 20*K*, um *LE* pode operar em três modos distintos: modo normal, modo aritmético ou modo de contador. Cada um desses modos usa diferentes recursos do *LE* e, em cada um, sete entradas estão disponíveis: quatro entradas vindas

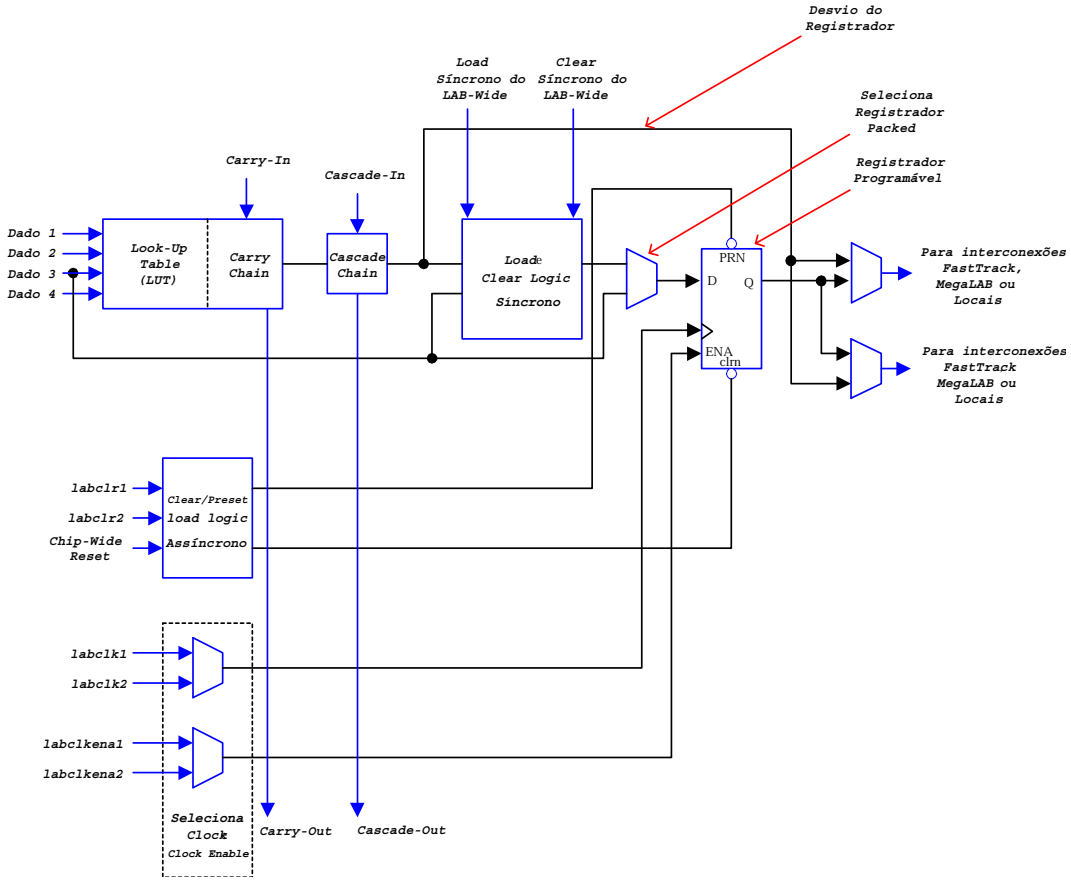

Figura 2.4: Estrutura do elemento lógico do Apex 20K.

da interconexão local do LAB, a realimentação do registrador e os sinais de *carry-in* e *cascade-in* vindas do LE prévio; possibilitando diferentes direcionamentos de modo a implementar as funções lógicas desejadas em conjunto com outros LE. O ambiente de desenvolvimento $Quartus\ II$, juntamente com funções parametrizadas, como por exemplo as LPM, escolhe automaticamente o modo apropriado para implementar contadores, somadores, multiplicadores e outras funções comuns. Se necessário, o usuário também pode criar funções de propósito especial e especificar qual modo de operação deve ser usado para obter máximo desempenho.

Os dispositivos $Apex\ 20K$ suportam as características de gerenciamento de relógio *ClockLock* e *ClockBoost* que são implementadas via PLL. O circuito de *ClockLock* usa uma PLL de sincronização que reduz o atraso e o defasamento do relógio dentro do dispositivo, maximizando seu desempenho. O circuito de *ClockBoost* permite a divisão e a multiplicação da freqüência do relógio de entrada. Estes dispositivos possuem ainda uma árvore de distribuição do sinal de relógio de alta velocidade que não exige otimizações

específicas por parte do projetista.

2.4 Ambiente de Desenvolvimento

O projeto de sistemas digitais utilizando a família de *PLD Apex* 20*K* e também de outras famílias da *Altera*, é realizado com auxílio de um ambiente de desenvolvimento chamado *QUARTUS II*, figura 2.5. O processo de projeto usando o programa *QUARTUS* passa por quatro fases: entrada, compilação, verificação e programação (configuração).

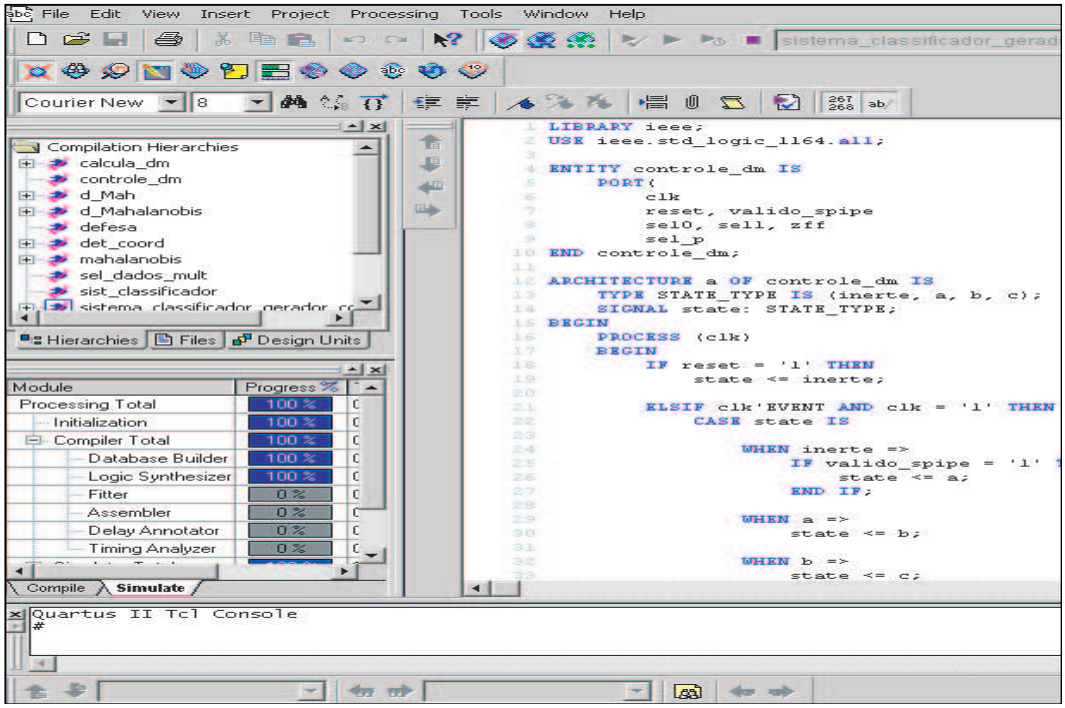

Figura 2.5: Interface gráfica do programa *QUARTUS II*.

O *QUARTUSII* pode integrar múltiplos arquivos de projeto - gerados por ele próprio, ou atráves de outra ferramenta compatível, numa única hierarquia de projeto. No ambiente *QUARTUS*, os erros identificados durante a compilação, simulação e análise temporal podem ser identificados automaticamente e destacados no arquivo original ou no editor de *floorplan*. Se o projeto possui mais de um nível de hierarquia, o usuário pode navegar de uma entidade de projeto para outra não importando se estas são baseadas em captura de esquemáticos ou em forma textual. O editor de blocos permite a criação de arquivos de blocos do projeto (*block design files* .bdf), o que permite a combinação de diagramas de blocos, megafunções, macrofunções e símbolos de primitivas.

O *QUARTUS* também possui um editor de textos para a entrada de arquivos de projeto escritos em *HDL* (*VHDL*, *Verilog* ou *AHDL*), sendo que seu compilador sintetiza lógica a partir de qualquer uma dessas *HDL* e, também, mapear esta lógica para algumas famílias de *PLD*. Ele também permite que funções *LPM* sejam instanciadas em *HDL* e seu editor de *floorplan* simplifica o processo de associação de lógica a pinos e células lógicas do componente. O projetista pode associar pinos e células lógicas antes de compilar um projeto e também verificar e modificar os resultados pós compilação.

O processo de compilação gera arquivos padrão para a programação, simulação e análise da temporização. Após uma compilação inicial, o programa só irá compilar as mudanças feitas no projeto sem fazer novas compilações completas, ou seja, verifica dependências automaticamente. Seu módulo de síntese lógica suporta várias opções de síntese, com aptidão para selecionar rotinas de redução de lógica apropriadas para minimizar, reduzir, ou eliminar partes lógicas redundantes ou desnecessárias.

O módulo de adequação (*fitting*) do compilador aplica regras heurísticas para escolher um meio de implementação para o projeto sintetizado em um ou mais dispositivos, isto é, faz o posicionamento e o roteamento do projeto automaticamente. O *QUARTUS* também pode receber arquivos de *netlist* de outra *EDA* e fazer o posicionamento e roteamento dos circuitos destes arquivos para os dispositivos da *Altera*.

O compilador considera também as restrições de temporização estabelecidas pelo usuário para atrasos de propagação (*propagation delay* - t_{PD}), atrasos de relógio para saída (*clock-to-output delays* - t_{CO}) tempos de estabelecimento (*setup times* - t_{SU}), freqüência do relógio interno (*clock frequency* - f_{MAX}) e freqüência do relógio do sistema. O projetista pode especificar os tempos de processamento para funções lógicas específicas, bem como do projeto todo. Na verificação de um projeto, o programa *QUARTUS* é capaz de realizar simulações funcionais e análises de temporização. O simulador extrai informações da base de dados do projeto durante a compilação para realizar simulações funcionais temporizadas e multi-dispositivo.

O simulador suporta simulação funcional para testar as operações lógicas do sistema depois do mesmo ter sido sintetizado. Ele também realiza simulações temporizadas no projeto após ter sido completamente sintetizado e otimizado. O analisador de temporização pode calcular uma matriz de atrasos ponto-a-ponto do dispositivo, determinar as freqüências de relógio nos terminais dos dispositivos e calcular a freqüência de relógio máxima do sistema.

Capítulo 3

Sistema de Visão Computacional

Este capítulo apresenta fundamentação necessária e comentários importantes no que diz respeito à visão computacional e ao processamento de imagens, tanto em hardware quanto em software, para o entendimento das técnicas desenvolvidas e aplicadas neste estudo. São apresentadas também as características relevantes dos sinais de vídeo, do sistema de aquisição de imagens e do sistema de processamento usado.

3.1 Introdução

A visão computacional pode dotar um robô com um mecanismo sofisticado de sensoriamento, permitindo-lhe responder de maneira mais flexível às exigências do ambiente de trabalho. Embora outros sensores, como aqueles destinados a medir proximidade, toque, força etc; tenham finalidades importantes sobre o desempenho destas máquinas, a visão é reconhecidamente a mais útil e abrangente das capacidades sensoriais, e como é de se esperar, também seus conceitos, técnicas, e *hardware* de processamento associado são normalmente de maior complexidade. O conhecimento e o entendimento dos processos de visão computacional de alto nível são ainda considerados vagos e especulativos.

A visão robótica pode ser definida como o processo de extração, caracterização, e interpretação de informações contidas em imagens produzidas a partir de um espaço tridimensional, e que pode ser subdividido em seis áreas: sensoriamento, preprocessamento, segmentação, descrição, reconhecimento, e interpretação. Sensoriamento é processo de captura da imagem propriamente dita. O preprocessamento aplica técnicas como redução de ruído e melhoramento de detalhes. A segmentação é o processo de particionamento da imagem em objetos de interesse. A descrição trata da determinação de características

(forma, tamanho etc). O reconhecimento é o processo que identifica estes objetos (metal, madeira etc). Finalmente, a interpretação dá significado aos objetos reconhecidos.

É conveniente agrupar estas várias áreas de acordo com a sofisticação envolvida na implementação de um determinado sistema. O modo tradicional de agrupamento considera-os como distribuídos entre três níveis de processamento: baixo, médio e alto. Neste sentido, sensoriamento e preprocessamento são consideradas funções de baixo nível. Processos que extraem, caracterizam e nomeiam (atribuem rótulos) componentes das imagens são considerados de médio nível, portanto, segmentação, descrição e reconhecimento são funções de médio nível. As funções de alto nível referem-se a processos que visam emular habilidades cognitivas (Fu *et al.*, 1987).

As soluções para os problemas envolvidos com o baixo nível geralmente são conseguidas com implementações baseadas puramente em *hardware*, principalmente pela necessidade de rapidez no processamento, pelo domínio tecnológico, e pela relativa simplicidade dos procedimentos envolvidos. Já as funções de médio nível, devido a envolverem procedimentos mais sofisticados, são de grande dificuldade de implementação em *hardware*, e altamente dependentes do tipo da aplicação envolvida. Na atualidade existem muitas aplicações para problemas envolvidos com as funções de médio nível, entretanto, elas estão limitadas a aplicações em processos com constantes de tempo mais longas, e na sua grande maioria solucionadas por *software*.

No caso dos sistemas robóticos, principalmente naqueles onde são empregados acionadores elétricos, as constantes de tempo de resposta são rápidas o bastante (alguns poucos milissegundos) para exigirem soluções que dificilmente podem ser conseguidas somente por *software*, isso se for desejável usar a visão computacional como auxiliar nas tarefas de controle, ou então exigiriam um sistema de processamento com altíssima capacidade computacional, elevando de forma inaceitável o custo global do projeto, obviamente considerando-se o domínio tecnológico atual.

Nas seções a seguir são apresentadas várias informações e comentários sobre os aspectos fundamentais envolvidos com as funções de médio nível de visão computacional, que motivaram a criação do sistema de processamento via *hardware*, que forneceu soluções bastante eficientes para a resolução de problemas deste nível da visão computacional.

3.2 Imagem Digital

Uma imagem digital é composta por um conjunto de pontos, denominados *picture elements (pixels)*. Esses pontos estão dispostos na tela do computador formando uma matriz de pontos que é denominada de mapa de *bits* (*bitmap*), e cada elemento desta matriz

possui informação referente ao nível de cinza ou à cor associada a um ponto específico da imagem. Uma imagem digital possui também uma determinada resolução associada a ela, ou seja, o número de elementos que possui nas coordenadas horizontal e na vertical, sendo que cada elemento da imagem possui uma localização específica em relação ao sistema de coordenadas adotado, figura 3.1.

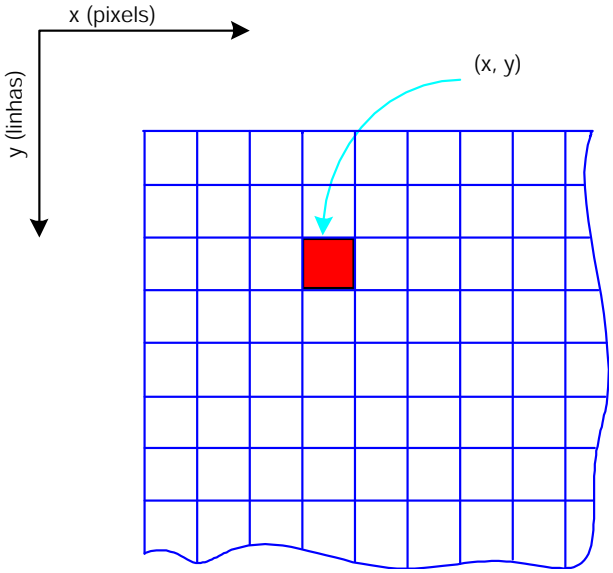

Figura 3.1: Imagem digital.

3.3 Modelos de Cor

O propósito de um modelo de cores é facilitar a especificação das cores em alguma forma padrão. Essencialmente, um modelo de cor é uma especificação de um sistema de coordenadas tridimensionais e um subespaço dentro desse sistema no qual cada cor é representada por um único ponto (Gonzalez e Woods, 2000). Esta representação tridimensional possibilita a manipulação simples da informação de cor e é uma maneira natural de visualizar as relações espaciais entre as cores. O espaço de cores RGB, figura 3.2, é o formato mais comum para imagens digitais pois é compatível com uma vasta gama de monitores de computador e câmeras de vídeo.

No modelo RGB, cada cor é representada como uma tríplice (R, G, B) onde R, G e B representam respectivamente as componentes de cor vermelha (*red*), verde (*green*) e azul (*blue*). Imagens no modelo RGB consistem em três planos de imagem independentes, um para cada cor primária. Quando alimentadas num monitor RGB, essas três imagens combinam-se sobre a tela fosfórea para produzir uma imagem de cores compostas, figura 3.3.

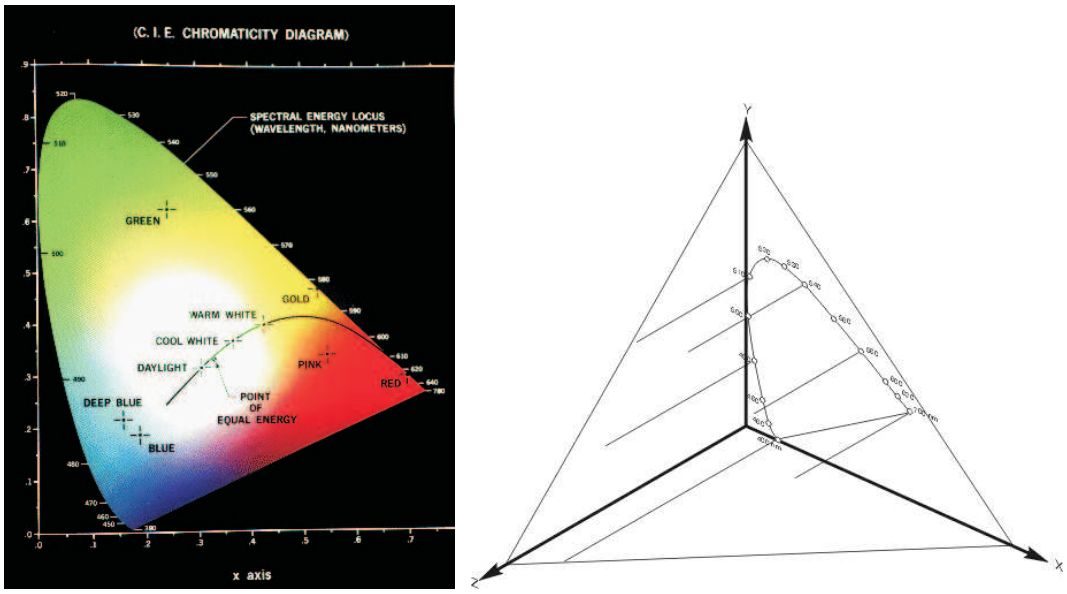

Figura 3.2: Mapeamento RGB.

Para um melhor processamento das imagens coloridas, o espaço RGB pode ser transformado em espaços de cores alternativos. Alguns desses espaços alternativos são destinados ao uso por exemplo em: impressoras (CMY), na transmissão de sinais de televisão colorida (YIQ, $Icrcb$) e na manipulação de imagens coloridas (HSI, HSB, HSV etc). É importante salientar que, diferentemente do RGB, alguns desses espaços alternativos fazem a separação das componentes de luminância (brilho) e de crominância (cor), justamente para facilitar o processamento de imagens coloridas. Dentre esses espaços de luminância-crominância, dois exemplos importantes são o YES e o HSI. No modelo YES (Xerox, 1989), Y representa a luminância, enquanto que E e S representam a crominância. O modelo YES é definido por uma transformação linear das coordenadas RGB dada por

$$\begin{bmatrix} Y \\ E \\ S \end{bmatrix} = \begin{bmatrix} 0,253 & 0,684 & 0,063 \\ 0,500 & -0,500 & 0,000 \\ 0,250 & 0,250 & -0.500 \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.1}$$

Como exemplo de um modelo não-linear que faz a separação da cor e da intensidade luminosa, pode-se citar o HSI. Onde a componente H *(Hue)* representa o matiz, S *(Saturation)* representa a saturação e I *(Intensity)* a intensidade.

A obtenção das componentes H, S e I a partir de R, G e B é conseguida através das seguintes equações[1]

[1] A dedução da transformação entre os espaços de cor RGB e HSI apresentado poder ser encontrada em Gonzalez e Woods (2000).

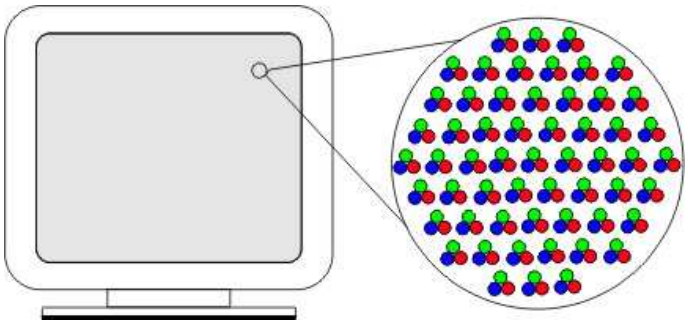

Figura 3.3: Monitor RGB.

$$
\begin{aligned}
I &= \tfrac{1}{3}\left(R+G+B\right) \\
S &= 1 - \tfrac{3}{R+G+B}\left[min(R,G,B)\right] \\
H &= \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}}\right\}
\end{aligned}
\tag{3.2}
$$

3.4 Padrões de Vídeo

Os sinais de vídeo que são enviados ou recebidos por vários dispositivos, como câmeras de vídeo, monitores, vídeos cassete, DVD, geradores de sinais e equipamentos de medida, geralmente se adequam às especificações de certos padrões de vídeo. Assim, os sinais de vídeo podem ser compartilhados por diversos equipamentos de forma direta ou por simples transcodificação de padrões. Um padrão de vídeo especifica os tempos de sincronização e parâmetros elétricos do sinal de vídeo (Baxes, 1994).

Todos os padrões de vídeo dividem uma imagem em linhas horizontais, procedimento este chamado *raster*, iniciando com a linha na parte superior da imagem e terminando com a linha na parte inferior da mesma. Cada linha é percorrida no sentido da esquerda para a direita. O brilho, ou nível de cinza, da imagem é representado por um nível de tensão; geralmente tensões baixas representam os tons escuros e tensões mais altas representam tons claros. Os sinais de sincronização são adicionados ao sinal de brilho para significar o início de cada linha e de cada campo (ou quadro) da imagem, e o sinal de vídeo final obtido normalmente aparece com tensão variável.

Adicionalmente, uma técnica de varredura entrelaçada é utilizada. O quadro é dividido em duas partes, uma composta por linhas ímpares (campo impar) e outra composta por linhas pares (campo par). Inicialmente, varre-se o campo ímpar e em seguida o campo par. O esquema de entrelaçamento é utilizado para que o olho humano tenha a sensação de que a imagem é atualizada duas vezes mais rápido do que realmente acontece. Isto resulta numa imagem com menor cintilação

luminosa (*flicker*) (Baxes, 1994). Exemplos de formatos padrão de vídeo são o $RS-170$ e o $CCIR$. Estes formatos são utilizados nos EUA e na Europa respectivamente e definem os parâmetros para sinais de vídeo monocromáticos. As especificações do padrão EIA (*Eletronics Industries Association*) $RS-170$ podem ser resumidas da seguinte forma:

- 525 linhas por quadro;
- 30 quadros (ou 60 campos) por segundo;
- varredura entrelaçada de $2:1$;
- razão de aspecto de $4:3$;
- sinal de vídeo com amplitude de 1 volt.

Uma razão de aspecto de $4:3$ significa que a relação entre a largura e a altura da tela é de $4/3$. Cada linha de um sinal $RS-170$ é composta por uma tensão analógica variável que representa o brilho da imagem naquela linha. Um pulso de sincronização horizontal separa cada linha de sua subseqüente. Um pulso se sincronização mais longo, denominado sincronização vertical, separa cada campo da imagem. Como um quadro de vídeo $RS-170$ é constituído por 525 linhas. Cada quadro completo ocorre a cada $33,33$ ms ($1/30$ s). Isso significa que cada campo contém $262,5$ linhas e ocorre a cada $16,67$ ms ($1/60$ s). Dividindo-se o tempo de cada campo pelo número de linhas em um campo, obtém-se o tempo de linha, que é igual a $63,49$ μs ($16,67$ ms$/262,5$ linhas). A determinação do tempo de linha é de suma importância na digitalização pois, determina o quão rápido o processo de digitalização deve ser para que se obtenha o número de pixels desejado por linha na imagem digital resultante.

No sinal de vídeo $RS-170$, o intervalo de sincronização vertical ocorre durante as primeiras nove linhas de cada campo. Logo após as linhas de sincronização vertical, ocorrem 11 linhas que não carregam informações de vídeo. Assim, um total de $9+11+242,5$ tempos de linha compõe um campo completo. Dentro de cada linha há um intervalo de sincronização horizontal de $10,9$ μs. Isso deixa $52,59$ μs do tempo de linha para o sinal de vídeo. A imagem digital é obtida fazendo-se a digitalização do sinal dentro dos tempos linha ativa ($52,59$ μs).

A especificação $RS-170$ também apresenta informações sobre os níveis de tensão para o sinal de vídeo e para a sincronização. A amplitude deste sinal varia no intervalo $[-0,286;+0,714]v$. A porção do sinal que varia de $0,143v$ até $0,714v$ é utilizada para caracterizar os níveis de cinza (do preto ao branco) da imagem. O nível de $0v$ é comumente denominado nível de *blanking*, nele o sinal é considerado "mais preto do que preto". Os pulsos de sincronização vão de $-0,286v$ até $0v$.

O padrão de vídeo $CCIR$ (*Comitê Consultatif International dês Radiocommunications*) é a versão européia do padrão $RS-170$ Esse padrão especifica 625 linhas por quadro com uma taxa de 40 ms, ou 25 quadros por segundo. Da mesma forma que o $RS-170$, o $CCIR$ especifica um rastreamento entrelaçado e uma razão de aspecto de 4:3. Os sinais de sincronização são, na maior parte, similares aos do $RS-170$, exceto que eles são adaptados para uma taxa de vídeo menor (25 quadros por segundo).

3.4.1 Componentes de Cor RGB e HSB

Muitos sistemas que empregam imagens coloridas simplesmente usam três sinais do tipo $RS-170$, um para cada componente de cor - vermelho, verde e azul RGB). Esses sistemas aceitam e geram componentes de vídeo compatíveis com as componentes R, G e B. Existem dois métodos para formar sinais de vídeo coloridos: componente de cor e cor composta. O sinal de cor composto é um sinal físico simples, ambas as informações de cor e brilho estão combinadas no sinal, como mostrado na figura 3.4. Diferentemente, um sinal de componentes de cor é constituído de sinais físicos múltiplos. Quando se usa uma fonte de vídeo RGB, o sistema de processamento de imagens digitais armazena e processa cada canal RGB separadamente. Alternativamente, as componentes RGB podem ser digitalmente convertidas para outro espaço de cor. Um dos mais conhecidos é o de matiz *(Hue)*, saturação *(Saturation)* e brilho *(Brightness)* - HSB - antes que sejam aplicadas quaisquer operações de processamento. Para muitas aplicações, a transformação do espaço de cores proporciona melhores resultados.

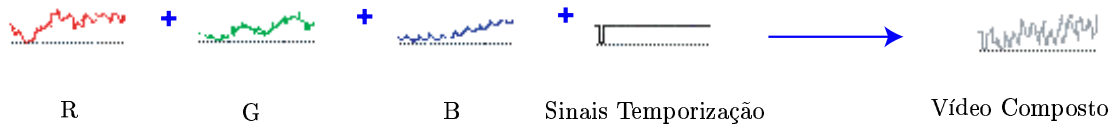

Figura 3.4: Composição do sinal de video.

3.5 Sinal de Vídeo Composto ($NTSC$)

Esse padrão foi criado com o advento da televisão comercial colorida. O sistema de temporização básica do padrão $RS-170$ foi mantida de maneira a fazer com que os consumidores que possuíssem aparelhos de TV monocromáticos pudessem continuar recebendo os sinais de vídeo colorido na forma monocromática. O padrão de TV colorida $NTSC$ (*National Television System Committee*) é também chamado de $RS-170A$, que é o padrão $RS-170$ adaptado para sinais de vídeo coloridos. Tal adaptação trata-se simplesmente de uma adição da informação de crominância (cor) ao sinal monocromático existente. O $NTSC$ é codificado tomando-se as três componentes RGB e, em seguida, combinando-as em sinais intermediários de luminância (brilho) e crominância (cor). Adicionalmente, um sinal de referência de cor, denominado *color burst*, é introduzido no começo de cada linha de vídeo após o pulso de sincronização horizontal. No lado da recepção, o sinal $NTSC$ é decodificado separando-se os sinais de luminância e crominância. O sinal de crominância é então demodulado usando o *color burst* como o sinal de referência para a demodulação . Os sinais de cor demodulados são finalmente combinados com o sinal de luminância para formar os sinais R, G e B. Um detalhamento maior sobre a composição de sinais de vídeo pode ser encontrado em Grob (1979).

3.6 Componentes de um Sistema de Processamento de Imagens e Visão Computacional

Um sistema de processamento de imagens digitais e visão computacional é uma coleção de componentes de *hardware* e *software* que fazem a aquisição, armazenamento, processamento (melhoramento, extração de características, reconhecimento de padrões etc) e a exibição das imagens digitais, conforme mostrado na figura 3.5. Embora esses componentes possam ser fisicamente separados, cada um é fundamentalmente necessário para completar o ciclo de processamento de imagens digitais.

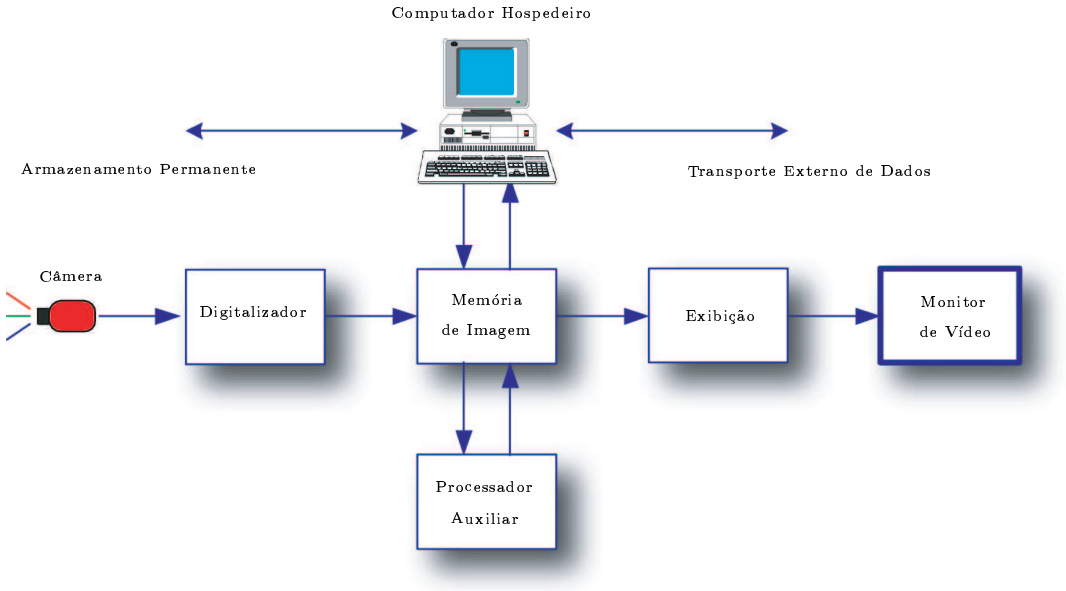

Figura 3.5: Componentes básicos de um sistema de processamento das imagens.

O primeiro estágio em qualquer sistema de processamento de imagens digitais é a aquisição das imagens. Este é um processo de 2 passos, requerendo algum dispositivo sensor e uma função de digitalização. Há muitos tipos de sensores diferentes. Essencialmente, o sensor deve criar um sinal elétrico que represente o brilho da cena em questão. Uma câmera de vídeo é o sensor mais comumente usado. Outros dispositivos sensores de imagens são: os *scanners*, os tomógrafos, os satélites etc. Uma câmera de vídeo converte a imagem óptica em um sinal elétrico analógico. Ela usa lentes para focar os raios de luz para um fotodetector bi-dimensional. O fotodetector converte a energia luminosa em um sinal elétrico proporcional que representa a imagem.

A função de digitalização vem em seguida a aquisição. Na digitalização, o sinal analógico é convertido para a forma digital através de uma função de conversão A/D. A imagem digital é então armazenada numa memória digital, geralmente em dispositivos semicondutores de alta velocidade. Uma vez que a imagem esteja na memória, ela torna-se acessível para operações de processamento subseqüentes. Para exibir uma imagem armazenada na memória, seus dados são lidos dela e enviados para dispositivos de exibição, tais como impressoras e monitores de vídeo.

Em alguns casos é necessário que os dados digitais sejam convertidos novamente para a forma analógica por meio de um conversor digital-analógico (D/A), para então serem exibidos.

Um computador mestre controla o sistema. Ele proporciona interface com o usuário e também faz o seqüenciamento da aquisição, armazenamento, processamento e exibição. A imagem digital armazenada na memória pode ser livremente acessada pelo computador mestre. Ele também pode transferir imagens da memória para outros computadores, vários outros dispositivos conectados em rede e para dispositivos de armazenagem permanente (discos rígidos, CD etc). Embora o computador mestre, em princípio, possa realizar qualquer operação sobre imagens armazenadas na memória, sua velocidade de operação pode ser limitada. Geralmente, processadores dedicados são adicionados ao sistema de maneira a aumentar o desempenho global do sistema. Estes processadores, da mesma forma que o computador mestre, têm livre acesso às imagens armazenadas na memória. Estes dispositivos adicionais podem aparecer na forma de circuitos de aplicação específica ($ASIC$), processadores de sinais digitais (DSP), dispositivos de lógica programável (PLD) ou microprocessadores adicionais, otimizados para manipular operações comuns de processamento de imagens.

Os sistemas de processamento de imagens digitais podem assumir várias arquiteturas diferentes. A configuração ideal para uma determinada aplicação pode ser uma mistura de várias combinações de componentes de *hardware* e *software*. A escolha, obviamente, depende do capital disponível, da tecnologia, do nível de habilidade dos projetistas e dos requisitos da aplicação.

3.6.1 Digitalização

Antes que uma imagem possa ser processada digitalmente, o sinal de vídeo analógico original deve ser convertido para a forma digital. Este processo é conhecido como digitalização de imagens. O processo de digitalização pode ser dividido em duas funções principais, a amostragem e a quantização.

A quantização é também conhecida como conversão de analógico para digital (A/D). Em adição, várias funções auxiliares são empregadas a fim de processar o sinal de vídeo analógico antes do processo de quantização propriamente dito. No processo de amostragem de um sinal de vídeo para criar uma imagem digital o primeiro passo é selecionar a taxa na qual a amostragem irá ocorrer. Essa taxa é geralmente baseada na resolução espacial desejada para a imagem resultante. A resolução de linha inerente e a qualidade do sinal de vídeo podem também ser de grande importância na determinação da taxa de aquisição. Considere o caso da amostragem de um sinal de vídeo monocromático padrão $RS-170$. Na discussão sobre a temporização $RS-170$ vários fatos importantes aparecem. Primeiramente, um sinal de vídeo $RS-170$ tem 485 linhas de informação visual. Segundo, o tempo de linha ativa, que é a parte de cada linha onde a informação visual está contida, é igual a $52,59$ μs. O sinal $RS-170$ também tem razão de aspecto de $4:3$. Com essas informações, é possível calcular a taxa de amostragem necessária para criar uma imagem digital. Com a razão de aspecto de 4:3 e o número de linhas ativas, calcula-se o número de pixels por linha como sendo 485 x $4/3 = 646,66$ *pixels*. Estes *pixels* serão quadrados, ou seja, eles cobrem uma área quadrada na imagem.

A taxa de amostragem é calculada como sendo a divisão do tempo de linha ativa pelo número de *pixels* em cada linha. Essa divisão resulta numa taxa de $81,33$ ns $(52,59/646,66)$, que é equivalente à freqüência de $12,3$ MHz $(1/81,33$ $ns)$. Na prática, algumas linhas e pixels são desconsiderados na imagem original para resultar numa imagem digital de 640 *pixels* x 480 linhas. Para criar a imagem, a aquisição dos *pixels* é feita em cada porção de linha ativa em intervalos iguais ao da taxa de amostragem $(81,33$ $ns)$. Este período é também conhecido como período de relógio (*clock*) de *pixel* ou tempo de *pixel* e, é fator crítico no projeto de um sistema de digitalização de vídeo. Durante este tempo, o pixel amostrado deve ser quantizado por meio de um conversor A/D e, então, ter este valor quantizado armazenado na memória de imagem.

O fluxo de digitalização pode ser entendido referindo-se ao diagrama de temporização do padrão $RS-170$, figura 3.6.

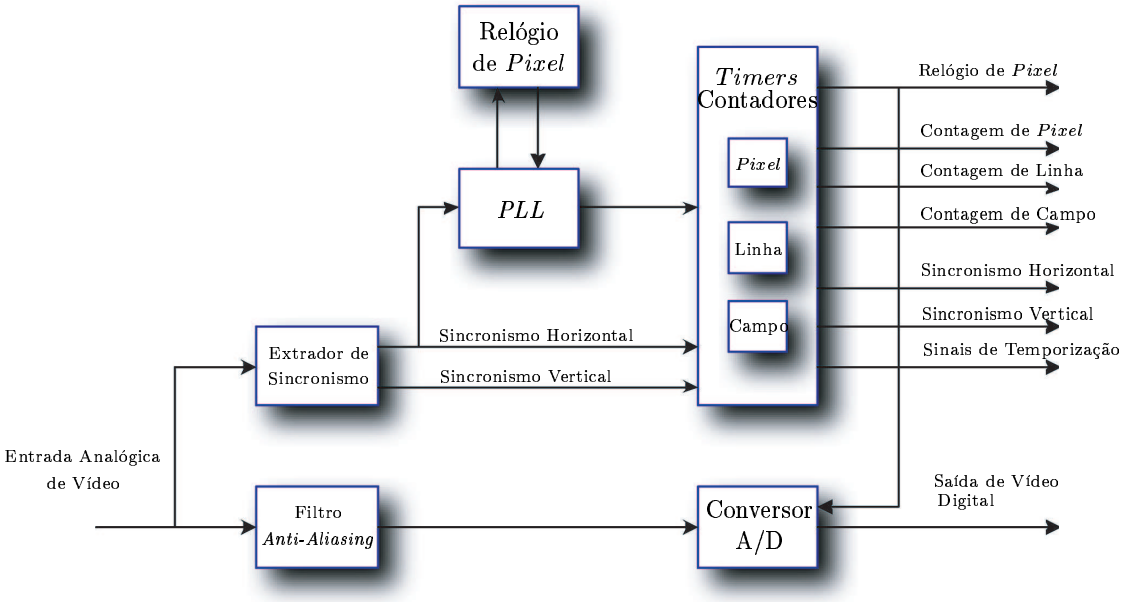

Figura 3.6: Processo de digitalização de imagem.

O processo começa com o sinal de sincronização vertical, que indica o início de um campo (aqui considerado o ímpar). O contador de linhas recebe então o valor do ponto inicial. As primeiras vinte linhas (9 de sincronização vertical e 11 qua não contêm informações de vídeo) são ignoradas pelo digitalizador. Quando ocorre o 21^o pulso de sincronização horizontal, chega-se à primeira linha de vídeo válida (ativa). Nesse ponto, o sinal de vídeo está no canto superior esquerdo da porção ativa do quadro da imagem. O contador de *pixels* é zerado. Os $10,9$ μs correspondentes ao período de sincronização horizontal são ignorados, então, a digitalização tem início. Numa taxa de $81,33$ ns por *pixel*, são amostrados, quantizados e armazenados 640 pixels da linha de vídeo. No final da linha de vídeo, a digitalização pára, e o processo aguarda o próximo pulso de sincronização horizontal. Quando o pulso de sincronização horizontal termina, a digitalização tem início novamente. Este processo se repete linha por linha até o fim do campo.

3.6.2 Extração do Sincronismo e Temporização

Os sinais de sincronização vertical e horizontal encontrados no sinal de vídeo de entrada estabelecem a temporização do processo de digitalização. Um circuito de extração de sincronismo detecta os níveis de tensão do sinal de vídeo. Um sinal de sincronização ocorre quando ocorre uma transição no sinal de vídeo para a faixa de sincronismo, isto é, a tensão deste sinal fica abaixo do nível de *blanking*. A duração do sinal de sincronismo é, então, avaliada para determinar se este é um pulso de sincronização horizontal ou vertical. Um pulso de sincronização de curta duração indica um sinal de sincronização horizontal, enquanto que um pulso longo indica um sinal de sincronização vertical.

Os sinais de sincronização são usados para gerar vários sinais de temporização. Os sinais de temporização coordenam o processo de digitalização, bem como os processos de armazenamento e exibição. Primeiramente, um sinal de relógio de *pixel* é gerado usando um oscilador operando na taxa desejada para a digitalização (ex. $12,3$ MHz para um sinal $RS-170$). Um circuito de *phase-locked-loop* (PLL) trava o relógio de *pixel* ao pulso de sincronização horizontal, figura 3.7.

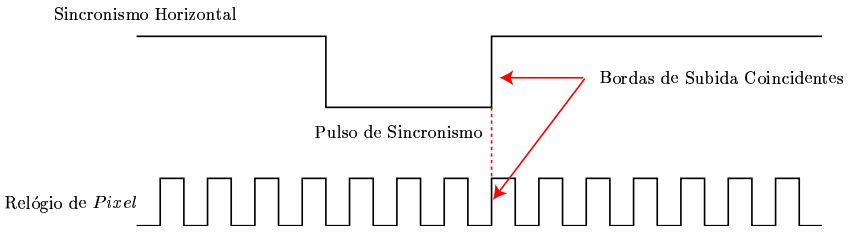

Figura 3.7: PLL para eliminação do *jitter* na imagem.

O propósito do travamento de fase é duplo. Inicialmente, deseja-se garantir que o primeiro *pixel* de cada linha de vídeo seja colocado precisamente na mesma localização do lado esquerdo do quadro da imagem. De outra forma, podem ocorrer oscilações (*jitter*) horizontais na imagem degradando a sua qualidade visual. O segundo propósito é o de manter constantes a temporização do relógio de *pixel* e outros sinais de temporização da digitalização, mesmo quando ocorrer algum distúrbio no sinal de vídeo.

De maior importância são os contadores de *pixels*, de linhas e de campos. Para um sinal de vídeo $RS-170$, o contador de *pixels* conta de 0, no início da linha, até 639, no final. Similarmente, o contador de linhas conta de 0, no início do campo, até 239, na parte inferior ou final do campo. O contador de campos simplesmente conta de 0 a 1, indicando se o campo é par ou ímpar. Os contadores são coordenados pelos sinais de sincronização horizontal e vertical. Sempre que um pulso de sincronização horizontal ocorre, o contador de *pixels* é zerado. O pulso de sincronização horizontal também é usado para incrementar o contador de linhas, indicando que o sinal de vídeo moveu-se uma linha abaixo. Sempre que um pulso de sincronização vertical ocorre, o contador de linhas é zerado, significando que o sinal de vídeo está na parte superior do quadro da imagem. O pulso de sincronização vertical também incrementa o contador de campos. Toda vez que o contador de campos conta de 0 para 1 e volta para 0, implica na aquisição de uma nova imagem (quadro) composta de dois campos.

Algumas câmeras de propósito específico, tanto de baixas como de altas velocidades, ou câmeras de varredura de linha, não transportam sinais de sincronização nos seus sinais de vídeo, como prescrito pelo padrão $RS-170$. Nesses casos, linhas separadas de sincronização com temporizações fora de padrão são freqüentemente usadas para acoplar a câmera ao circuito de digitalização. Desta forma, o digitalizador pode permanecer perfeitamente sincronizado com a câmera.

3.6.3 Visão Computacional, Processamento de Imagens e Computação Gráfica

A diferença entre visão computacional, processamento de imagens e computação gráfica está na forma de representação dos dados de entrada e de saída, como pode ser visto na figura 3.8.

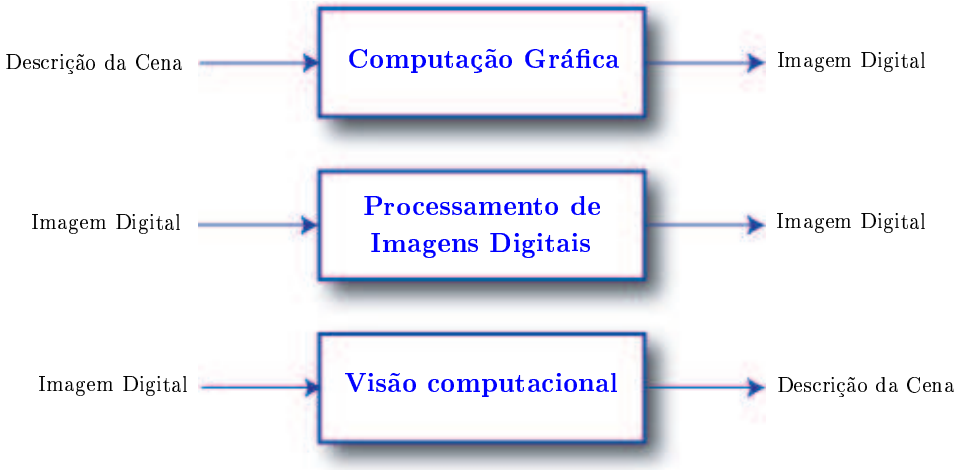

Figura 3.8: Relações entre visão computacional, processamento de imagens digitais e computação gráfica.

A entrada de um sistema de computação gráfica consiste na síntese de imagens, ou seja, na transformação de uma lista de atributos que descreve a cena numa imagem digital. Como tal, um sistema de computação gráfica simula a formação da imagem. Na visão computacional, o ponto de partida é uma imagem capturada de uma cena existente. O propósito é obter informações sobre a cena. O processamento de imagens age sobre imagens e resulta em outras imagens, isto é, transforma uma imagem digital em outra. O processamento de imagens pode ser classificado como: codificação, melhoramento, restauração e extração de suas características.

A codificação é útil para o armazenamento e transporte de imagens. O objetivo do melhoramento de imagens é facilitar por exemplo a visão humana, melhorando características específicas das imagens; este é o caso de certos sistemas de diagnóstico médico, imagens de raios-X apresentadas em monitores. De modo a adaptar esta exibição para a percepção humana, aplica-se manipulações de contraste e filtros às imagens. A restauração é uma operação que tenta corrigir distorções originadas no sistema de formação das imagens, tais como borramento e ruído. O

objetivo da extração de características é transformar a imagem em outra, onde certas medições possam ser feitas.

3.6.4 Processamento de Imagens

As funções de processamento de imagens digitais dividem-se em duas categorias básicas: o processamento ponto-a-ponto (ou pontual) e o processamento de vizinhança.

O processamento ponto-a-ponto é uma das técnicas mais primitivas, embora importante, de processamento de imagens digitais. Usado inicialmente em operações de melhoramento de contraste, o processamento ponto-a-ponto é bastante simples. O nível de cinza (cor) de cada pixel na imagem de entrada é modificado para um novo valor, freqüentemente por uma operação matemática ou lógica, e colocado na imagem de saída na mesma localização espacial, conforme mostrado na figura 3.9.

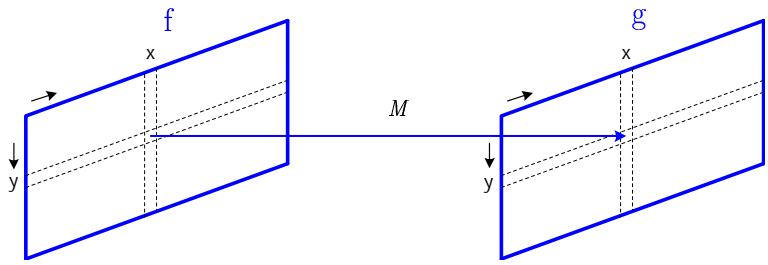

Figura 3.9: Processamento ponto-a-ponto.

As operações pontuais processam apenas atributos de brilho da imagem sem atuar sobre atributos espaciais e sua equação geral é dada por

$$g(x,y) = M[f(x,y)] \tag{3.3}$$

onde M é a função de mapeamento, que converte o nível de cinza da entrada para o nível de cinza da saída. Isto significa que o brilho de um *pixel* de saída nas coordenadas (x,y) é igual ao brilho do respectivo *pixel* de entrada, nas coordenadas (x,y), após ter sido mapeado pela função M. Está implícito que todos os *pixels* da imagem de entrada são mapeados, através da função de mapeamento, para a imagem de saída. Um exemplo de uma operação ponto a ponto é a limiarização (*thresholding*). A limiarização é uma das abordagens mais importantes para a segmentação de imagens. Nesta operação, o *pixel* $g(x,y)$ da imagem de saída recebe valores 0 ou 1 de acordo com a comparação do nível de cinza do *pixel* da imagem de entrada com um certo limiar,

$$O(x,y) = \begin{cases} 1 & Se \;\; I(x,y) > t \\ 0 & Caso\ contrário \end{cases} \tag{3.4}$$

Ou seja, valores '1' pertencem ao(s) objeto(s) de interesse na imagem, e valores '0' pertencem ao fundo (*background*).

Exemplos de outras operações de processamento de imagens digitais que usam o processamento ponto-a-ponto são: equalização do histograma, complementação (negativo), correção do fator gama etc. As operações pontuais também podem ser implementadas em *hardware* utilizando-se tabelas de procura (*look-up tables*). Essas tabelas são implementadas com uso de memórias ROM ou RAM, por exemplo adotando-se os endereços correspondentes aos níveis de cinza da imagem de entrada, enquanto que os respectivos conteúdos correspondem aos níveis de cinza da imagem de saída.É importante notar que esse método é viável para imagens quantizadas por poucos *bits*, como é o caso de imagens em níveis quantizados em 8 *bits* ($2^8 = 256$ posições de memória). Já para imagens coloridas (24 *bits*), esse método pode tornar-se inviável ($2^{24} = 16777216$ posições de memória). No processamento de vizinhança, o valor do *pixel* (x, y) na imagem de saída depende do *pixel* (x, y) correspondente na imagem de entrada, bem como de uma vizinhança local a este último. Uma vizinhança local de um *pixel* de coordenadas (x, y) é definida como um conjunto de *pixels* numa subárea da imagem que engloba este determinado *pixel*. Geralmente, esta subárea é escolhida como sendo retangular com (x, y) no centro da área (van der Heijden, 1994). Suponha que o tamanho da vizinhança seja de $(2K + 1)x(2L + 1)$, então a vizinhança local $f(x, y)$ do *pixel* de coordenadas (x, y) na imagem f é

$$f_{n,m} = \begin{bmatrix} f_{n-K,m-L} & f_{n-K,m-L+1} & \cdots & f_{n-K,m+L} \\ f_{n-K+1,m-L} & & & \vdots \\ \vdots & & & \vdots \\ f_{n+K,m+L} & \cdots & \cdots & f_{n+K,m+L} \end{bmatrix} \tag{3.5}$$

Uma operação numa vizinhança local é uma operação na qual cada *pixel* de saída em (x, y) tem associado a ele um nível de cinza que depende da vizinhança local $f(x, y)$ correspondente da imagem de entrada, figura 3.10.

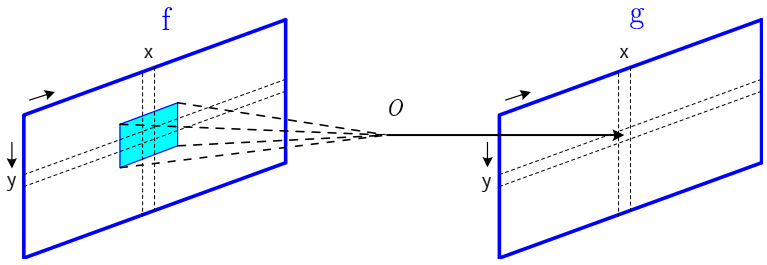

Figura 3.10: Processamento de vizinhança.

$$g(x, y) = O[f(x, y)] \tag{3.6}$$

As operações de vizinhança podem ser lineares ou não-lineares. Algumas aplicações do processamento de vizinhança são: detecção de bordas, detecção de cantos, filtragem, crescimento de regiões etc.

3.6.5 Conectividade e Rotulação

A conectividade entre *pixels* é um conceito importante usado no estabelecimento de bordas de objetos e componentes de regiões em uma imagem (Pratt, 1991).

Um *pixel* p nas coordenadas (x, y) possui quatro vizinhos horizontais e verticais, cujas coordenadas são dadas por

$$(x+1, y)(x-1, y)(x, y+1)(x, y-1)$$

Esse conjunto de *pixels* é, chamado *vizinhança-de-4* de p. Os quatro vizinhos diagonais de p possuem como coordenadas

$$(x+1, y+1)(x+1, y-1)(x-1, y+1)(x-1, y-1)$$

Esses pontos, juntos com a *vizinhança-de-4*, são chamados de *vizinhança-de-8* de p.

Para estabelecer se dois *pixels* estão conectados, é preciso determinar se eles são de alguma forma adjacentes, por exemplo *vizinhos-de-4*, e se seus níveis de cinza (ou cores) satisfazem algum critério de similaridade, por exemplo iguais.

Considerando-se que uma imagem seja percorrida *pixel* por *pixel*, da esquerda para a direita e de cima para baixo, e assumindo-se, por enquanto, que deseja-se determinar os componentes *conectados-de-4*. Seja p o *pixel* em qualquer passo no processo de varredura, e sejam r e t, respectivamente os vizinhos superior e esquerdo de p. A natureza da seqüência de varredura garante, que pontos anteriores a p, no caso r e t, já foram encontrados e rotulados, no caso de apresentarem valor 1. Um algoritmo clássico para a rotulação de imagens pode ser encontrado em (Gonzalez e Woods, 2000).

Neste algoritmo, se o valor de p for 0, p vai para a próxima posição. Se o valor de p for 1, examina-se r e t. Se ambos forem zero, um novo rótulo é atribuído a p. Se apenas um dos vizinhos for 1, p recebe o rótulo deste vizinho. Se ambos forem 1 e possuem o mesmo rótulo, este rótulo é atribuído a p. Se ambos r e t forem 1, mas possuírem rótulos diferentes, um dos rótulos é atribuído a p e define-se que os rótulos de r e t são equivalentes (isto é, os pontos r e t estão conectados por p). Ao final da varredura todos os pontos com valor 1 terão sido rotulados, mas alguns destes rótulos poderão ser equivalentes. Então, os pares de rótulos equivalentes são ordenados em classes de equivalência. O procedimento continua atribuindo-se um rótulo diferente a cada classe e então percorre-se a imagem novamente, trocando-se cada rótulo pelo rótulo pré-estabelecido da classe de equivalência criada e correspondente.

A rotulação de componentes *conectados-de-8* é feita de modo semelhante, mas os dois vizinhos diagonais superiores de p, denotados por q e s, também são levados em consideração.

3.7 Propriedades de Regiões na Imagem

O passo seguinte à segmentação das imagens é a caracterização de regiões. A descrição de regiões inclui aspectos como: forma, tamanho, posição, orientação etc. Freqüentemente, é desejável que o parâmetro que descreve um aspecto não seja afetado por outros aspectos. Por exemplo, os parâmetros de forma devem ser invariantes à posição, escala e orientação.

Geralmente, uma região consiste de um número de componentes conexos. Para extrair as características de componentes individuais é necessário ter um algoritmo que identifique cada componente. A maneira clássica de se extrair os componentes de uma imagem é através da rotulação de componentes. A determinação da conectividade entre *pixels* numa imagem binária é um passo fundamental na segmentação de objetos e regiões (*blobs*) numa imagem. Todos os *pixels* dentro de uma mesma região recebem o mesmo rótulo. Esse procedimento pode ser usado para estabelecer as bordas de um objeto, os componentes de uma região, o número de objetos, o centróide de objetos, etc (Gonzalez e Woods, 2000). Exemplos de aplicações podem ser: inspeção automática, reconhecimento de caracteres, visão robótica, controle por visão, futebol de robôs etc (Costa *et al.*, 1999; Hutchison *et al.*, 1996).

O primeiro algoritmo de rotulação é creditado a Rosenfeld e Pfaltz (1966). Este algoritmo realiza duas varreduras na imagem. Na primeira, a imagem é processada da esquerda para a direita e de cima para baixo, de modo a gerar rótulos para cada *pixel*. Durante esse processo, um mesmo objeto pode receber mais de um rótulo. Tal característica impõe a necessidade de marcar-se os rótulos equivalentes, ou seja, marcar os rótulos diferentes que pertencem a um mesmo objeto. As relações de equivalência são expressadas através de uma matriz binária. A partir dessa matriz determina-se as classes de equivalências e os novos rótulos para cada classe (Gonzalez e Woods, 2000). Na segunda varredura, cada rótulo é substituído por um rótulo associado a sua classe de equivalências. Uma desvantagem desse método é que a tabela de equivalências pode se tornar inaceitavelmente grande à medida que o tamanho da imagem aumenta. Haralick e Shapiro (1992) descrevem algumas variantes do algoritmo clássico que requerem menos esforço computacional e recursos de memória.

3.7.1 Parâmetros das Regiões

A geometria de uma região planar diz respeito a aspectos como: tamanho, posição, orientação e forma. Muitos desses aspectos são cobertos por uma família de parâmetros denominados "momentos". Na teoria de probabilidades, os momentos são usados para caracterizar funções de densidade de probabilidade, isto é, esperança (momento de primeira ordem), variância, covariância (momentos centrais de segunda ordem). No contexto atual, essas mesmas definições são usadas, mas a função de densidade é substituída por uma função binária bi-dimensional com nível '1' dentro da região e nível '0' fora. Os momentos de ordem $p+q$ de uma região representada pelo mapa de *bits* $b_{n,m}$ são:

$$M_{p,q} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} n^p m^q \, b_{n,m} \quad = \sum_{n,m \in \, regi\tilde{a}o} n^p m^q \tag{3.7}$$

Uma definição na qual $b_{n,m}$ é substituído por uma imagem em nível de cinza $f_{n,m}$ também existe. Entretanto, os momentos correspondentes não são mais parâmetros geométricos. O momento de ordem zero de uma função de densidade é o volume coberto por esta função. Na definição de mapa de *bits*, os valores da função são zero e um. Isso faz com que o volume coincida com a área da região. De fato, $M_{0,0}$ é o número de *pixels* dentro da região.

Os momentos de primeira ordem $M_{0,1}$ e $M_{1,0}$ estão relacionados ao ponto de equilíbrio (xb, yb) da região

$$\bar{x} = M_{1,0}/M_{0,0} \qquad \bar{y} = M_{0,1}/M_{0,0} \tag{3.8}$$

Esses parâmetros são apropriados para determinar a posição da região dada em unidades de *pixels* (Δ). O ponto (xb, yb) é chamado de centro de gravidade ou centróide.

De maneira a fazer as descrições independentes da posição, os momentos podem ser calculados com respeito ao centróide. Os resultados são chamados de momentos centrais:

$$\mu_{p,q} = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1}(n-\bar{x})^p(m-\bar{y})^q b_{n,m} \quad = \sum_{n,m\in região}(n-\bar{x})^p(m-\bar{y})^q \tag{3.9}$$

Se os momentos ordinários são conhecidos, os momentos centrais podem ser calculados mais facilmente do que diretamente a partir da equação (3.9). Por exemplo:

$$\begin{aligned}
&\mu_{0,0} = M_{0,0}\\
&\mu_{1,0} = \mu_{0,1} = 0\\
&\mu_{0,2} = M_{2,0} - \bar{x}M_{1,0}\\
&\mu_{1,1} = M_{1,1} - \bar{x}M_{0,1}\\
&etc.
\end{aligned} \tag{3.10}$$

Os momentos centrais de segunda ordem exibem um número de propriedades que são comparáveis com as matrizes de covariância da Teoria de Probabilidade e dos momentos de inércia da Mecânica. Os eixos principais de uma região são dados pelos auto-vetores da matriz:

$$\begin{bmatrix} \mu_{2,0} & \mu_{1,1} \\ \mu_{1,1} & \mu_{0,2} \end{bmatrix}$$

Os momentos principais são os auto-valores correspondentes.

$$\begin{aligned}
\lambda_{max} &= \tfrac{1}{2}(\mu_{2,0}+\mu_{0,2}) + \tfrac{1}{2}\sqrt{\mu_{2,0}^2+\mu_{0,2}^2-2\mu_{0,2}\mu_{2,0}+4\mu_{1,1}^2}\\
\lambda_{min} &= \tfrac{1}{2}(\mu_{2,0}+\mu_{0,2}) - \tfrac{1}{2}\sqrt{\mu_{2,0}^2+\mu_{0,2}^2-2\mu_{0,2}\mu_{2,0}+4\mu_{1,1}^2}
\end{aligned} \tag{3.11}$$

A direção do maior momento principal é

$$\theta = \tan^{-1}\left(\frac{\lambda_{max}-\mu_{2,0}}{\mu_{1,1}}\right) \tag{3.12}$$

θ é usado para especificar a orientação de uma região.

A "excentricidade"de uma região pode ser definida como a raiz quadrada da razão dos dois momentos principais:

$$excentricidade = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \tag{3.13}$$

Como $\mu_{0,0} = M_{0,0}$ é a área da região, ele pode ser usado como uma medida de tamanho. Então, para obter-se uma descrição independente do tamanho, normaliza-se os momentos em relação à área. Os momentos centrais normalizados podem ser calculados como:

$$\eta_{p,q} = \frac{\mu_{p,q}}{\mu_{0,0}^{\alpha}} \quad com: \quad \alpha = \frac{p+q}{2} + 1 \tag{3.14}$$

Uma característica essencial na análise de padrões é o reconhecimento de objetos e caracteres sem levar em consideração sues respectivos tamanhos, posições e orientações. Os momentos e funções de momentos têm sido intensivamente empregados como características globais invariantes de uma imagem no reconhecimento de padrões (Simon e Pawlak, 1996).

A partir dos momentos normalizados de ordem até três, sete parâmetros podem ser extraídos. Estes parâmetros são chamados de momentos invariantes:

$$\begin{aligned}
h_1 &= \eta_{2,0} + \eta_{0,2} \\
h_2 &= (\eta_{2,0} - \eta_{0,2})^2 + 4\eta_{1,1}^2 \\
h_3 &= (\eta_{3,0} - 3\eta_{1,2})^2 + (3\eta_{1,2} - \eta_{0,3})^2 \\
h_4 &= (\eta_{3,0} - \eta_{1,2})^2 + (\eta_{0,3} - \eta_{2,1})^2 \\
h_5 &= (\eta_{3,0} - 3\eta_{1,2})(\eta_{3,0} + \eta_{1,2})[(\eta_{3,0} + \eta_{1,2})^2 - 3(\eta_{0,3} + 3\eta_{2,1})^2] + \\
&(3\eta_{2,1} - \eta_{0,3})(\eta_{0,3} + \eta_{2,1})[3(\eta_{3,0} + 3\eta_{1,2})^2 - (\eta_{0,3} + \eta_{2,1})^2] \\
h_6 &= (\eta_{2,0} - \eta_{0,2})[(\eta_{3,0} + \eta_{1,2})^2 - (\eta_{0,3} + \eta_{2,1})^2] + 4\eta_{1,1}(\eta_{3,0} + \eta_{1,2})(\eta_{0,3} + \eta_{2,1}) \\
h_7 &= (3\eta_{2,1} - \eta_{0,3})(\eta_{3,0} + \eta_{1,2})[(\eta_{3,0} + \eta_{1,2})^2 - 3(\eta_{0,3} + \eta_{2,1})^2] - \\
&(\eta_{3,0} - 3\eta_{1,2})(\eta_{0,3} + \eta_{2,1})[3(\eta_{3,0} + \eta_{1,2})^2 - (\eta_{0,3} + \eta_{2,1})^2]
\end{aligned} \tag{3.15}$$

Estes parâmetros não dependem nem da orientação, nem do tamanho, e nem da posição dos objetos. A invariância desses descritores de forma foi mostrada por Hu (1962). Uma desvantagem de tais momentos invariantes é que é difícil atribuir uma interpretação geométrica a eles.

Capítulo 4

Sistema Proposto e Técnicas Aplicadas

Este capítulo apresenta os aspectos mais importantes das técnicas propostas e aplicadas no desenvolvimento do sistema de visão computacional proposto nesta tese. Inicialmente, são apresentados os procedimentos para a classificação de cores e determinação da posição dos múltiplos objetos, seguida pela apresentação da arquitetura de hardware desenvolvida.

4.1 Introdução

O objetivo deste trabalho foi o de desenvolver um sistema de visão computacional, em *hardware*, capaz de determinar a localização de diversos objetos numa seqüência de imagens por meio de suas cores. De modo a garantir a opeção em tempo real, isto é, 30 qps ou 60 cps, o sistema foi concebido de forma a tirar o máximo proveito das característica de varredura e temporização de sinais de vídeo padrão, tais como o $NTSC$ e o PAL. No projeto de *hardware* foi usado um dispositivo lógico programável (PLD) que, além do paralelismo inerente, oference grande quantidade de recursos lógicos e de memória.

Inicialmente, o sistema foi implementado em $MatLab$, onde sua funcionalidade foi testada exaustivamente. Em seguida, passou-se para o projeto dos circuitos usando como ferramenta de desenvolvimento o programa $QUARTUS$ da *Altera*. As técnicas empregadas neste trabalho podem ser aplicadas em diversos processos, destacando-se os sistemas robóticos, que serviram como paradigma para nortear seu desenvolvimento.

Nas seções subseqüentes são apresentados os procedimentos de classificação de cores e cálculos dos momentos dos objetos desenvolvidos nestes trabalho, bem como os aspectos de projeto da arquitetura de *hardware* que implementa tais procedimentos.

4.2 Classificação de Cores

Imagens coloridas têm sido usadas na visão computacional e no processamento de imagens para tarefas, tais como segmentação, reconhecimento e rastreamento. As cores oferecem várias vantagens significantes, em certas tarefas, sobre informações geométricas ou de níveis de cinza. Algumas dessas vantagens são: robustez sob oclusão parcial, rotação, mudanças de escala e mudanças de resolução (McKenna *et al.*, 1999; Wren *et al.*, 1997).

A segmentação de áreas coloridas que pertencem ao objeto é feita com base no modelo de cor do objeto. No primeiro caso, os *pixels* capturados são testados para determinar se eles estão dentro ou fora de um elipsóide que representa a cor do objeto buscado. No segundo caso, dados a estimativa de densidade e os modelos gaussianos de mistura, a probabilidade de um *pixel* pertencer ao objeto é calculada. Uma gaussiana é usada como uma função membro e, se a probabilidade for maior que um certo limiar, então o *pixel* capturado é considerado como pertencente ao objeto.

Neste trabalho foi implemntado um método capaz de classificar os *pixels* pertencentes a uma das cores de interesse ou pertencente ao fundo (*background*) da image. Para isto, foi utilizado uma variação do método de classificação apresentado por Saber *et al.* (1996). Neste método, os *pixels* da imagem de entrada têm sua representação de cores transformada do espaço RGB para o plano de crominância (ES) do espaço YES conforme a equação 4.1.

$$\begin{bmatrix} E \\ S \end{bmatrix} = \begin{bmatrix} 0,500 & -0,500 & 0,000 \\ 0,250 & 0,250 & -0,500 \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.1}$$

O espaço YES foi escolhido porque é linear e livre de singularidades, reduz a influência da intensidade luminosa e é computacionalmente eficiente, pois requer apenas deslocamentos de *bits* para a obtenção das componentes E e S. Cada cor é representada como uma classe no plano ES. A distribuição dessas classes (cores) no domínio ES se dá fazendo x_g representar um vetor composto das componentes E e S de um *pixel* na posição g. Então, a distribuição de x_g dentro de cada região é modelada por uma gaussiana bidimensional dada pela equação 4.2.

$$\begin{aligned} p\left(x_g \mid W_i\right) = \quad & (2\pi)^{-n/2} \mid K_i \mid^{-1/2} \\ & exp\left\{-\tfrac{1}{2}[x_g - m_i]^T K_i^{-1}[x_g - m_i]\right\} \end{aligned} \tag{4.2}$$

onde

$$x_g = [E_g \quad S_g]^T,\ m_i = [m_E \quad m_S]^T,\ e\ K_i = \begin{bmatrix} \sigma_E^2 & \sigma_{ES} \\ \sigma_{ES} & \sigma_s^2 \end{bmatrix} \tag{4.3}$$

e w_i, $i = 1, ..., N$, representa a classe de interesse.

Este modelo é baseado na consideração de que o vetor de crominância na posição g, que pertence à região i, pode ser representado por um vetor de média da crominância da classe i e por uma gaussiana residual de média zero.

O contorno das funções de densidade de probabilidade (uma para cada cor) na equação 4.2 define elipses, conforme equação 4.4, no domínio ES, cujos centros e eixos principais são determinados por m_i e K_i, $i = 1, ..., N$.

$$[x_g - m_i]^T K_i^{-1} [x_g - m_i] = \lambda_g^i \tag{4.4}$$

A equação 4.4 define um mapeamento escalar no domínio ES sobre N estatísticas escalares para cada *pixel* em g. O valor de λ_i, $i = 1, ..., N$, é proporcional à probabilidade do *pixel* pertencer à respectiva classe. Um pequeno valor numérico de λ implica em uma grande probabilidade do *pixel* pertencer à classe em questão. Dessa forma, a classificação dos *pixels* é realizada comparando-se os valores com limiares globais, t_i, pré-definidos para cada classe, conforme a equação 4.5:

$$pc_g = \begin{cases} 1 & Se\ \lambda_i \leq t_i \\ 0 & Senão \end{cases} \tag{4.5}$$

As figuras 4.1, 4.2 e 4.3 apresentam alguns resultados, obtidos via simulação no *Matlab*, do método de classificação usado. A título de ilustração, nestas figuras foram colocados também os espectros dos canais R, G e B, assim como os respectivos histogramas da intensidade luminosa das imagens de entrada. Como as cores aparentes sofrem modificações em função de rotações, translações, mudanças de escala, variações na intensidade luminosa etc (Rasmussen *et al.*, 1996), escolheu-se três imagens em diferentes posições, rotações e níveis de luminosidade, de modo a comprovar a eficiência do método de classificação implementado.

Os parâmetros utilizados na classificação da cor das figuras 4.1,4.2 e 4.3 foram os seguintes:

$$[m_E \ \ m_S] = [10 \ \ 19] \qquad K^{-1} = \begin{bmatrix} 0.1941 & -0.0953 \\ -0.0953 & 0.0990 \end{bmatrix} \qquad t = 8,25$$

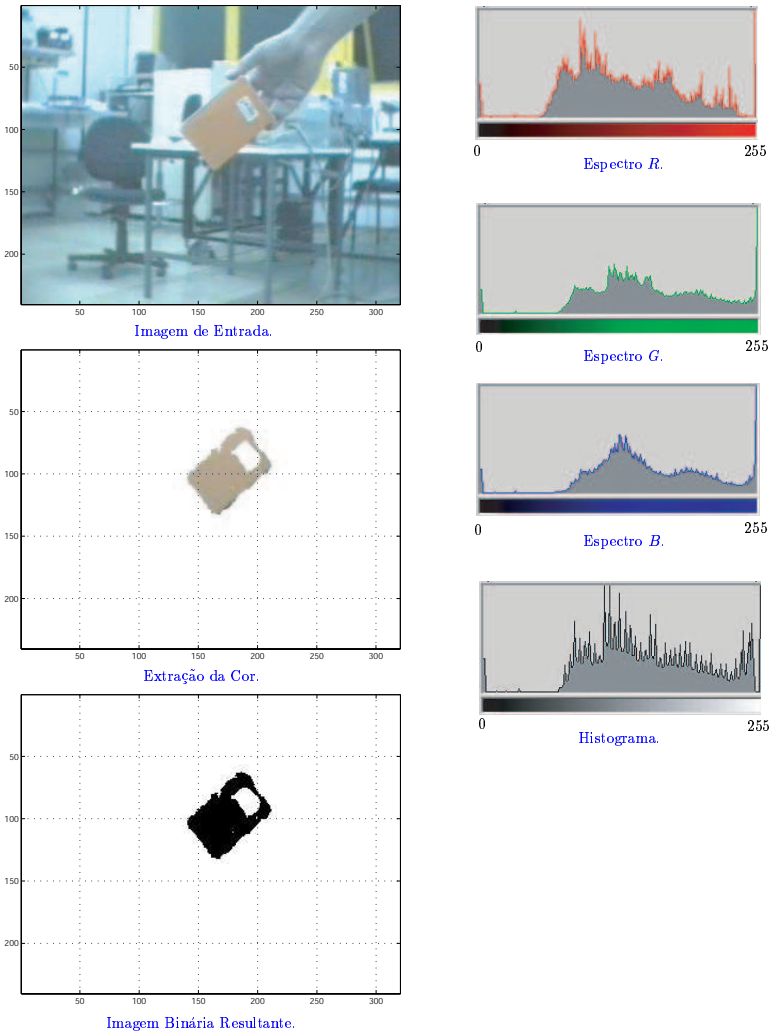

Figura 4.1: Aplicação do método de classificação de cores.

Com os *pixels* da imagem já classificados, deve-se determinar a localização dos objetos segmentados. Para tal, analísa-se a conectividade dos *pixels* para determinar a qual objeto um determinado *pixel* pertence. Cada objeto na imagem recebe um ou mais rótulos (*labels*) que o diferencia ou assemelha aos demais, possibilitando assim, a determinação de características, tais como centróide e área daquele objeto. A seção a seguir apresenta um método diferenciado para realizar a rotulação de objetos em uma imagem.

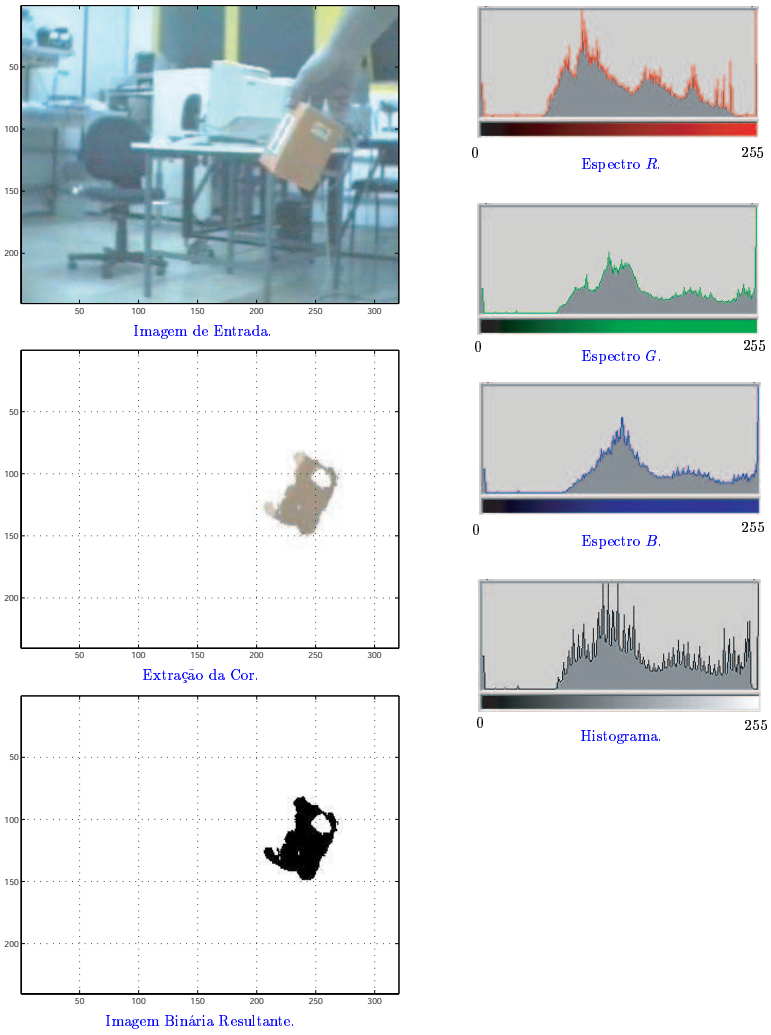

Figura 4.2: Aplicação do método de classificação de cores.

4.3 O Algoritmo Proposto para a Rotulação e o Cálculo dos Centróides

O algoritmo de rotulação desenvolvido nesta tese é uma variação daquele desenvolvido por Rosenfeld e Pfaltz (1966), com a diferença de que as equivalências são armazenadas numa fila ao invés de numa tabela de equivalências. Além disso, as equivalências são resolvidas no fim de cada fila e não no fim da imagem como acontece no algoritmo clássico. Esta modificação causa uma sensível diminuição no tamanho da memória necessária para realizar o processamento. Os centróides dos objetos são calculados por meio dos momentos dos objetos (veja seção 3.7.1) fazendo apenas uma varredura na imagem. Diversos vetores são usados para armazenar os somatórios das coordenadas (x, y), das áreas dos objetos e dos rótulos equivalentes. Existe ainda um vetor de *flags* que marca os rótulos que já passaram pelo processo de equivalência.

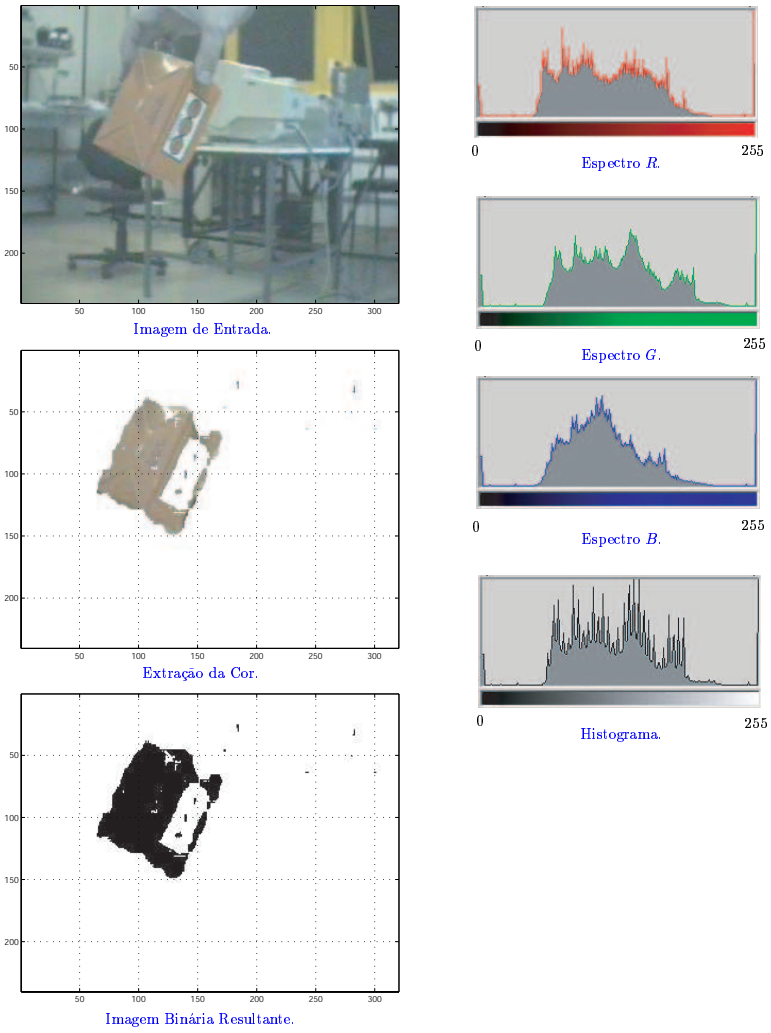

Figura 4.3: Aplicação do método de classificação de cores.

Considerando imagens binárias, $f_{m,n}$, onde os *pixels* pertencentes às regiões (objetos) de interesse assumem valor '1', enquanto os *pixels* do fundo (*background*) assumem valor '0', e também que as imagens são varridas da esquerda para a direita e de cima para baixo; sendo p o pixel atual nas coordenadas (x, y), o processo de rotulação e determinação dos centróides pode ser descrito pelo pseudo-código apresentado no algoritmo 1.

Quando o valor de p for 1, a função ***ATRIBROT*** atribui um rótulo, rp, a p, levando em consideração os valores e rótulos de seus vizinhos (q, r, s e t), já processados, dispostos conforme o mapeamento de vizinhança mostrado na figura 4.4.

Além de atribuir um rótulo ao *pixel* atual, a função ***ATRIBROT*** gera pares de rótulos equivalentes que poderão ser colocados numa fila de espera para serem processados quando a varredura da linha terminar. De maneira a fazer com que o sistema possa calcular os centróides

Algoritmo 1 Pseudo-código do algoritmo proposto.

```
Define NumMaxRot; {Número máximo de rótulos}
Define TamIm; {Tamanho da imagem}
Declare Variable X, Y, p, q, r, s, t, rp, rq, rr, rs, rt, Bvert, Bhoriz, rmaior, rmenor;
Declare Global Variable rotulo;
Declare Global Vector AuxRot(NumMaxRot);
Declare Vector cX(NumMaxRot), cY(NumMaxRot), RotEquiv(NumMaxRot), Flags(NumMaxRot);
Declare Matrix ImRot(TamIm); {Imagem rotulada}
Declare Global Matrix FilaEsp(2, NumMaxRot);
Declare Function ATRIBROT, REQUIV, CENTROIDES;
X := 0; Y := 0; rotulo := 1; AuxRot := 0; RotEquiv := 0; Flags := 1; ImRot := 0
while (Bvert = 0) do
  while (Bhoriz = 0) do
    if (p = 1) then
      ... obtém vizinhança de p;
      rp := ATRIBROT(q, r, s, t, rq, rr, rs, rt); {Atribui um rótulo a p e gera a fila de espera}
      ImRot(X, Y) := rp;
      sX(rp) := sX(rp) + X; {Somatório das coordenadas X}
      sY(rp) := sY(rp) + Y; {Somatório das coordenadas Y}
      sA(rp) := sA(rp) + 1; {Cálculo da área}
    end if;
    X := X + 1;
  end while;
  while (FilaEsp not empty) do
    [rmenor, rmaior] := REQUIV(RotEquiv); {Realiza equivalências}
    if (Rmenor ≠ Rmaior) then
      sX(rmaior) := sX(rmaior) + sX(rmenor);
      sY(rmaior) := sY(rmaior) + sY(rmenor);
      sA(rmaior) := sA(rmaior) + sA(rmenor);
      Flags(rmenor) := 0; rotEquiv(rmenor) := rmaior;
    end if;
  end while;
  Y := Y + 1; X := 0;
end while;

[cX, cY] := CENTROIDES(sX, sY, sA, Flags);
```

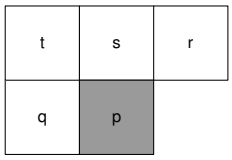

Figura 4.4: Vizinhança de p usada no processo de rotulação.

dos objetos presentes na imagem corretamente, os rótulos que já sofreram equivalência não devem mais ser atribuídos a nenhum outro *pixel*. Para isso, o algoritmo proposto faz uso de um vetor auxiliar, denominado ***AuxRot***, cuja função é garantir que um determinado objeto não receba um rótulo que já sofreu equivalência e, também, evitar que pares de rótulos sejam colocados mais de uma vez na fila de espera.

Após a rotulação de p, a área, sA do rótulo atribuído a p é incrementada de uma unidade. Da mesma forma, as coordenadas x e y são adicionadas aos somatórios sX e sY correspondentes àquele rótulo. Quando ocorre um fim de linha, a função ***REQUIV*** processa os pares de rótulos presentes na fila de espera (***FilaEsp***) de modo a gerar os rótulos equivalentes corretos para serem usados no processo de equivalência. Este processamento baséia-se no conteúdo do vetor ***RotEquiv***, que armazena todos os pares de rótulos que já sofreram equivalência na imagem. A equivalência se dá somente se os valores de *rmaior* e *rmenor* forem diferentes, e dá-se pela soma dos conteúdos de sX, sY e sA nas posições correspondentes aos rótulos do par de equivalência. O resultado desta soma é, então, atribuido à respectiva posição dos vetores sX, sY e sA correspondente ao rótulo

de maior valor do par de equivalências (*rmaior*). Em seguida, o vetor ***RotEquiv*** recebe o valor de *rmaior* na posição *rmenor*. O mesmo acontece com o vetor ***Flags***, cuja posição correspondente a *rmenor* recebe o valor 0. Se *rmaior* e *rmenor* forem iguais, nenhuma ação será realizada pela função ***REQUIV***, pois a igualdade significa que os rótulos do par de equivalência que originou *rmaior* e *rmenor* já sofreram o processo de equivalência entre si. Finalmente, quando a imagem termina, a função ***CENTROIDES*** faz o cálculos dos centróides dos objetos realizando a divizão de sX e sY por sA nas posições onde ***Flags*** for um e sA maior que zero.

O procedimento de atribuição de rótulos, representado por ***ATRIBROT***, para cada *pixel* na imagem, poder ser sintetizado pelo pseudo-código apresentado no algoritmo 2.

Algoritmo 2 Pseudo-código da função *ATRIBROT*.

```
Function rp := ATRIBROT(q,r,s,t,rq,rr,rs,rt)
  Declare Global Variable rotulo;
  Declare Variable c1, c2;
  Declare Global Vector AuxRot(NumMaxRot);
  Declare Global Matrix FilaEsp;
begin function
if (p não tem vizinhos) then
  c1 := rotulo;
  c2 := rotulo;
  rotulo := rotulo + 1;
else if (p tem apenas um vizinho) then
  c1 :="Rotulo deste vizinho"; {rq,rr,rs ou rt}
  c2 :="Rotulo deste vizinho";
else if (p tem mais de um vizinho) then
  c1 :="menor rotulo entre o 1º e o 2º processados mais recentemente, na ordem: rq,rr,rs,rt";
  c2 :="maior rotulo entre o 1º e o 2º processados mais recentemente, na ordem: rq,rr,rs,rt";
end if;
... ordena c1 e c2;
if (c1 ≠ c2) then
  if (c2 ≠ AuxRot(c1)) then
    FilaEsp(1,:) := [FilaEsp(1,:), c1];
    FilaEsp(2,:) := [FilaEsp(2,:), c2];
  end if;
  AuxRot(c1) := c2;
  rp := c2;
else
  if (AuxRot(c1)) then
    rp := AuxRot(c1);
  else
    rp := c2;
  end if;
end if;

end function;
```

Se p for igual a 1, a função ***ATRIBROT*** examina os valores e os rótulos dos *pixels* vizinhos de p. Caso p não possua vizinhos, isto é, q, r, s e t forem iguais a 0, um novo rótulo é atribuído a p . Neste caso, os elementos $c1$ e $c2$ do par de rótulos equivalentes receberão valores iguais ao rótulo atribuído a p. Havendo somente um vizinho, o rótulo desse vizinho será atribuído ao par de equivalência e também a p. Quando houver mais de um vizinho, p receberá o rótulo do primeiro vizinho na ordem: q, r, s, t. Já $c1$ e $c2$ receberão, respectivamente, os rótulos de menor e maior valor dos dois primeiros vizinhos de p na seqüência: q, r, s e t. Se os elementos do par de equivalências forem diferentes, e o conteúdo do vetor ***AuxRot*** na posição do menor elemento for diferente do rótulo de maior valor, o par de equivalências será colocado na fila de espera.

No fim de cada linha, caso haja algum par de endereços em ***FilaEsp***, a função ***REQUIV*** realiza a atualização desses rótulos para que os mesmos estejam aptos a sofrer o processo de equivalência. A necessidade dessa função se deve à característica inerente do processo de equivalência, na qual os conteúdos dos vetores sX, sY e sA, nas posições dos rótulos pertencentes ao par de

equivalência, são adicionadas e o resultado é atribuído aos respectivos vetores sX, sY e sA na posição correspondente ao rótulo de maior valor presente no par. Assim, o conteúdo correspondente ao rótulo de menor valor fica desatualizado e, portanto, não deve mais sofrer equivalência. Com exemplo, considere as imagens binária e rotulada mostradas na figura 4.5.

1	1	1	1	1	1	1
0	0	1	0	0	0	1
1	1	0	0	1	1	1

(a)

1	1	1	1	1	1	1
0	0	1	0	0	0	1
2	2	0	0	3	3	3

(b)

Figura 4.5: Imagens: (a) binária inicial e (b) rotulada resultante.

De acordo com o procedimento de rotulação apresentado anteriormente e considerando o primeiro rótulo atribuido como sendo 1, a imagem binária da figura 4.5a será rotulada conforme mostrado na figura 4.5b. Assim, no fim da terceira linha da imagem haverá dois pares, $(1,2)$ e $(1,3)$, de endereços equivalentes na fila de espera. Quando a equivalência for feita entre os rótulos 1 e 2, o rótulo 1 do par $(1,3)$ ficará desatualizado. Para que a equivalência ocorra de forma correta, o par $(1,3)$ deverá ser substituído pelo par $(2,3)$, resultando numa equivalência entre os rótulos 1, 2 e 3. Esse processo de atualização dos rótulos da fila de espera pode ser representado pelo pseudo-código do algoritmo 3.

Algoritmo 3 Pseudo-código da função $REQUIV$.

```
Function [fa, fb] := REQUIV(RotEquiv)
  Declare Variable aux1, aux2;
Declare Global Matrix FilaEsp(2, NumMaxRot);
begin function
fa := TiraFila(FilaEsp(1, :));
fb := TiraFila(FilaEsp(2, :));
aux1 := RotEquiv(fa);
auxb := RotEquiv(fb);
while (aux1) do
   fa := aux1;
   aux1 := RotEquiv(aux1);
end while;
while (aux2) do
   fb := aux2;
   aux2 := RotEquiv(aux2);
end while;
... ordena fa e fb;

end function;
```

Inicialmente, ***REQUIV*** lê o primeiro par de endereços que entrou em ***FilaEsp***, fazendo com que fa receba o valor do rótulo de menor valor e fb receba o rótulo de maior valor. Então, as variáveis locais $aux1$ e $aux2$ recebem o conteúdo do vetor ***RotEquiv*** nas posições de fa e fb respectivamente. A partir desse momento, enquanto $aux2$ for diferente de zero, fa receberá o valor de $aux1$ e $aux1$ receberá o conteúdo de ***RotEquiv*** na posição $aux1$. O mesmo procedimento acontece para $aux2$ e fb. A função do vetor ***RotEquiv*** é armazenar os valores dos pares de rótulos que já sofreram equivalência. As posições de ***RotEquiv*** correspondem aos endereços de valores menores nos pares, enquanto que seus respectivos conteúdos correspondem aos endereços de maior valor. Sempre que uma posição do vetor tem valor zero, significa que o rótulo correspondente àquela posição ainda não possui equivalente. É importante salientar que, como não há, obviamente, equivalência entre pares de rótulos iguais, o conteúdo (diferente de zero) de uma determinada

posição sempre apontará para outra posição de ***RotEquiv***. Esta é uma característica importante que é usada na função ***REQUIV***. Quando $aux1$ ou $aux2$ é diferente de zero significa que o rótulo representado por $aux1$ ou $aux2$ já sofreu equivalência com um outro rótulo de mais valor e, portanto, deve ser substituído pelo seu respectivo equivalente no par de equivalências.

Uma vez apresentado o procedimento proposto para a classificação de cores e para o cálculo dos centróides, as seções a seguir apresentam detalhes da implementação em *hardware* desses procedimentos.

4.4 Descrição dos Blocos Constituintes do Sistema

O projeto da arquitetura para extração de características da imagem foi concebido maneira modular, onde cada módulo (bloco) possui funcionalidade própria e pode ser modificado e/ou substituído por outro sem comprometer o funcionamento do restante do sistema. Quando necessário, o controle de cada bloco é feito por uma máquina de estados interna que sustenta seu funcionamento. Embora esta estratégia de controle possa resultar em máquinas de estado iguais para blocos diferentes, ela permite que se tire o máximo proveito da reconfigurabilidade dos dos dispositivos lógicos rogramáveis. O projetista pode adicionar ou remover funcionalidades (blocos) da arquitetura sem afetar o funcionamento do restante do sistema, incrementando assim, sua versatilidade.

O dispositivo escolhido para implementar os circuitos projetados nesta tese foi um PLD da família *Apex* $20K$ fabricado pela Altera *Corporation*. Essa família foi escolhida por apresentar grande capacidade lógica (mais de 1.000.000 de portas lógicas), altas freqüências de relógio e expressivos recursos de memória (Altera-Apex20K, 2001). Neste trabalho, a metodologia do projeto baseou-se na utilização de bibliotecas de funções fornecidas pelo fabricante do componente, no caso *library of parameterized modules* - LPM (Altera-LPM, 1996) da *Altera*, para a implementação das funções básicas e em uma HDL, no caso $VHDL$, para implementar os circuitos mais complexos, tanto combinacionais como seqüenciais.

No estágio atual do projeto, as entradas da arquitetura são consideradas como sendo:

- Freqüência de relógio (*clock*) do digitalizador;
- Formato de vídeo no padrão RGB;
- Sinais de sincronização horizontal e vertical.

Com essas idéias em mente, este trabalho realizou a implementação de algumas funcionalidades dessa arquitetura, que são:

- Sistema de geração de coordenadas de imagem;
- Sistema classificador de cores;

- Sistemas de geração de vizinhança e rotulação de componentes;
- Sistema de cálculo dos momentos de ordem zero e primeira ordem dos objetos na cena;
- Determinação da posição dos objetos.

Seguindo a premissa da modularidade, cada um desses blocos possui funcionalidade completa. Por isso, uma descrição, independente, de cada um deles é apresentada nas seções a seguir.

4.4.1 Gerador de Coordenadas

Este bloco faz a determinação das coordenadas x ($pixels$) e y (linhas) da imagem a partir dos sinais de sincronização horizontal ($Hsync$) e vertical ($Vsync$). O gerador de coordenadas é constituído por um contador de linhas, um contador de $pixels$ e uma máquina de estados, que juntamente com os sinais $Vsync$ e $Hsync$, controla o processo de determinação das coordenadas da imagem em questão, levando em consideração as características do padrão do sinal de vídeo usado. O contador de linhas é incrementado sempre que ocorre um pulso de sincronização horizontal, enquanto que o contador de pixels é incrementado a cada ocorrência de um pulso do relógio do digitalizador.

Como foi visto na seção 3.6.1, devido às características dos sinais de vídeo e aos tamanhos padrão (240x320, 480x640 etc) das imagens usadas na prática, um determinado número de linhas devem ser descartadas no início e no fim de cada campo de imagem, da mesma forma que um determinado número de $pixels$ deve ser desconsiderado no início e no fim de cada linha da imagem. Assim, levando em consideração as características dos sinais de vídeo, o controle do bloco de geração de coordenadas é realizado pela máquina de estados da figura 4.6.

Sempre que o sinal ***reset*** assumir o nível lógico '1', isto é, quando o sistema for reiniciado, o gerador de coordenadas passa para o estado ***inerte*** e fica esperando que o controle mestre envie um sinal de configuração para a arquitetura, então a máquima vai para o estado ***configuracao***. Este estado serve para que a arquitetura receba os parâmetros necessários para seu funcionamento. No contexto desta tese, esses parâmetros são os da(s) cor(es) de interesse: elementos da matriz de covariância inversa, as médias de E e S e o limiar. Mesmo após o término do processo de configuração, o gerador de coordenadas permanece no estado ***configuracao*** até que ocorra um pulso de sincronização vertical, indicando o início de um novo campo da imagem. Então, o gerador para o estado ***retracoc***, onde os contadores de linhas e de $pixels$ são zerados. Após o término do pulso de sincronização vertical, algumas linhas da imagem devem ser desconsideradas, pois não são linhas de vídeo válidas. Isso ocorre durante o estado ***nvideoc***, onde o contador de linhas é incrementado até chegar a um valor, ***lmin***, predeterminado. Quando o contador de linhas atinge o valor ***lmin***, o gerador passa para o estado ***nv2vc***, indicando que as linhas subseqüentes contêm informações de vídeo válidas e que o contador de linhas deve ser zerado. O estado ***nv2vc*** permanece ativo até o término do próximo sincronização horizontal, então, o estado do gerador de coordenadas passa a ser ***nvideol***. Analogamente ao que ocorre no início de cada campo de imagem, alguns pixels devem ser desconsiderados no início de cada linha, portanto, o estado ***nvideol*** dura até que que o contador de $pixels$ atinja o valor, ***pmin***, predeterminado. No qual, o estado passa a ser ***nv2vl***, que

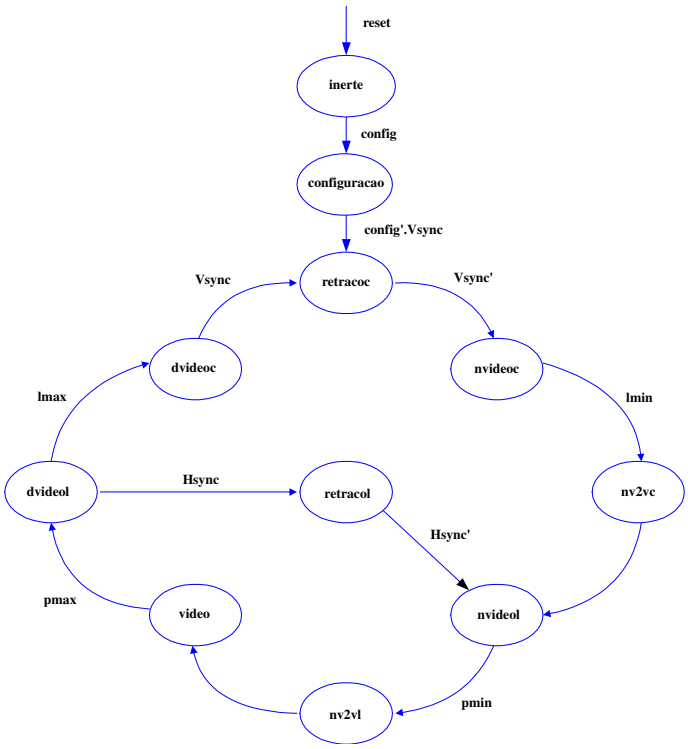

Figura 4.6: Máquina de estados do gerador de coordenadas.

indica que o contador de *pixel* deve ser zerado e que os pixels subseqüentes contém informações válidas. O estado ***nv2vl***, ocorre durante apenas um ciclo de relógio, passando logo em seguida para o estado ***vídeo***. Este estado é mantido até que o contador de *pixels* chegue no valor, ***pmax***, pré-estabelecido para a largura da imagem. Nesse momento, o estado passa a ser ***dvideol***, garantindo que os últimos *pixels* de cada linha não sejam levados em consideração. Caso ocorra um pulso de sincronização horizontal com a máquina de estados em ***dvideol*** e o contador de linhas não tenha atingido o valor, ***lmax***, para a altura da imagem, o estado passa a ser ***retracol***, correspondendo ao período de retraço horizontal do sinal de vídeo. Contrariamente, caso o contador de linhas chegue ao valor de ***lmax***, o estado passa a ser ***dvideoc***, que permanece até a ocorrência de um pulso de sincronização vertical. Quando isso ocorre, a máquina de estados assume o estado ***retracoc***, que corresponde ao período de retraço vertical do sinal de vídeo e indica o início de um novo campo de imagem.

O projeto do bloco gerador de coordenadas foi realizado considerando todos os períodos do sinal de vídeo que não carregam informação de vídeo válida como sendo períodos de *blanking*. Esses períodos podem ser o período de *blanking* horizontal, representado pelo sinal ***bhoriz***, e o período de *blanking* vertical, representado pelo sinal ***bvert***. O sinal ***bhoriz*** assume o nível lógico '1' nos estados: ***retracoc***, ***nvideoc***, ***nv2vc*** e ***dvideoc***. Já o sinal ***bvert*** assume o nível lógico '1' nos estados: ***nvideol***, ***nv2vl***, ***dvideol*** e ***retracol***. Durante o estado ***video***, o sinal de controle ***coord_val_p*** assume o nível lógico '1', indicando para os outros blocos da arquitetura que as informações processadas por eles naquele período correspondem a informações de vídeo válidas.

4.4.2 Sistema Classificador de Cores

O projeto da arquitetura para detectar características de objetos na imagem prevê a inclusão de um bloco que seja capaz de detectar uma ou mais cores pré-definidas. Inicialmente, é necessário transformar as componentes R, G e B do sinal de entrada para o plano de crominância ES, equação 4.1. A implementação dessa equação em *hardware* foi realizada de forma simples e direta, pois necessita apenas de 5 deslocadores de *bits*, dois somadores e um subtrator.

Para realizar a classificação de cores, o processo escolhido foi o definido pelas equações 4.4 e 4.5. A equação 4.4 realiza o cálculo da distância de *Mahalanobis* entre a posição do *pixel* em questão e a posição da média da cor desejada no plano ES. Se esta distância for menor que um determinado limiar, calculado *off-line*, e que foi fornecido previamente à arquitetura durante o processo de configuração, a equação 4.5 atribuirá o valor '1' para este *pixel*, caso contrário ele receberá o valor '0'.

Observando a equação 4.4, nota-se que os elementos da diagonal secundária da matriz de covariância inversa são iguais. Considerando este fato e expandindo a equação chega-se a

$$a^2 m_1 + abm_2 + b^2 m_3 = \lambda \tag{4.6}$$

onde

- $a = [E - m_E]$
- $b = [S - m_S]$
- m_E é a média da cor desejada no canal E
- m_S é a média da cor desejada no canal S
- m_1 e m_3 são os elementos da diagonal principal de K^{-1}
- m_2 é o dobro do elemento da diagonal secundária de K^{-1}

Foram realizadas duas implementações de *hardware* para a equação 4.6. A primeira foi o mapeamento direto dessa equação, usando três estágios de *pipeline*, com 6 multiplicadores e dois somadores. Os multiplicadores foram dispostos em 2 camadas com três deles em cada. A primeira camada faz os cálculo: a^2, ab e b^2, onde cada um deles possui entradas de 8 *bits* e saída de 16 *bits*. A segunda camada realiza os cálculos: a^2m_1, abm_2 e b^2m_3, com cada multiplicador possuindo uma entrada de 16 *bits*, resultados das multiplicações da camada anterior, e uma entrada de 12 *bits* (m_1, m_2, e m_3), resultando numa saída de 28 *bits*. Os dois somadores são de 28 *bits*, sendo que um deles faz a soma $a^2m_1 + abm_2$, e o outro adiciona o resultado desta soma com o produto b^2m_3, resultando no valor da distância de *Mahalanobis* e um sinal de estouro de faixa (*overflow*).

A segunda implementação da equação 4.6 visou a redução do número de elementos lógicos necessários para a implementação. Para isso, levou-se em consideração a característica de que os componentes da família *Apex 20K* podem trabalhar com freqüências de relógio diferentes. Os 6

multiplicadores necessários na primeira implementação foram substituídos por apenas 2, enquanto que os somadores foram reduzidos de 2 para 1. Nesta implementação, o cálculo da distância de *Mahalanobis* é feito numa freqüência três vezes maior que a freqüência do sistema de aquisição, com cada multiplicador realizando a sua respectiva operação em 3 ciclos de relógio. O somatório final também é realizado em três ciclos de relógio. A tabela 4.1 apresenta os ciclos de *pipeline* que ocorrem durante os cálculos até a obtenção do resultado final.

Est. 1			**Est. 2**			**Est. 3**		
a^2	ab	b^2	a^2	ab	b^2	a^2	ab	b^2
	a^2	ab	b^2	a^2	ab	b^2	a^2	ab
		a^2	ab	b^2	a^2	ab	b^2	a^2
			a^2m_1	abm_2	b^2m_3	a^2	ab	b^2
				a^2m_1	abm_2	b^2m_3	a^2m_1	abm_2
					a^2m_1	abm_2	b^2m_3	a^2m_3
						$a^2m_1 + 0$	$a^2m_1 + abm_2$	$a^2m_1 + abm_2 + b^2m_0$

Tabela 4.1: Estágios de *pipeline* da segunda implementação para o cálculo da distância de *Mahalanobis.*

É importante notar que a segunda implementação continua realizando cálculos nos mesmos três ciclos de relógio da primeira implementação. Essa segunda implementação conseguiu uma redução da ordem de 75% no número de blocos lógicos usados, mantendo uma freqüência de operação acima daquela estipulada como mínima ($15MHz$). Entretanto, a segunda implementação apresentou violações de tempo de estabelecimento (t_{SUP}) e tempo de permanência (t_{HD}) na simulação temporizada ($15MHz$), indicando a necessidade de modificações no projeto.

A representação dos valores envolvidos na classificação é feita por meio de uma representação em ponto fixo, conforme definido no apêndice B. A necessidade de uma representação desse tipo reside no fato de que, na maioria das vezes, a matriz de covariância inversa, K^{-1}, pode assumir valores fracionários e negativos. Sendo assim, para que se possa trabalhar com tais números sem que seja necessário construir multiplicadores e somadores em ponto-flutuante, é conveniente usar uma representação em ponto-fixo. Neste trabalho optou-se pela representação $A(2,9)$, que significa 1 *bit* de sinal, 2 *bits* para a parte inteira e 9 *bits* para a parte fracionária para os elementos da matriz. Tal representação proporciona uma resolução de $1/9^2 = 0,001953125$ para os elementos da matriz de covariância inversa. Os valores de R, G e B são representados por $U(8,0)$, enquanto que os valores de E, S, m_E e m_S são representados na forma $A(7,0)$, ou seja, 1 *bit* de sinal e 7 para a parte inteira.

A primeira camada de multiplicadores, ou a primeira multiplicação no caso da segunda implementação, produz resultados com representação $A(15,0)$. Já a segunda camada resulta numa representação $A(18,9)$. Assim, uma vez que o limiar é representado por $U(6,4)$, uma redução no tamanho da palavra resultante do cálculo da distância de *Mahalanobis* faz-se necessária. Essa redução é feita deslocando-se a palavra resultante do cálculo da distância de *Mahalanobis* 13 *bits* para a esquerda e desconsiderando-se os 18 *bits* menos significativos da palavra resultante. Feito isso, a comparação da distância entre a cor do *pixel* atual e a média da cor de interesse é realizada por um comparador (combinacional) de 10 *bits*.

Tanto o processo de soma dos produtos quanto o deslocamento de *bits* possuem sinais que indicam o estouro de faixa (*overflow*). A ocorrência de um estouro de palavra força o resultado da comparação assumir o valor zero, significando que o valor da distância de *Mahalanobis* é maior que o limiar pré-estabelecido, e portanto aquele *pixel* em questão não será considerado como pertencente à cor de interesse.

4.4.3 O Bloco Gerador de Vizinhanças

No algoritmo 1 foi visto a necessidade de se armazenar uma determinada quantidade de informações das imagens original e rotulada para que se possa gerar a vizinhança de um determinado *pixel* p no momento de seu processamento. A vizinhança usada neste trabalho, figura **??**, requer que o sistema tenha acesso aos vizinhos à esquerda e superiores de p e o seu processo de obtenção é mostrado na figura 4.7.

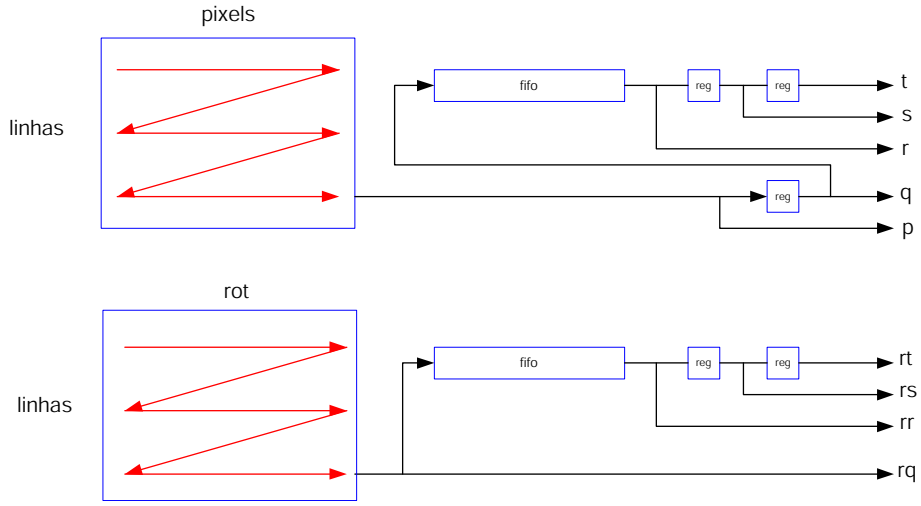

Figura 4.7: Processo de geração de vizinhanças.

O gerador de vizinhança recebe os *pixels* da imagem, e também os rótulos provenientes do sistema de rotulação, de maneira serial e, a cada ciclo de relógio do sistema de aquisição, fornece os valores dos vizinho (q, r, s e t) de p, bem como os seus respectivos rótulos (rq, rr, rs e rt). A implementação do gerador de vizinhanças usa filas *first in first out* ($FIFO$) e registradores para armazenar os *pixels*, e seus respectivos rótulos, necessários para a geração da vizinhança de p, figura A.4 no Anexo A. Quando ocorre uma nova imagem, isto é, o sinal ***bvert*** assume o nível lógico '1', as filas e os registradores são zerados. Assim, quando, por exemplo, o *pixel* p for pertencente à primeira linha da imagem, os seus vizinhos superiores serão necessariamente iguais a zero e, portanto, serão desconsiderados no processamento. Quando ocorre uma mudança de linha, o sinal ***bhoriz*** assume o nível lógico '1', as filas e os registradores são desabilitados, fazendo com que o sistema pare de armazenar os *pixels* e os rótulos. Entretanto, de maneira a se lidar com as bordas horizontais da imagem, as filas e registradores são habilitados um *pixel* antes do início da linha e permanecem habilitados um *pixel* depois do término da linha de vídeo. O sinal que habilita as

filas e os registradores é o ***coord_val***. Como visto na seção 4.4.3, p só assume o valor '1' durante o período de vídeo ativo, quando ***coord_val_p*** assume o valor '1'. Dessa forma, os vizinhos q e t do primeiro *pixel* da linha, assim como o vizinho r do último *pixel*, são necessariamente zero. Isto faz com que eles sejam desconsiderados no processo de rotulação. O procedimento descrito acima reproduz o processo de preenchimento das bordas da imagem com zeros.

4.4.4 Sistema de Rotulação e Cálculo dos Momentos

O bloco de cálculo dos momentos é uma das partes mais importantes no contexto deste trabalho, pois é nele que as operações mais complexas são realizadas. O circuito que realiza as os cálculos dos momentos está mostrado na figura A.5. O processo de atribuição de rótulos é realizado por um circuito combinacional implementado em $VHDL$, que realiza os passos descritos no algoritmo 2. Este circuito recebe, do gerador de vizinhanças, o *pixel* p e seus vizinhos, assim como seus respectivos rótulos. Como resultado, é gerado um par de rótulos equivalentes que será processado pelo circuito responsável pelo cálculo dos momentos.

O bloco de cálculo dos momentos realiza a implementação em *hardware* do algoritmo 1. Este algoritmo foi desenvolvido especialmente para a implementação da arquitetura proposta. Como pode ser observado na seção 4.3, o algoritmo proposto faz uso intensivo de vetores e filas $FIFO$. Tal abordagem, se voltada para implementação em *software* poderia levar a uma diminuição muito acentuada do desempenho do sistema. Porém, quando o objetivo é a implementação em *hardware*, e mais especificamente usando dispositivos lógicos programáveis, com emprego de múltiplos vetores armazenados em memórias RAM distribuídas e filas $FIFO$, pode-se obter desempenhos muito superiores na realização das mesmas tarefas. Isto ocorre devido ao fato de que as memórias podem ser utilizadas de forma paralela, possibilitando que os resultados dos cálculos possam ser armazenados e recuperados de forma muito rápida e fácil.

Os vetores usados no algoritmo 1 para armazenar os resultados dos cálculos dos momentos, o vetor de *flags*, e o vetor de equivalências, são implementados como memórias RAM com registradores em todas as entradas, figura 4.8. Já o vetor auxiliar de equivalências é implementado por uma memória RAM com o barramentos de endereços assíncrono, figura 4.9. Isto é feito devido à necessidade de se ter acesso imediato aos valores armazenados nesta memória.

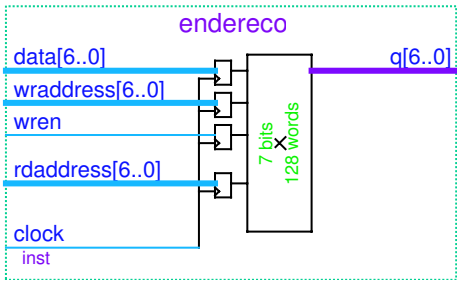

Figura 4.8: Memória com o barramento de leitura síncrono.

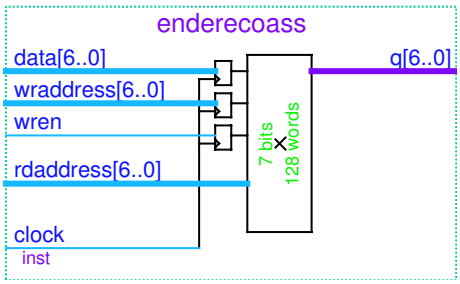

Figura 4.9: Memória com o barramento de leitura assíncronas.

Outra característica importante do sistema de rotulação e de cálculo de momentos, é que ele usa as características inerentes aos padrões de vídeo para aumentar a capacidade de processamento e minimizar o gasto dos recursos disponibilizados pelo dispositivo usado. Por exemplo, as equivalências de rótulos diferentes são realizadas no fim de cada linha e não no fim da imagem como ocorre com a maioria dos algoritmos de rotulação convencionais.

O sistema de rotulação implementado neste trabalho foi projetado de maneira que os cálculos necessários possam ser realizados paralelamente e que as características inerentes, como os tempos de retraço, dos sinais de vídeo pudessem ser usadas para diminuir o tempo de processamento e a quantidade de recursos lógicos e de memória usados na implementação da arquitetura. Como mencionado anteriormente, o mapeamento do algoritmo 1 foi feito de forma que os vetores se tornem memórias RAM e as filas, como filas $FIFO$ disponíveis na arquitetura $Apex\ 20K$. Já os laços (*loops*) são implementados como parte de uma máquina de estados que realiza o controle de todo o processamento que ocorre no bloco. Esta máquina de estados é mostrada na figura 4.10.

Toda vez que o sistema é reiniciado, a máquina de estados da figura 4.10 leva o sistema para o estado ***inerte***. Neste estado nenhuma ação é realizada, o sistema apenas aguarda pelo sinal de configuração para que a arquitetura receba os parâmetros necessários para o seu funcionamento. O estado ***configuracao*** permanece ativo até que ocorra um sinal de *blanking* vertical, indicando o início de uma nova imagem. Então a máquina de estados assume o estado ***resetm***, no qual todas as memórias da arquitetura são zeradas de modo a ficarem prontas para armazenarem os dados dos cálculos que serão realizado sobre o sinal de vídeo de entrada. Quando as memórias já foram totalmente varridas e zeradas, a máquina de estados passa para o estado ***fimrm*** e, em seguida, para o estado ***esp***. Neste último estado o sistema espera por uma seqüência de vídeo válida. Tão logo isto ocorra, o sistema assume o estado ***processamento***, onde os cálculos dos momentos são realizados.

No estado ***processamento***, os pares de rótulos provenientes da primeira etapa da rotulação são avaliados para saber se os rótulo são iguais e também se eles já sofreram equivalência entre si. Caso os rótulos sejam diferentes e o valor do rótulo maior seja diferente do conteúdo de ***AuxRot*** na posição do rótulo menor, o par de rótulos é colocado na fila de espera (***FilaEsp***), e o cálculo dos momentos é feito para o rótulo de maior valor. Isto é feito lendo-se os conteúdos das memórias RAM responsáveis pelo armazenamento da área e dos somatórios das coordenadas X e Y daquele rótulo. Tão logo os dados estejam disponíveis, no próximo ciclo de relógio, a área do

respectivo rótulo é incrementada de um e os momentos de primeira ordem de X e Y são adicionados, respectivamente, às coordenadas X e Y do *pixel* p em questão. Porém, se os rótulos forem iguais, nenhum rótulo é colocado na fila de equivalências. Quando isso acontece, o rótulo atual deve ser testado de maneira a garantir que ele ainda não tenha sofrido equivalência com um outro rótulo de maior valor. Isto é feito, verificando-se o conteúdo da memória vetor ***AuxRot***. Se esse valor for diferente de zero, o rótulo processado será aquele correspondente ao conteúdo da memória ***AuxRot*** na posição do rótulo analisado.

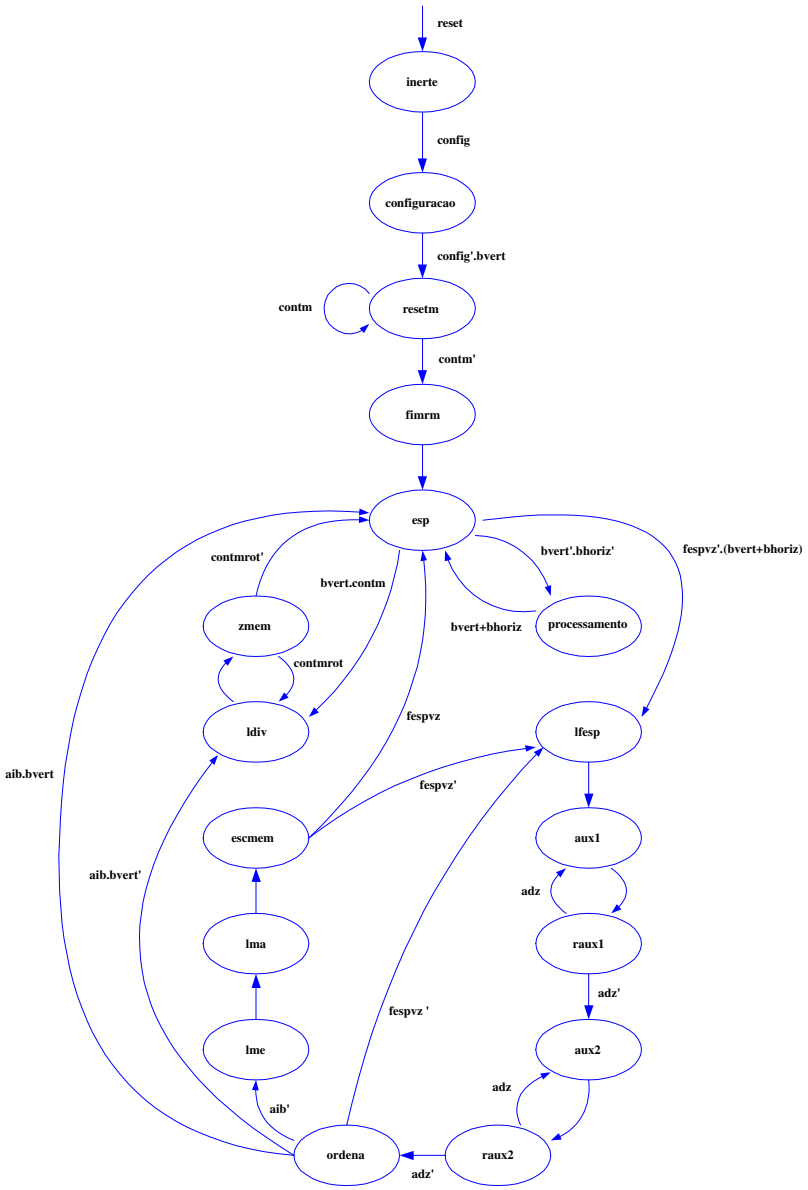

Figura 4.10: Máquina de estados do sistema de rotulação e cálculo de centróides

Quando a linha de vídeo termina, o sistema retorna para o estado ***esp***. Então, se houver algum par de endereços na ***FilaEsp***, a máquina passa para o estado ***lfesp***. Este estado dura

apenas um ciclo de relógio e tem finalidade de gerar um sinal de leitura para ***FilaEsp***. No estado seguinte, ***aux1***, o rótulo de menor valor do par é colocado no barramento de endereços da memória ***RotEquiv***. No próximo ciclo de relógio, já no estado ***raux1***, o conteúdo da memória correspondente ao menor rótulo é analisado. Se o conteúdo de ***RotEquiv*** for diferente de zero, o sistema retorna para o estado ***aux1*** com a diferença que, agora, o valor colocado no barramento de endereços é o conteúdo da memória usado no estado ***raux1***. Como pode-se observar no diagrama da figura 4.10, os estados ***aux1*** e **raux1** ficam num laço (*loop*) até que o conteúdo da posição da memória de equivalência assuma valor zero. Quando isso ocorre, esta posição passa a ser o novo rótulo do par de equivalências e o sistema vai para o estado ***aux2***. Os estados ***aux2*** e ***raux2*** realizam as mesmas operações que os estados ***aux1*** e ***raux1***, com a diferença de que o rótulo analisado é aquele de maior valor presente no par de equivalências. Estes últimos quatro estados descritos representam o mapeamento em *hardware* do algoritmo 3.

Após a atualização dos endereços equivalentes, o sistema passa, durante um ciclo de relógio, pelo estado ***ordena*** para que ocorra um re-ordenamento nos rótulos resultantes. Se os endereços do par resultante forem diferentes. O valor do rótulo de menor valor é colocado no barramento de endereços da memória dos momentos, no estado ***lme***. Em seguida, já no estado ***lma***, o valor dos momentos do rótulo de menor valor é armazenado num registrador e o rótulo de maior valor é colocado no barramento de endereços da memória de momentos. Desta forma, no estado ***escmem*** são obtidos os momentos correspondentes aos rótulos presentes no par de equivalências. Então, esses conteúdos são somados e o resultado é escrito na memória de momentos no endereço que corresponde ao rótulo de maior valor. Ao mesmo tempo, a memória ***Flags*** recebe o valor '1' na posição correspondente ao rótulo de menor valor, indicando que este rótulo já sofreu equivalência. Uma vez que o sistema esteja no estado ***escmem***, ele pode migrar para dois novos estados. Se ainda houver pares de endereços na fila de equivalências, a máquina de estados retorna ao estado ***lfesp***. Caso contrário, o novo estado será o ***esp***. Porém, se os rótulos atualizados forem iguais no estado ***ordena***, o processo de equivalência dos estados ***lme***, ***lma*** e ***escmem*** não ocorrerá. O sistema irá para o estado ***lfesp***, ***ldiv***, ou ***esp***, dependendo da ocorrência de um sinal de *blanking* vertical ou se existirem pares de rótulos equivalentes na fila de espera.

Nos estados ***ldiv*** e ***zmem***, as memórias de momentos e de $Flags$ são varridas de zero até o maior valor do rótulo usado no processamento do último quadro de imagem, sendo que no estado ***zmem*** o conteúdo do endereço que acabou de ser lido é zerado. O objetivo do estado ***zmem*** é, obviamente, o de zerar as memórias para preparar o processamento de um novo quadro de imagem. Os valores dos momentos lidos no estado ***ldiv*** são transferidos para o computador hospedeiro por meio do bloco de interfaceamento. Somente os valores cujos respectivos *flags* forem iguais a zero são considerados válidos nos cálculos posteriores.

4.5 Cálculo dos Centróides e Interface com o Computador Hospedeiro

Como visto na seção 3.7.1, a partir dos momentos da imagem, pode-se calcular várias características dos objetos presentes na cena. Entre estas características estão os centros geométricos ou centróides dos objetos que são obtidos a partir da divisão dos momentos de primeira ordem

(somatório das coordenadas x e y) pelo momento de ordem zero (área) dos objetos presentes na imagem.

Quando o sistema de cálculo dos momentos está no estado **ldiv**, os momentos correspondentes aos rótulos que não sofreram equivalência ($Flags = '1'$) são lidos da memória de momentos e, em sequida, armazenados numa fila denominada ***filadiv***para que, a partir deles, os centróide de cada objeto possa ser calculado. O objetivo desta fila é o de fazer o interfaceamento entre as diferentes freqüências de relógio (do sistema de cálculo dos momentos e do divisor) e, também, livrar a memória principal para o processamento do próximo quadro de imagem. O cálculo dos centróides (divisão) é realizado numa frequência de relógio inferior a frequência do restante do sistema, pois os divisores, implementados pelo bloco ***lpm_div***, não suportam operar na frequência requerida para o sistema. A outra finalidade da fila de divisão é armazenar os valores dos momentos a serem usados posteriormente na divisão, liberando a memória para ser usada no processamento da próxima imagem. A passagem dos centróides calculados para o computador hospedeiro é feita por meio de uma outra fila que faz a compatibilização entre as freqüências do microprocessador e dos divisores. Toda vez que um novo centróide é colocado na fila, um sinal de leitura de fila é gerado. Esse sinal também é usado como uma requisição de interrupção (IRQ) para o computador.

Capítulo 5

Resultados da Implementação

Visando a validação das partes do sistema de extração de características de imagens desenvolvidos neste trabalho, este capítulo apresenta os resultados de testes realizados nos circuitos implementados.

Inicialmente, os circuitos foram compilados no programa *Quartus II* $v1.0$ que realiza a síntese e faz análises de temporização nos circuitos. Nesta fase a única restrição imposta ao processo de síntese foi que o sistema fosse capaz de operar numa freqüência igual ou superior a 15 MHz. Esta freqüência foi importa para que o sistema pudesse ser capaz de operar em tempo real para sinais de vídeo do tipo $NTSC$ ou PAL que requerem freqüências de digitalização um pouco inferiores a esta.

Os circuitos foram implementados no dispositivo $EP20K200EFC484-2X$ da família *Apex* $20K$ da *Altera*. Este dispositivo foi usado, pois faz parte do KIT de desenvolvimento $NIOS$ (Altera-NIOS, 1996) disponível no laboratório de pesquisa onde este trabalho de tese foi desenvolvido. Outro fator que levou a escolha por este dispositivo é sua adequada quantidade de recursos para as finalidades a que o trabalho se propôs.

Como mencionado anteriormente a implementação dos circuitos projetados baseou-se nas seguintes premissas:

1. Circuitos usuais tais como somadores, multiplicadores, multiplexadores, comparadores etc; foram implementadas usando blocos predefinidos pelo fabricante (*library of parametrized modules* - LPM);
2. Circuitos combinacionais ou sequenciais mais completos e máquinas de estados foram projetadas em $VHDL$;
3. Circuitos de memórias RAM e filas $FIFO$ foram instanciadas por meio das funções LPM. Estas funções são mapeadas diretamente nos blocos $ESBs$ do dispositivo alvo (vide capítulo 2).

A tabela 5.1 apresenta alguns resultados da compilação dos blocos implementados.

Bloco	**Elementos Lógicos**	**Bits de Memória**	f_{max} MHz
Distância de *Mahalanobis* (1)	1997/24%	0	64,81
Distância de *Mahalanobis* (2)	664/7%	0	22,21
Gerador de Coordenadas	53/ < 1%	0	92,26
Gerador de Vizinhança	136/1%	4096/3%	80,37
Gerador dos Momento	551/6%	10224/9%	44,84
Cálculo dos Centróides	1299/15%	1312/1%	129,03 93,80 8,17

Tabela 5.1: Resultados da compilação dos blocos da arquitetura.

Note que embora as freqüências de operação obtidas durante o processo de compilação estejam muito acima daquela requerida inicialmente para o projeto, ainda é possível conseguir-se melhores resultados, utilizando dispositivos mais velozes. Após o processo de compilação, foram realizadas simulações funcionais e temporizadas nos circuitos. As simulações funcionais foram utilizadas para comprovar a funcionalidade dos circuitos projetados, enquanto que as simulações temporizadas tiveram o intuito de garantir que o circuito estivesse apto a operar segundo as restrições do dispositivo alvo.

O primeiro circuito projetado e simulado foi o que realiza o cálculo da distância de *Mahalanobis*, figura A.1. As simulações funcional e temporizada, para a primeira implementação, são apresentadas nas figuras 5.4 e 5.5 respectivamente. Os parâmetros utilizados para esta simulação foram os mesmos utilizados na seção 4.2. Como exemplo do resultado da simulação observe o instante em que as entradas E e S assumem os valores 6 e 7 respectivamente. Três ciclos de relógio a frente tem-se o resultado do cálculo da distância de *Mahalanobis* para estas entradas (081h = 8,06d).

O segundo circuito simulado foi o circuito do gerador de coordenadas e classificador, figura A.3 e cujas formas de onda de simulação estão mostradas na figura 5.6. Da mesma forma que no circuito anterior, esta simulação utilizou os parâmetros da sessão 4.2 para realizar a classificação dos *pixels* que tiveram seus valores RGB obtidos das figuras 4.1, 4.2 e 4.2. É importante observar que, como descrito na seção 4.4.1, alguns *pixels* da imagem de entrada foram desconsiderados no começo e no fim das linhas de vídeo.

A simulação do circuito gerador de vizinhanças, figura A.4, está apresentada na figura 5.7. Como entrada, foram fornecidos: a imagem binária mostrada na figura 5.1a e seus rótulos correspondentes, figura 5.1b. Como exemplo, observe o segundo *pixel* da segunda linha. Como esperado, a sua vizinhança é formada por $q = 1$, $r = 1$, $s = 0$, e $t = 0$; assim como seus respectivos rótulos vizinhos são $rq = 1$ e $rr = 0$.

Para a validação do circuito de cálculo dos momentos, figura A.5, foi usada a imagem mostrada na figura 5.2a, cujos rótulos correspondentes são mostrados na figura 5.2b. Na figura 5.2a, os centróides dos objetos presentes na imagem são dados pelos *pixels* destacados. Como ilustração,

0	0	1	1
1	1	0	0
0	0	0	0
0	1	1	0

(a)

x	x	0	0
1	1	x	x
x	x	x	x
x	2	2	x

(b)

Figura 5.1: (a) Imagem binária inicial e (b) Rótulos correspondentes.

0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
1	1	0	1	0	0	0	1	1	1	0	0	0	0	1
0	0	1	0	1	1	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	1	0	1	0	0	1
1	0	0	1	0	1	1	1	1	0	0	1	1	1	1

(a)

x	x	x	x	x	x	x	x	x	x	x	0	0	0	x
1	1	x	2	x	x	x	3	3	3	x	x	x	x	0
x	x	2	x	2	2	3	x	x	x	3	x	x	x	0
x	x	x	3	x	x	x	x	x	3	x	3	x	x	0
4	x	x	3	x	5	5	5	5	x	x	5	5	5	5

(b)

Figura 5.2: (a) Imagem binária inicial e (b) Rótulos correspondentes.

o processamento da terceira linha é apresentado na figura 5.3, onde os valores arnazenados nas memórias estão apresentados na rpresentação hexadecinal. Os 6 primeiros dígitos, da esquerda para a direira, representam o valor de sX, os próximos 6 representam o valor de sY e os 4 digitos restantes representam sA. Isto corresponde a uma memória de 64 *bits*.

Antes do início da linha, o conteúdo da memória de momentos era o apresentado na figura 5.3a. Após o término da linha e antes do processo de equivalência, o valor da memória de momentos apresentava-se como na figura 5.3b, com os pares de rótulos (1,3) e (2,3) presentes na fila de equivalências. Ainda durante o período de *blanking* horizontal, o sistema realiza as equivalências e retorna ao estado *esp* , com a memória de momentos apresentando os valores da figura 5.3c. Observando-se rapidamente a imagem de teste, pode-se facilmente conferir que os valores presentes na figura 5.3c correspondem exatamente ao processamento da imagem de teste até a terceira linha, com as devidas equivalências entre rótulos realizadas. A simulação funcional, correspondente ao processo de equivalência realizado na terceira linha da figura 5.2a, para o circuito é apresentada na figura 5.8 , enquanto que as figuras 5.9 e 5.10 apresentam, respectivamente os

resultados finais das simulações funcional e temporizada correspondente à figura 5.2 . Observe que os resultados da simulação temporizada são idênticos aos observados na simulação funcional.

Addr	+0	+1	+2	+3
0	0000320000010004	0000010000020002	0000030000010001	0000180000030003
4	0000000000000000	0000000000000000	0000000000000000	0000000000000000

(a) Conteúdo da memória da terceira linha - Começo.

Addr	+0	+1	+2	+3
0	0000320000010004	0000010000020002	00000E0000070004	0000280000070005
4	0000000000000000	0000000000000000	0000000000000000	0000000000000000

(b) Conteúdo da memória da terceira linha - Fim (antes da equivalência).

Addr	+0	+1	+2	+3
0	0000400000030005	0000010000020002	00000F0000090006	000037000010000B
4	0000000000000000	0000000000000000	0000000000000000	0000000000000000

(c) Conteúdo da memória da terceira linha - Fim (depois da equivalência).

Figura 5.3: Estados assumidos pela memória dos momentos.

Por fim, uma simulação para o circuito de cálculo dos centróides, figura A.6, é apresentada na figura 5.11. A partir dessa figura, pode-se constatar facilmente o correto funcionamento do circuito. Por exemplo, observe o segundo valor de entrada no circuito $M1x = 50$ $M1y = 120$, e $M0a = 30$. No segundo pulso de IRQ tem-se o resultado da divisão de $M1x$ e $M1y$ por $M0a$.

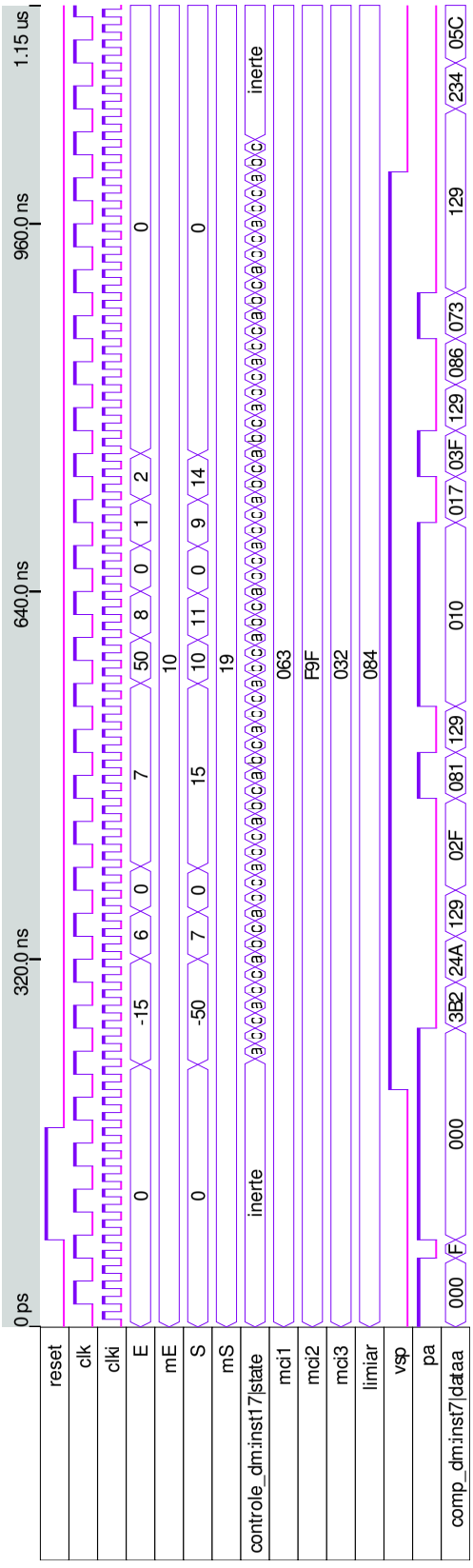

Figura 5.4: Formas de onda do cálculo da distância de *Mahalanobis* - Fun.

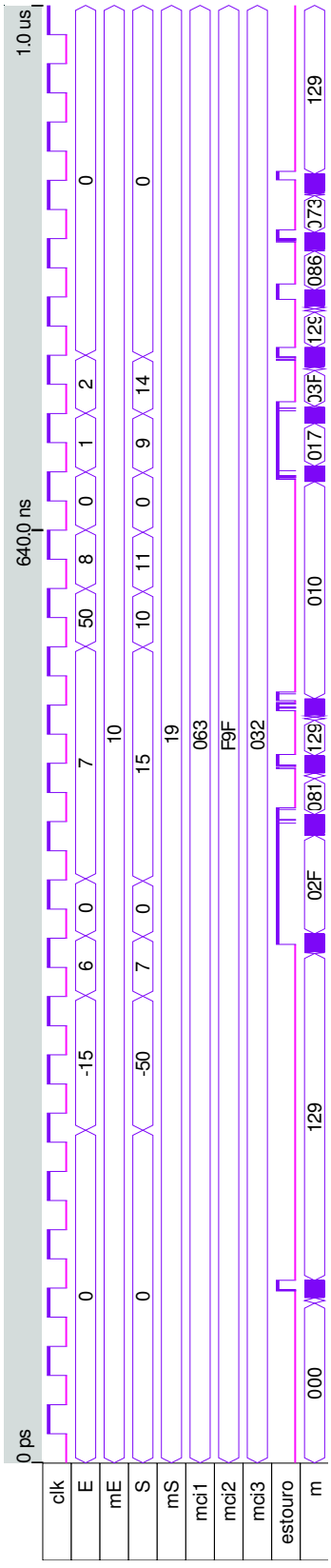

Figura 5.5: Formas de onda da simulação temporizada do cálculo da distância de *Mahalanobis*.

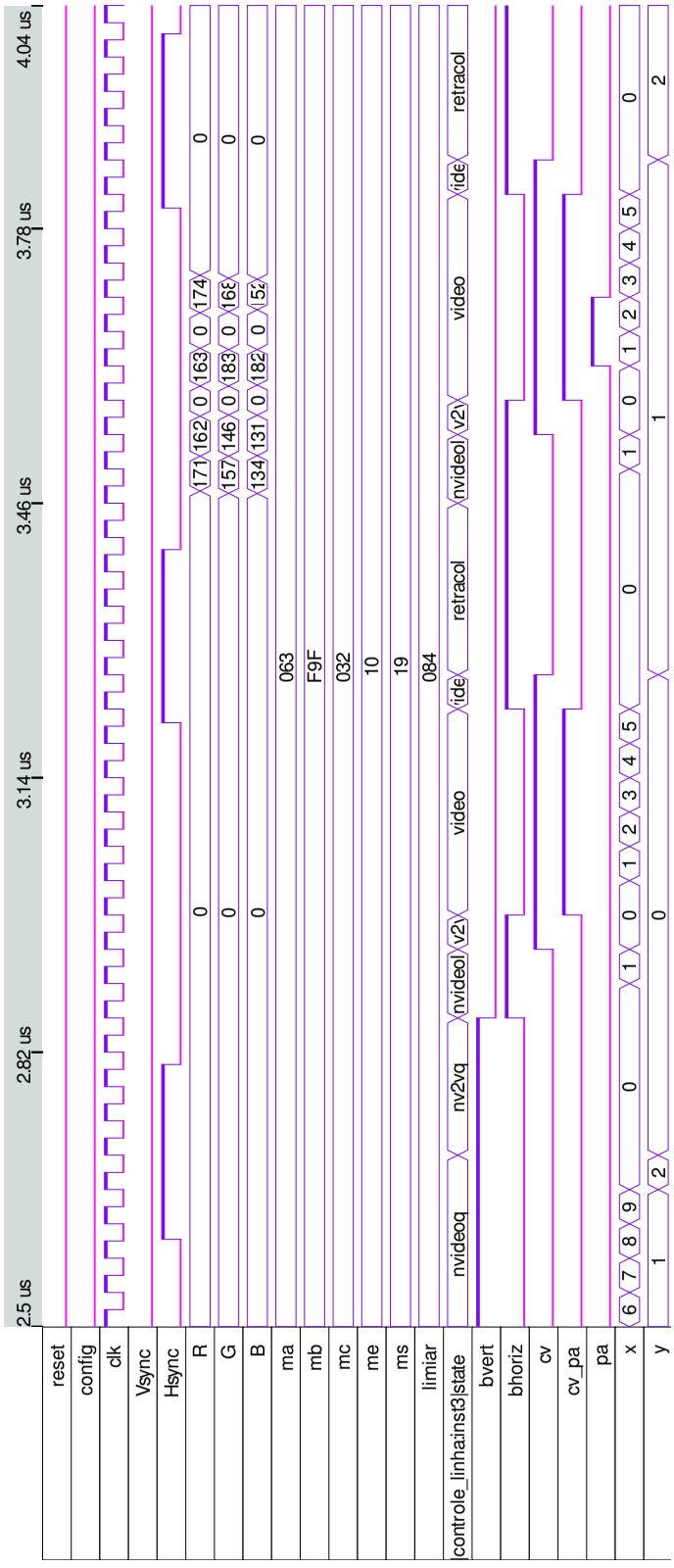

Figura 5.6: Formas de onda geradas pelo circuito classificador e gerador de coordenadas - Funcional.

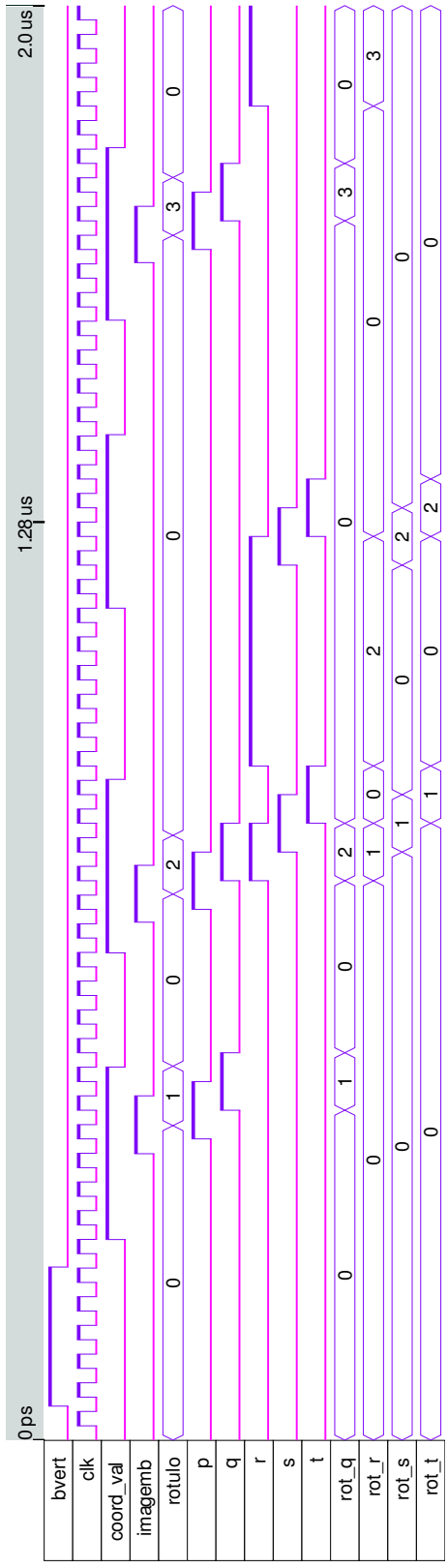

Figura 5.7: Formas de onda do circuito gerador de vizinhanças.

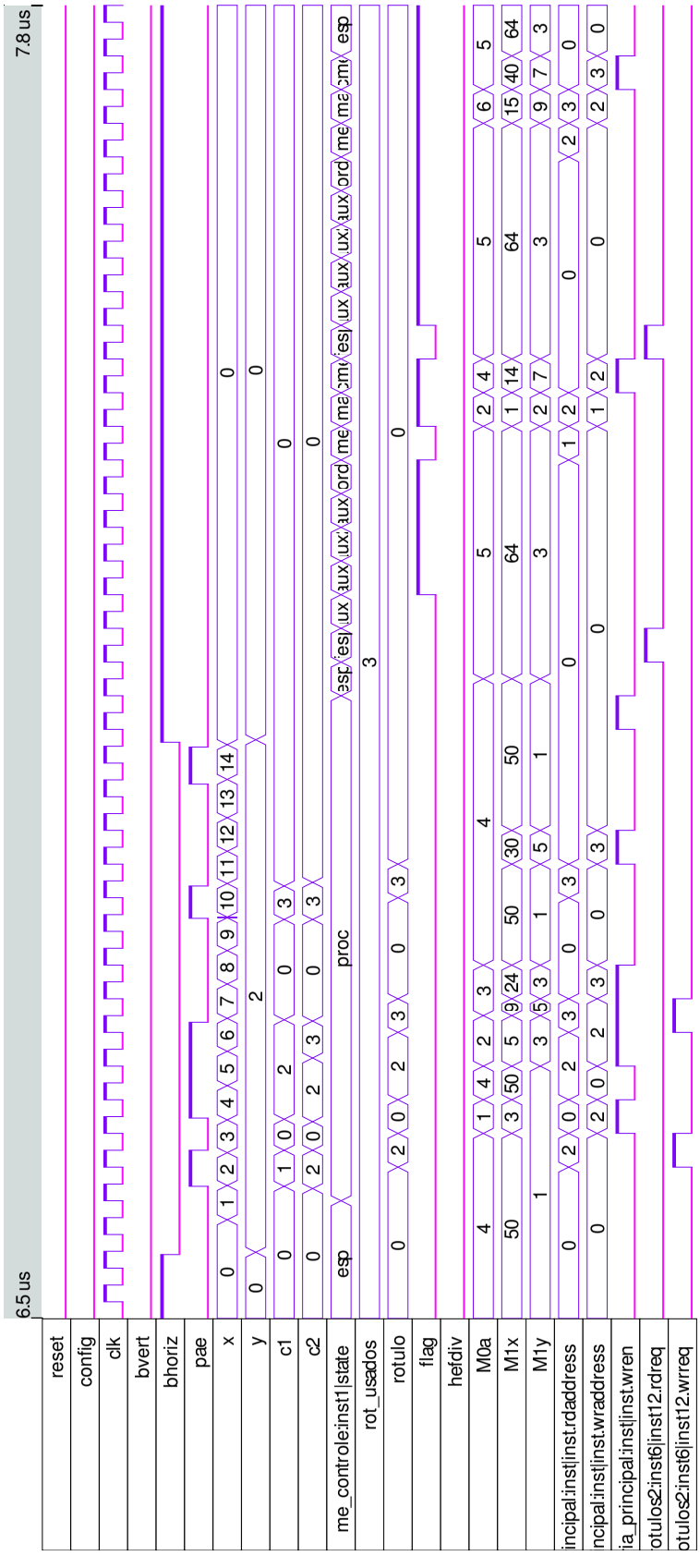

Figura 5.8: Formas de onda geradas pelo circuito principal - Terceira Linha.

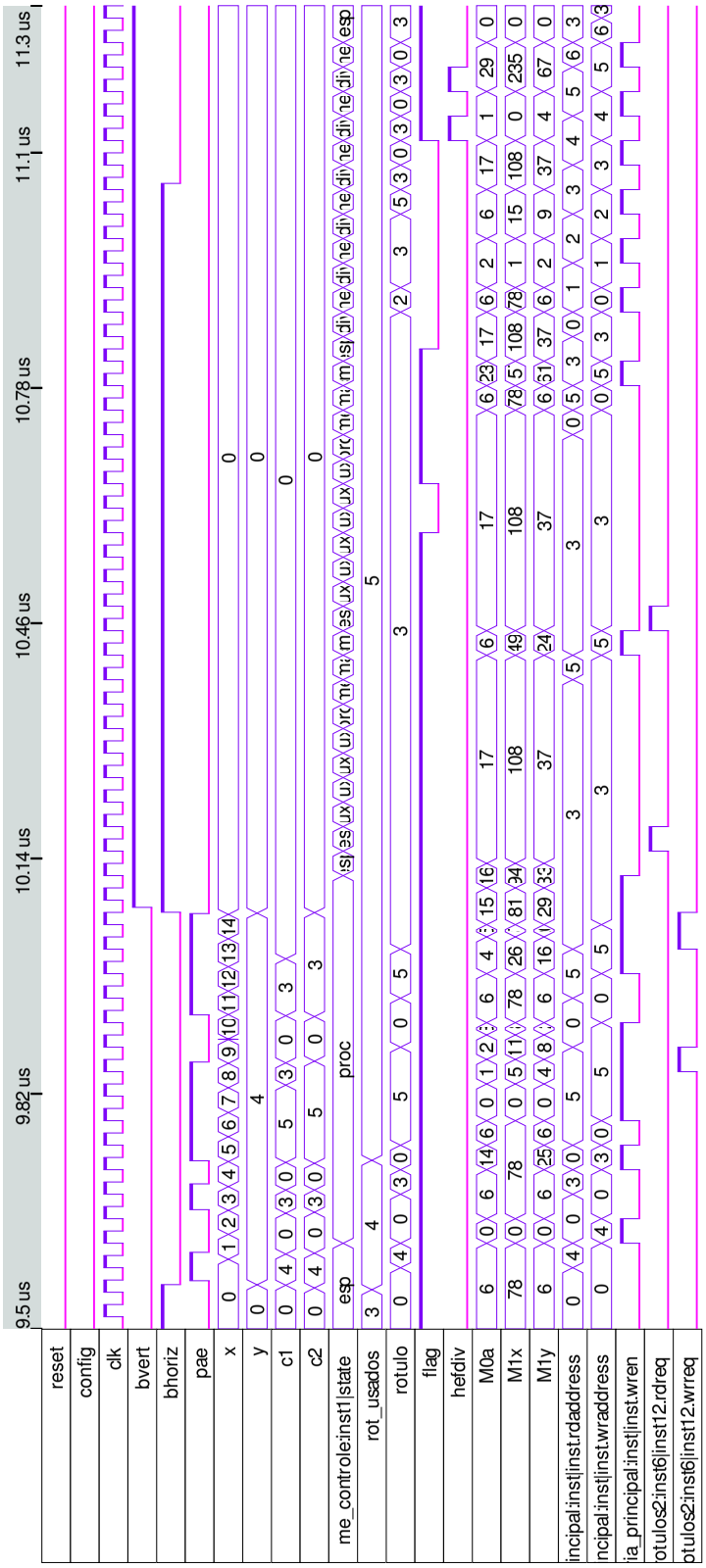

Figura 5.9: Formas de onda geradas pelo circuito principal - Funcional.

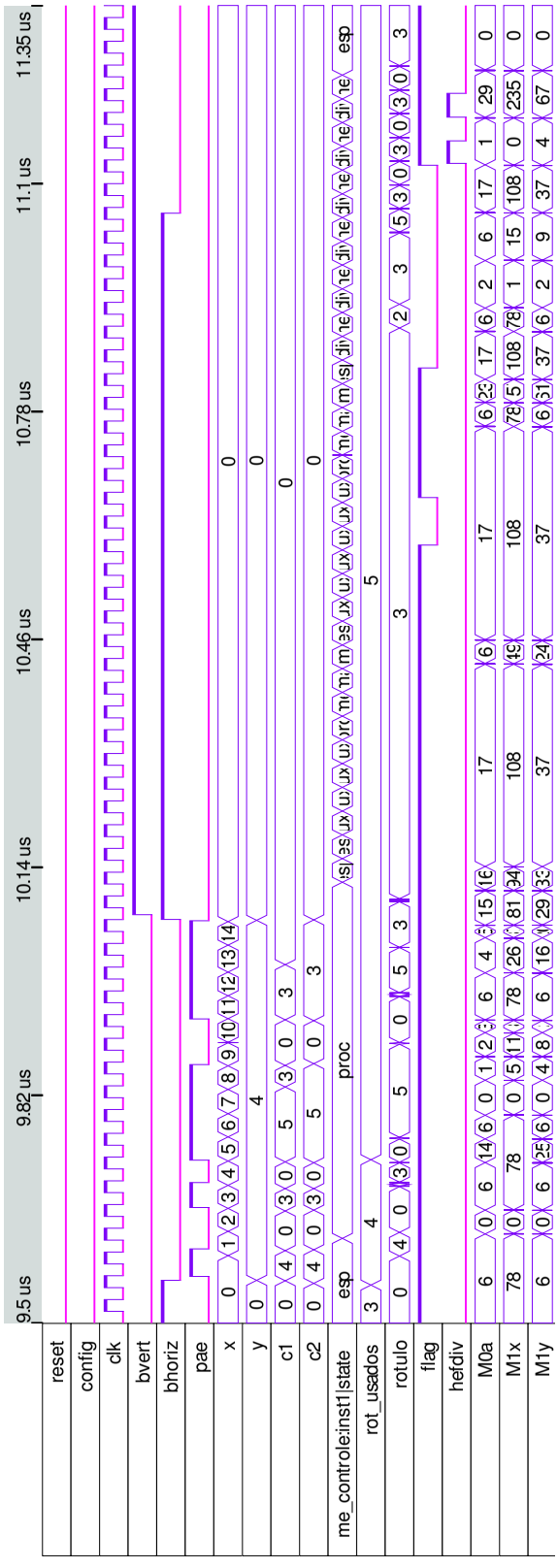

Figura 5.10: Formas de onda geradas pelo circuito principal - Temp.

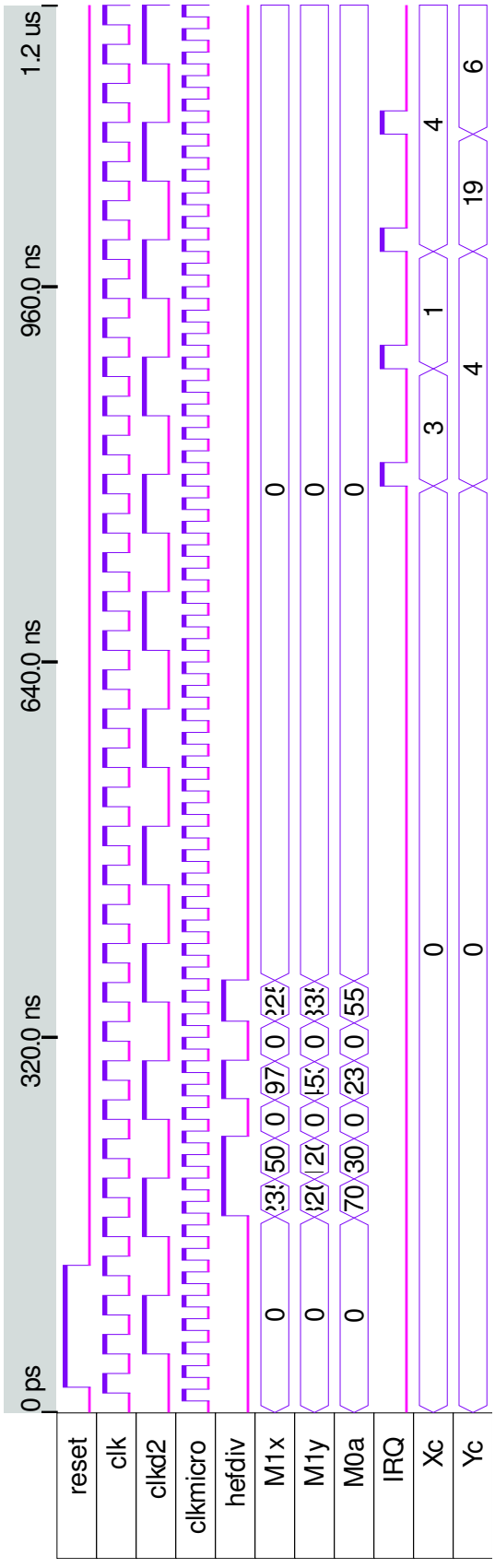

Figura 5.11: Formas de onda do cálculo dos centróides - Fun.

Capítulo 6

Conclusões e Perspectivas Futuras

A pesquisa realizada nesta tese buscou o desenvolvimento de técnicas de processamento capazes de dotar qualquer sistema que utilize a visão computacional como instrumento para realizar a observação ou o controle de um processo, com informações relevantes e em tempo real, possibilitando aumentar seu desempenho, confiabilidade e flexibilidade. Na maioria das vezes, as técnicas de processamento convencionais não são capazes de atender as necessidades de sistemas de visão que sejam projetados para trabalhar em tempo real. Por isso, técnicas de processamento paralelo em lógica programável foram usadas para desenvolver um sistema capaz de determinar características básicas, como cor e posição de múltiplos objetos numa seqüência de imagens.

A abordagem proposta fez uso de diversas técnicas de projeto envolvidas com lógica programável, como linguagens de descrição de *hardware*, para desenvolver a arquitetura de um sistema capaz de extrair características baseadas nas cores e em características regionais (momentos) dos objetos de interesse. Para a estração dos momentos dos objetos na imagem foi desenvolvido um método de rotulação de componentes conexos para aplicação em $FPGAs$ que é mesnos dependente de recursos de memória do que a maiorias dos métodos existentes.

Os resultados obtidos na implementação da arquitetura mostraram que o sistema não só é capaz de obedecer à exigência de tempo real, como é capaz de oferecer um desempenho muito superior àquele demandado pelo processamento de imagens utilizando câmeras de vídeo padrão. Isto torna esta arquitetura adequada para processar imagens provenientes de câmeras especiais de alta definição ou de altas taxas de quadros, tornando a arquitetura de extração de características, candidata a aplicações de controle por visão, controle de tráfego etc.

A abordagem modular dada ao projeto da arquitetura, bem como os resultados relativos à utilização dos recursos lógicos do componente alvo, mostram sua capacidade de ser incrementada com novas funcionalidades, ou ter suas funções modificadas para obter melhor desempenho, adequando-se mais eficazmente a determinadas tarefas.

Visando a continuação e a extensão da pesquisa realizada, são apresentadas a seguir

algumas idéias e sugestões para trabalhos futuros:

- Otimizar a arquitetura desenvolvida através do desenvolvimento de algoritmos de *hardware* mais eficientes, tanto do ponto de vista da utilização dos recursos lógicos, quanto do ponto de vista do desempenho;
- Estender a arquitetura desenvolvida para realizar os cálculos de momentos de ordem superior;
- Desenvolvimento de um sistema de aquisição de imagens usando FPGAs;
- Uso de linguagens de alto nível, como o $Handel - C$, para o projeto de lógica programável;
- Estudo da utilização da lógica para outras aplicações, por exemplo: controle de sistemas, simulação, robótica etc.

Apêndice A

Circuitos Essenciais da Arquitetura Proposta

Esse apêndice apresenta alguns dos circuitos mais importantes usados na implementação dos blocos que constituem a arquitetura para a extração das características dos objetos presentes na imagem.

O primeiro circuito apresentado, figura A.1, corresponde à primeira implementação que foi utilizada para a realização do cálculo da distância de *Mahalanobis*. Conforme abordado no capítulo 5, embora este circuito possa operar numa alta frequência de relógio ($\simeq$ 60MHz), a sua implementação utiliza uma grande quantidade ($\simeq$ 25%) dos recursos lógicos disponíveis no componente alvo (EP20K200EFC484-2X).

A figura A.2 apresenta uma implementação alternativa para o cálculo da distância de *Mahalanobis*. Como pode-se notar, embora constatado que este opere numa frequência inferior àquela da primeira implementação ($\simeq 22Mhz$), a quantidade de recursos lógicos necessários ficou reduzida aproximadamente à quarta parte da necessária para o circuito anterior.

O circuito classificador e gerador de coordenadas está apresentado na figura A.3. Este circuito é responsável pela geração da imagem binária a ser utilizada para o cálculo dos momentos. A parte de geração de coordenadas recebe os sinais de sincronização vertical ($Vsync$) e horizontal ($Hsync$), e retorna as coordenadas da imagem de entrada, assim como os sinais *bvert* e *bhoriz*, que representam, respectivamente, os períodos de *blanking* vertical e horizontal.

Na figura A.4 é apresentado o circuito responsável pela geração das vizinhanças necessárias para o processo de rotulação. Esta vizinhança é gerada por meio de filas $FIFO$.

O cálculo dos momentos é realizado pelo circuito da figura A.5. Esse circuito representa a parte essencial e de maior complexidade deste trabalho, sendo que na sua saída tem-se como resultado o valor dos momentos de ordem zero e de primeira ordem.

Uma vez tendo-se obtido tais momentos, o cálculo dos centróides é realizado pelo circuito mostrado na figura A.6. Este circuito também serve para fazer o interfaceamento entre a arquitetura proposta e um sistema de processamento externo.

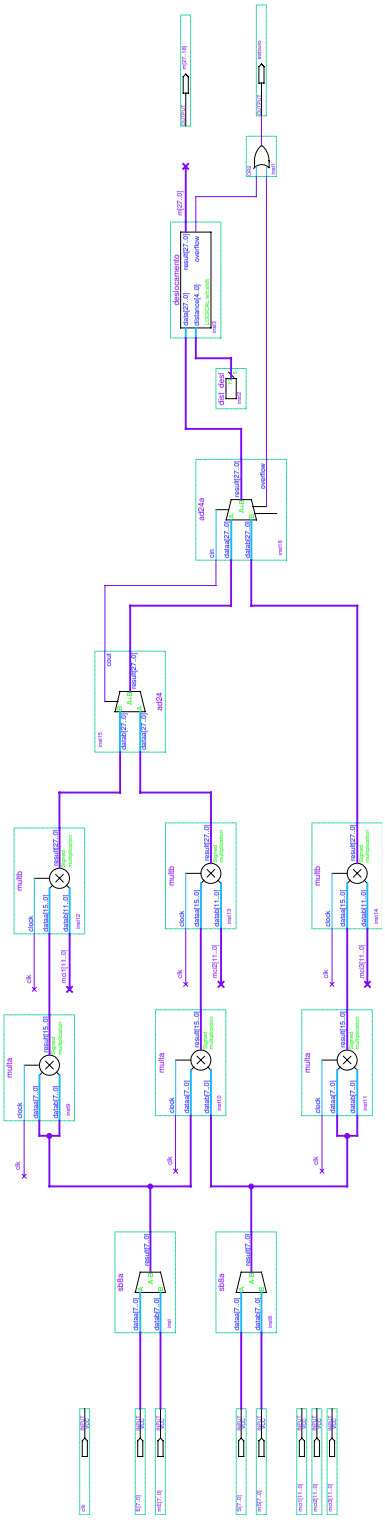

Figura A.1: Circuito para cálculo da distância de *Mahalanobis* - Primeira implementação.

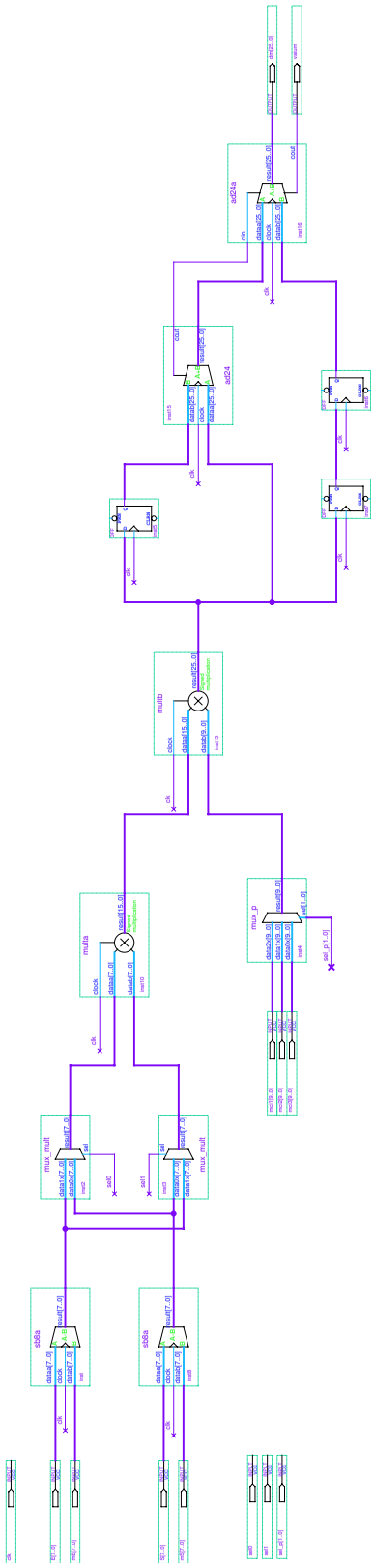

Figura A.2: Circuito para cálculo da distância de *Mahalanobis* - Segunda implementação.

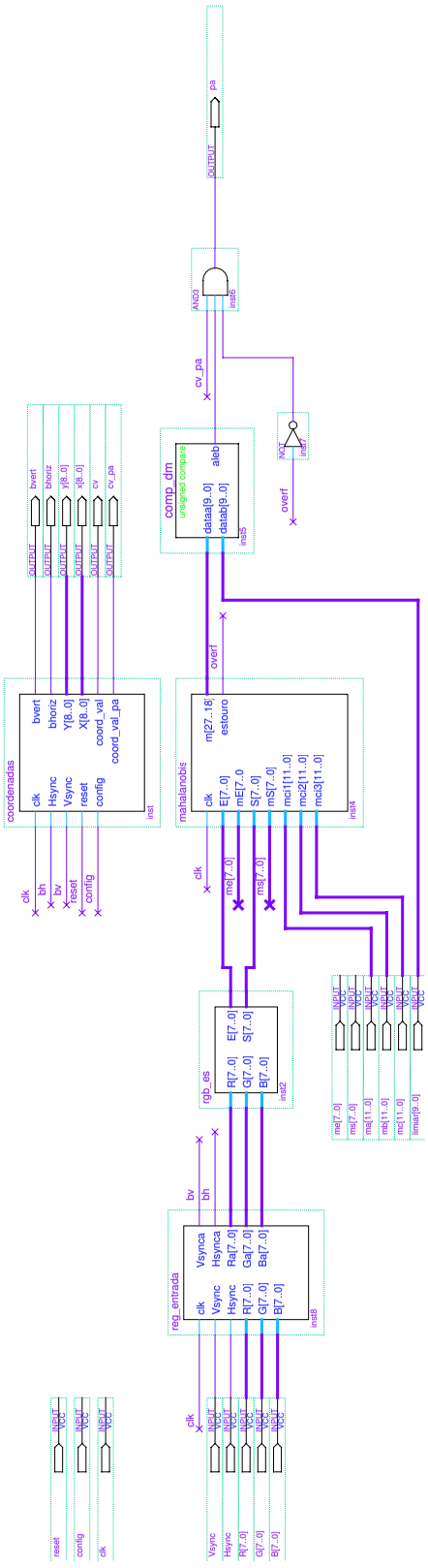

Figura A.3: Circuito classificador e gerador de coordenadas.

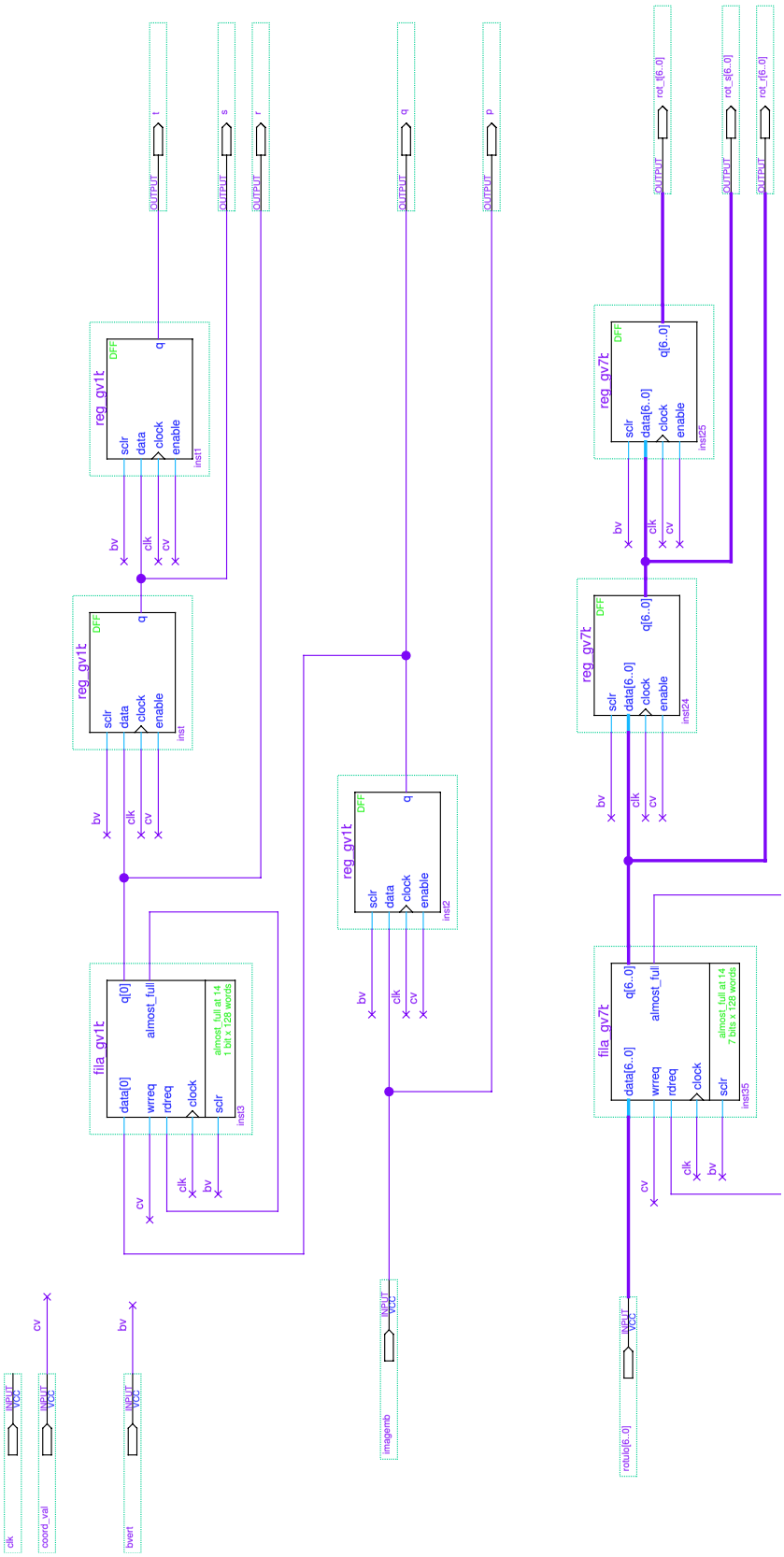

Figura A.4: Circuito para a geração das vizinhanças.

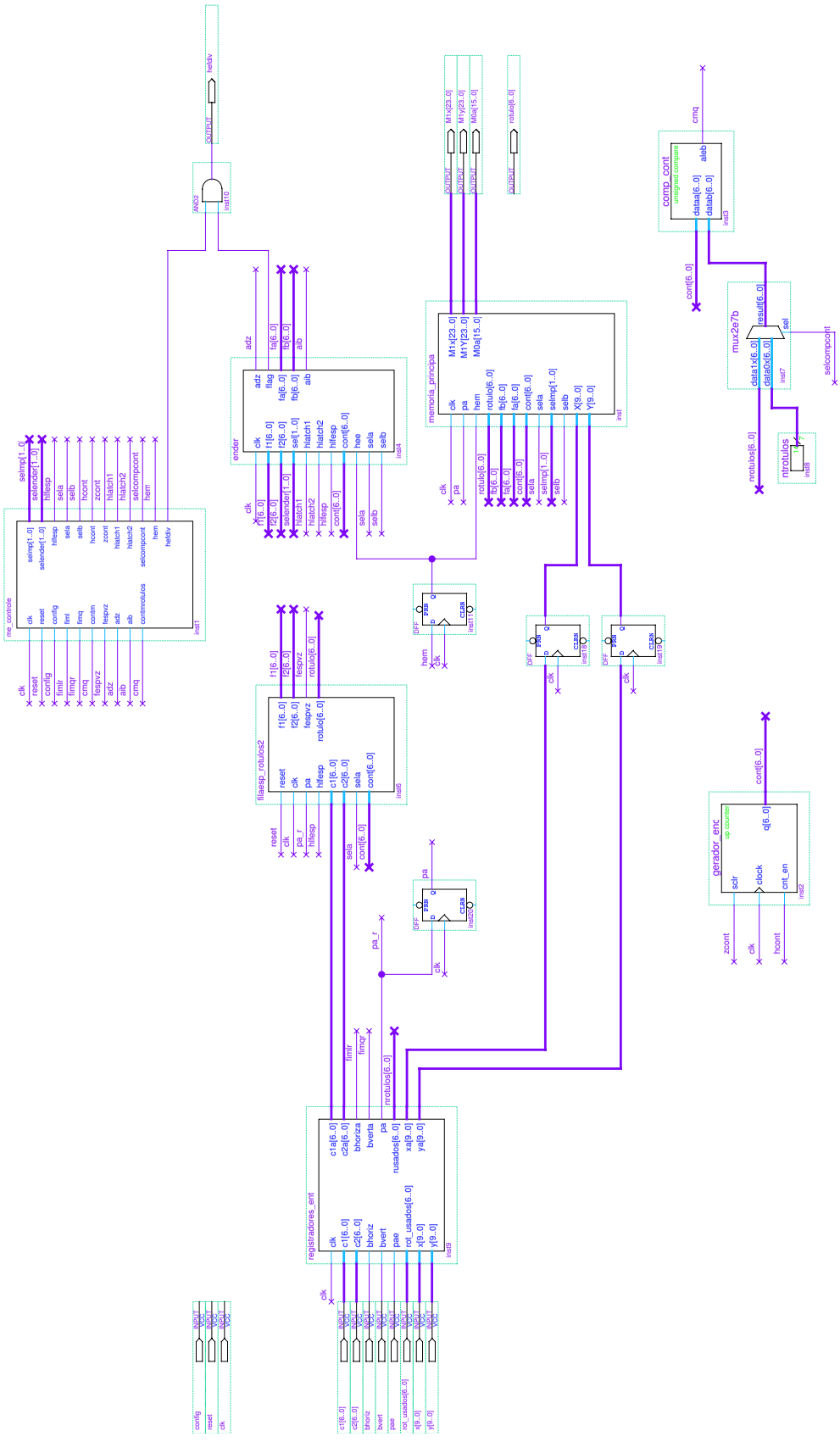

Figura A.5: Circuito para o cálculo dos momentos.

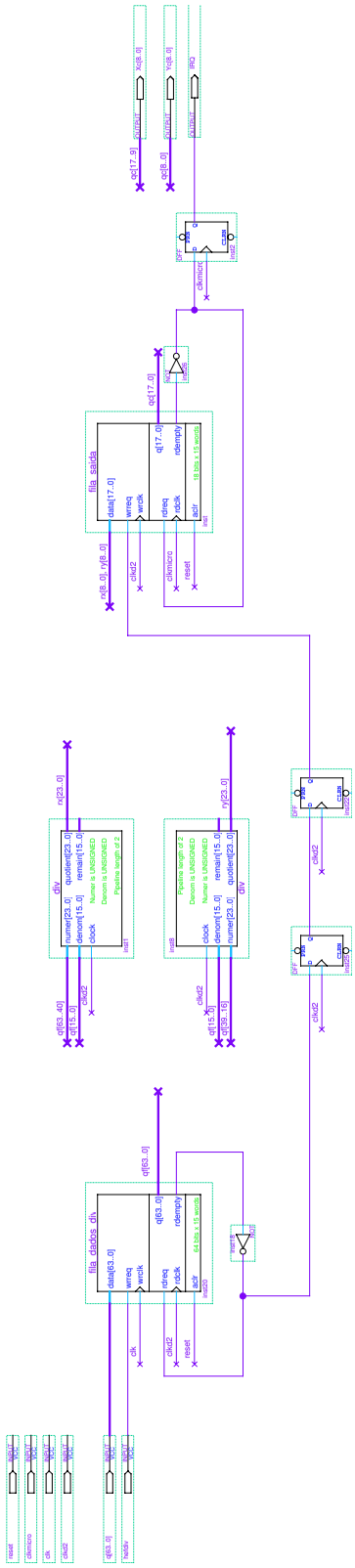

Figura A.6: Circuito para o cálculo dos centróides.

Apêndice B

Aritmética de Ponto Fixo

B.1 Introdução

A aritmética de ponto-fixo é de grande importância no projeto usando lógica programável, pois um grande número de aplicações que requerem a manipulação de números fracionários pode ser representado por ela. Dessa forma, este apêndice visa proporcionar ao leitor desse trabalho uma referência rápida aos conceitos básicos da aritmética de ponto-fixo, uma vez que, geralmente, este tipo de informação não é encontrado com facilidade na literatura.

B.2 Representação de números binários em ponto fixo

Uma coleção de N (N é um inteiro positivo) dígitos binários (*bits*) possui 2^N estados possíveis. Isto pode ser visto a partir da teoria de contagem elementar, que diz que há duas possibilidades para o primeiro *bit*, duas possibilidades para o próximo *bit* e assim por diante até o último *bit*, resultando em

$$\underbrace{2x2x2x...x2}_{Nvezes} = 2^N$$

possibilidades.

De maneira geral, estes estados podem representar uma grande variedade de coisas, por exemplo: os estudantes de uma universidade, espécies de plantas, elementos atônicos, inteiros, níveis de tensão etc.

A partir da teoria elementar de álgebra abstrata, pode-se ver uma representação como um mapeamento entre os estados binários e os elementos no conjunto de representação. Sendo

assim, o significado de uma palavra binária de N *bits* depende inteiramente de sua interpretação, isto é, da representação e do mapeamento escolhidos.

Representação de números binários fracionários sem sinal

Uma palavra de N bits, quando interpretada como um número fracionário em ponto-fixo, pode assumir valores a partir de um subconjunto P de frações não negativas dados por

$$P = \{ p/2^b \mid 0 \leq p \leq 2^N - 1, p \in Z \}$$

P contém 2^N elementos. Tal representação pode ser denotada por $U(a, b)$, onde $a = N - b$.

Na representação $U(a, b)$, o n-ésimo *bit*, contando da direita para a esquerda e começando com zero, possui um peso de $\frac{2^n}{2^b} = 2^{n-b}$. É bom notar que quando $n = b$ o peso é exatamente 1. Uma representação $U(a, b)$ possui a *bits* inteiros e b *bits* fracionários.

O valor de uma determinada palavra binária, x, de N *bits* na representação $U(a, b)$ é dado pela expressão

$$x = \left(\frac{1}{2^b}\right) \sum_{n=0}^{N-1} 2^n x_n$$

onde x_n representa o *bit* n de x. A faixa de valores da representação $U(a, b)$ vai de 0 até $\frac{(2^N-1)}{2^b}$.

Por exemplo, a representação de 8 *bits* sem sinal $U(6, 2)$ tem a forma

$$b_5 b_4 b_3 b_2 b_1 b_0 b_{-1} b_{-2}$$

onde o *bit* b_k tem um peso de 2^k. Desde que $b = 2$ o ponto binário é a direita do segundo *bit* a partir da direita. Assim, o número tem seis *bits* inteiros e dois *bits* fracionários. Esta representação tem um faixa de valores de $2^6 - 2^{-2} = 64 - \frac{1}{4} = 63\frac{3}{4}$.

A representação de números inteiros sem sinal pode ser vista como um caso especial da representação de números fracionários em ponto fixo, onde $b = 0$. Especificamente, um inteiro sem sinal de N *bits* é idêntico à representação $U(N, 0)$. Assim, a faixa de valores desse inteiro é

$$0 \leq U(N, 0) \leq 2^N - 1$$

B.2.1 Operações de complemento de um e complemento de dois

Considere uma uma palavra binária, x, de N *bits*, interpretada como uma representação binária natural de N *bits* (isto é, $U(N, 0)$). O complemnto de um de x é definido como sendo uma operação que inverte cada *bit* do valor original de x. Isto pode ser realizado aritmeticamente na

representação $U(N,0)$, subtraindo-se x de $2^N - 1$. Se o complemento de um de x for denotado por $\tilde{x}$, então

$$\tilde{x} = 2^N - 1 - x$$

O complemento de dois de x, denotado por $\hat{x}$, é determinado tomando-se o complemento de um de x e então somando-se um ao resultado

$$\begin{aligned} \hat{x} &= \tilde{x} + 1 \\ &= 2^N - x \end{aligned}$$

Complemento de dois de números fracionários

Uma palavra binária de N *bits*, quando interpretada como um número fracionário em complemento de dois, pode assumir valores de um subconjunto P de valores fracionários dado por

$$P = \{\, p/2^b \mid -2^{N-1} \leq p \leq 2^{N-1} - 1, p \in Z \,\}$$

Note que P contém 2^N elementos. Tal representação é denotada como $A(a,b)$, onde $a = N - b - 1$. O valor específico de um número binário de N *bits* na representação $A(a,b)$ é dado pela expressão

$$x = \left(\frac{1}{2^b}\right) \cdot \left[-2^{N-1}\, x_{N-1} + \sum_{n=0}^{N-2} 2^n x_n\right]$$

onde x_n é o *bit* n de x. A faixa de valores de uma representação $A(a,b)$ é

$$-2^{N-1-b} \;\leq\; x \;\leq\; 2^{N-1-b} - \frac{1}{2^b}$$

Regras fundamentais da aritmética de ponto fixo

Adição: dois números binários devem ser escalonados antes de serem adicionados, isto é, devem ter o mesmo número de *bits* para representar a parte inteira e o mesmo número de *bits* para representar a parte fracionária. A soma de dois números de M *bits* requer $M + 1$ *bits*.

Multiplicação de números sem sinal:

$$U(a_1, b_1)\ x\ U(a_2, b_2) = U(a_1 + a_2, b_1 + b_2)$$

Multiplicação de números com sinal:

$$A(a_1, b_1)\ x\ A(a_2, b_2) = A(a_1 + a_2 + 1, b_1 + b_2)$$

Divisão de números sem sinal:

$$U(a_1, b_1)/U(a_2, b_2) = U(a_1 + a_2, log_2(2^{(a_2+b_1)} - 2^(b_1 - b_2))$$

Divisão de números com sinal:

$$A(a_1, b_1)\ x\ A(a_2, b_2) = A(a_1 + a_2 + 1, a_2 + b_1)$$

Redução do tamanho de palabra;

Define-se a operação $HIn(X(a, b))$ como sendo a extração dos n *bits* mais significativos de $X(a, b)$. Similarmente, define-se a operação $HOn(X(a, b))$ como sendo a extração dos n *bits* menos significativos de $X(a, b)$. Para valores com sinal

$$\begin{aligned} HIn(A(a, b)) &= A(a, n - a - 1) \\ HOn(A(a, b)) &= A(n - b - 1, b) \end{aligned}$$

Similarmente, para valores sem sinal

$$\begin{aligned} HIn(U(a, b)) &= U(a, n - a) \\ HOn(U(a, b)) &= U(n - b, b) \end{aligned}$$

Deslocamento:

Um deslocamento pode ser tanto à esquerda ou à direita. O termo "deslocamento"pode assumir dois significados dependendo do contexto.

Definição 1: mover a palavra binária, incluindo o ponto binário, completando com zeros a partir do lado que está sendo deslocado e truncando o lado para o qual esta se deslocando.

$$\begin{aligned} X(a, b) >> n &= X(a + n, b - n) \\ X(a, b) << n &= X(a - n, b + n) \end{aligned}$$

Note que $X(a, b)$ representa tanto $U(a, b)$ quanto $A(a, b)$

Definição 2: Mover o ponto binário de modo oposto ao da direção de deslocamento, preservando o padrão de *bits*

$$\begin{aligned} X(a, b) >> n &= X(a - n, b + n) \\ X(a, b) << n &= X(a + n, b - n) \end{aligned}$$

Análise dimensional na aritmética de ponto fixo

Considere uma variável x em ponto fixo representada por $A(ax, bx)$. Sendo o valor escalonado da variável denotado por x e o valor não escalonado por X, tem-se

$$x = x/2^b$$

Unidades tais como, segundos, metros, volts etc, podem ser associadas com uma variável em ponto-fixo atribuindo-se um peso à variável. Denota-se o peso não escalonado por W de maneira que o valor e a dimensão de uma quantidade α_x que deve ser representada por x possa ser expressada na forma

$$\begin{aligned} \alpha_x &= x \times \omega \\ &= X \times W \end{aligned}$$

Se $x = X/2^{b_x}$

$$x \times \omega = X/2^{b_x} \times \omega$$

e se $x \times \omega = X \times W$,

$$X/2^{b_x} \times \omega = X \times W \Rightarrow \omega = 2^{b_x} W$$

B.2.2 Conceitos de matemática de precisão finita

Precisão: É o número máximo de números diferentes de zero representáveis. Por exemplo, a representação $A(13, 2)$ tem uma precisão de 16 *bits*. Para representações em ponto-fixo a precisão é igual ao tamanho da palavra.

Resolução: É o número máximo de *bits* diferentes de zero que podem ser representados. Por exemplo, a representação $A(13, 2)$ possui uma resolução de $1/2^2 - 0,25$.

Faixa de Valores: É a diferença entre os maiores números negativo e positivo representáveis

$$X_R = X_{MAX+} - X_{MAX-}$$

Referências Bibliográficas

Altera-Apex20K (2001), 'APEX 20K Programmable Logic Device Family'.

Altera-LPM (1996), LPM Quick Logic Reference, Relatório Técnico, Altera Corporation, http://www.altera.com/literature.

Altera-Max (2001), MAX 7000 Programmable Logic Device Family, Relatório Técnico, Altera Corporation, http://www.altera.com/literature.

Altera-NIOS (1996), Nios Quick Reference, Relatório Técnico, Altera Corporation, http://www.altera.com/literature.

Andreadis, I., Gasteratos, A. e Tsalides, P. (1996), 'An ASIC for fast grey-scale dilatation', *Microprocessors and Microsystems* **20**, 98–95.

Atmel (2001), ATF15XXAE Family Preliminary, Relatório Técnico, Atmel Corporation, http://www.atmel.com.

Baxes, G. A. (1994), *Digital Image Processing: Principles and Applications*, John Wiley and Sons, Inc.

Bezerra, E. A. e Gough, M. P. (2000), 'A Guide to Migrating From Microprocessor to FPGA Coping With The Support Tool Limitations', *Microprocessors and Microsystems* **23**, 561–572.

Bouridane, A., Crookes, D., Donachy, P., Alotaibi, K. e Benkrid (1999), 'A High Level FPGA-Based Abstract Machine for Image Processing', *Journal of Systems Architecture* pp. 809–824.

Brown, S. e Rose, J. (1996), 'Architecture of FPGAs and CPLDs: A Tutorial', *IEEE Design and Test of Computers* **13**(2), 42–57.

Choudray, A. e Patel, J. (1990), *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems*, Kluwer Academic Publishers, Dordrecht.

Costa, A., Pegoraro, R., Stolfi, G., Sichman, G., Pait, F. e Filho, F. (1999), 'GUARANÁ Robot - Soccer Team: Some Architectural Issues', *IV SBAI* pp. 457–461.

Crookes, D. (1999), 'Architectures for High Performance Image Processing: The Future', *Journal of Systems Architecture* **45**, 739–748.

Fu, K., Gonzalez, R. e Lee, C. (1987), *Robotics - Control, Sensing, Vision, and Intelligence*, McGraw-Hill, Inc.

Gonzalez, R. C. e Woods, R. E. (2000), *Processamento de Imagens Digitais*, Editora Edgard Blücher Ltda.

Grob, B. (1979), *Basic Television Principles and Servicing*, McGraw-Hill, Inc.

Hager, G. e Toyama, K. (1998), 'The 'XVision' System: A General Purpose Subtrate for Real-Time Vision Applications', *Computer Vision and Image Understending* **69**(1), 23–27.

Haralick, R. e Shapiro, L. (1992), *Computer and Robot Vision*, Vol. I, Addison-Wesley, Massachussets.

Hu, M. (1962), 'Visual Pattern Recognition By Moments Invariants', *IRE Tr Info. Theory* **IT-8**, 179–187.

Hutchison, S., Hager, G. D. e Corke, P. (1996), 'A Tutorial on Visual Servo Control', *IEEE Transactions on Robotics and Automation* **12**(5), 651–670.

Illgner, K. (2000), 'DSPs for Image and Video Processing', *Signal Processing* **80**, 2323–2336.

Lattice (2002), ispMACH 4000V/B/C Family Data Sheet, Relatório Técnico, Lattice Seniconductor Corporation, http://www.latticesemi.com.

Lee, C. e Salcic, Z. (1997), 'High-Performance FPGA-Based Implementation of Kalman Filter', *Microprocessors and Microsystems* pp. 257–265.

McKenna, S., Raja, Y. e Gong, S. (1999), 'Tracking Colour Objects Using Adaptive Mixture Models', *Image and Vision Computing* **17**, 225–231.

Mesquita, D. G. (2002), Contribuições para reconfiguração parcial, remota e dinâmica de FPGAs, Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Pós-Graduação em Ciência da Computação.

OptiMagic (2000), Frequently Asked Questions About Programmable Logic, Relatório Técnico, OptiMagic, Inc, http://www.optimagic.com/faq.html.

Perry, D. (1998), *VHDL*, Computer Science Series, third edn, McGraw-Hill, Inc.

Pratt, W. (1991), *Digital Image Processing*, 2nd edn, John Wiley & Sons, Inc.

Rasmussen, C., K.Toyama e Hager, G. (1996), Tracking Objects by Color Alone, Relatório Técnico, Yale University, 51 Prospect Street New Haven, CT 06520-8285.

Rinker, R., Carter, M., Patel, A., Chawathe, M., Ross, C., Hammes, J., Najjar, W. A. e Böhm, W. (2001), 'An Automated Process for Compiling Dataflow Graphs into Reconfigurable Hardware', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **9**(1), 130–139.

Rosenfeld, A. e Pfaltz, J. (1966), 'Sequential Operations in Digital Picture Processing', *J. Ass. Comp* pp. 471–494.

Saber, E., Tekalp, A. M., Eschbach, R. e Knox, K. (1996), 'Automatic Image Annotation Using Adaptive Color Classification', *Graphical Models and Image Processing* **58**(2), 115–126.

Sankaran, S. e Haggard, R. L. (2001), A convenient methodology for efficient translation of C to VHDL, pp. 203 –207.

Simon, L. e Pawlak, M. (1996), 'On Image Analysis by Moments', *IEEE Trans on Pattern Analysis and Machine Intelligence* **18**(3), 254–267.

Thompson, A. (1997), *Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97)*, AAI Books.

Thompson, A. (2002), *Adaptive Computing in Design and Manufacture V*, Springer-Verlag.

van der Heijden, F. (1994), *Image Based Measurements Systems: Object Recognition and Parameter Estimation*, John Wiley and Sons.

Wren, C., Azarbayejani, A., Darrell, T. e Pentland, A. (1997), 'Pfinder-Real Time Tracking of The Human Body', *IEEE Trans on Pattern Analysis and Machine Intelligence* **19**, 780–785.

Xerox (1989), Xerox Color Encoding Standards, Relatório Técnico, Xerox Systems Institute, Sunnyval, CA.

Xilinx (2002), XC2C256 CoolRunner-II CPLD, Relatório Técnico, Xilinx Corporation, http://www.xilinx.com.