Emerson Carlos Pedrino

ARQUITETURA *PIPELINE* PARA PROCESSAMENTO MORFOLÓGICO DE IMAGENS BINÁRIAS EM TEMPO REAL UTILIZANDO DISPOSITIVOS DE LÓGICA PROGRAMÁVEL COMPLEXA

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para a obtenção do Título de Mestre em Engenharia Elétrica.

ORIENTADOR : Prof. Dr. Valentin Obac Roda

São Carlos
2003

FOLHA DE JULGAMENTO

Candidato: Bacharel **EMERSON CARLOS PEDRINO**

Dissertação defendida e julgada em 17-10-2003 perante a Comissão Julgadora:

APROVADO

Prof. Assoc. **VALENTIN OBAC RODA (Orientador)**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Profa. Assoc. **ANNA HELENA REALI COSTA**
(Escola Politécnica/USP)

APROVADO

Prof. Dr. **EDUARDO MARQUES**
(Instituto de Ciências Matemáticas e de Computação/USP)

Prof. Assoc. **MURILO ARAUJO ROMERO**
Coordenador do Programa de Pós-Graduação
em Engenharia Elétrica

Profa. Assoc. **MARIA DO CARMO CALIJURI**
Presidente da Comissão de Pós-Graduação

Dedico este trabalho primeiramente a Deus, por todas as bençõe concedidas em minha vida, a meu pai Antonio (*in memorian*), à minha família, à minha esposa Marly e ao Prof. Valentin, meu orientador e um amigo inestimável.

RESUMO

PEDRINO, E. C. (2003). Arquitetura *pipeline* para processamento de imagens binárias em tempo real utilizando dispositivos de lógica programável complexa. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

A morfologia matemática é o estudo da forma utilizando as ferramentas da teoria de conjuntos e representa uma área extremamente importante em análise de imagens. Suas operações básicas são a dilatação e a erosão, e através destas é possível realizar outras operações mais complexas. A morfologia matemática fornece ferramentas poderosas para a realização de análise de imagens em baixo nível e tem encontrado aplicações em diversas áreas, tais como: visão robótica, inspeção visual, medicina, análise de textura, entre outras. Muitas destas aplicações requerem processamento em tempo real, e para sua execução de forma eficiente freqüentemente é utilizado *hardware* dedicado. A análise de imagens em baixo nível geralmente envolve computações repetidas sobre estruturas grandes de dados. Assim, o paralelismo parece ser um atributo necessário de um sistema de hardware capaz de executar eficientemente estas tarefas. As ferramentas da morfologia matemática são bem adequadas à implementação em arquiteturas *pipeline*. A necessidade de sistemas capazes de realizar o processamento de imagens digitais em tempo real, com o menor custo e tempo de desenvolvimento, tem sido suprida pela tecnologia de dispositivos de lógica programável complexa. Assim, neste trabalho foi projetada e implementada uma arquitetura *pipeline* dedicada para dilatação e erosão de imagens binárias em tempo real utilizando dispositivos lógicos programáveis de alta capacidade. Esta arquitetura é capaz de processar imagens binárias de 512x512 pixels. Os estágios desta arquitetura são flexíveis, permitindo a reprogramação da forma e do tamanho dos elementos estruturantes utilizados nas operações morfológicas. A arquitetura desenvolvida apresentou um desempenho satisfatório, demonstrando ser uma alternativa viável e eficiente.

Palavras-chave : morfologia matemática; arquitetura *pipeline*; dispositivos de lógica programável complexa

ABSTRACT

PEDRINO, E. C. (2003). Real time, programmable logic devices based, pipeline architecture for morphological binary image processing. MSc dissertation – São Carlos Engineering School, University of São Paulo, São Carlos, 2003.

Mathematical morphology is a very important image analysis area that uses set theory tools to study shapes. The basic operations in mathematical morphology are dilation and erosion, these can be used for more complex operations. Mathematical morphology has powerful tools for low level image processing and has been used in a wide range of applications such as robotic vision, visual inspection, medicine and texture analysis. Low level image processing requires repetitive processing over large data structures, dedicated parallel computing hardware is often used. Complex field programmable logic devices (CPLDs) have increasingly been used for the fast development of real time image processing systems. In this work we present a pipeline architecture for real time erosion and dilation operations, the architecture was developed using high density programmable logic devices. The developed architecture can process 512x512 pixels binary images, and has flexible stages that can be reprogrammed according to the shape and size of the structuring elements used in the morphological operations. Tests performed using the architecture demonstrated its good performance and that it is a good and efficient alternative for dedicated morphological image processing operations.

Keywords: mathematical morphology; pipeline architecture; complex programmable logic devices

SUMÁRIO

1 INTRODUÇÃO

A análise de imagens em baixo nível envolve o uso de estruturas grandes de dados e frequentemente requer altas taxas de computação. Muitas arquiteturas dedicadas já foram propostas e construídas para se lidar com estas tarefas de processamento de imagens. Logo, é intuitivo se pensar em alguma forma de paralelismo para que se possa executar estas tarefas de forma eficiente.

De acordo com (ABOTT et al., 1988), em se tratando de paralelismo, as arquiteturas *pipeline* podem algumas vezes oferecer vantagens significativas em relação às arquiteturas de arranjos de elementos de processamento individuais.

A morfologia matemática é o estudo da forma utilizando ferramentas da teoria de conjuntos e é comumente e eficientemente usada em tarefas de análise de imagens (BROGGI, 1994). Suas ferramentas básicas são a dilatação e a erosão, e podem ser descritas pelos operadores lógicos AND e OR. As ferramentas da morfologia matemática são bem adequadas à implementação em arquiteturas *pipeline*. Segundo (HARALICK et al., 1987), uma vez que a morfologia matemática se relaciona diretamente com a forma, torna-se evidente que o processamento natural para se tratar problemas de identificação de objetos por visão de máquina ou ainda identificar objetos defeituosos numa linha de montagem é a morfologia matemática. Também, dilatando ou erodindo objetos de forma algorítmica, pode-se detectar bordas, esqueletos, eliminar ruídos e outras operações.

O grande desafio na área de visão computacional é o desenvolvimento de sistemas capazes de realizar o processamento de imagens digitais em tempo-real com o menor custo

possível e com possibilidade de uso em aplicações industriais e comerciais. A utilização de computadores de uso geral permite a rápida implementação de sistemas e de vários tipos de algoritmos, simples ou complexos, específicos para o processamento de imagens. Todavia, não é possível obter o maior rendimento exigido com estas máquinas devido à diversas limitações. Uma das maneiras de se contornar este inconveniente é através do uso de arquiteturas dedicadas projetadas em hardware para se executar os algoritmos de processamento de imagens em tempo real. A utilização de dispositivos lógicos programáveis complexos do tipo CPLDs e FPGAs no desenvolvimento de projetos em hardware tem possibilitado o rápido desenvolvimento, análise e implementação de projetos lógicos complexos a um custo consideravelmente menor quando comparado aos custos de desenvolvimento de uma pastilha VLSI. Assim, o desenvolvimento de projetos de sistemas específicos para o processamento de imagens de forma *customizada* tem se beneficiado com esta nova tecnologia, tanto pela diminuição do tempo necessário para a implementação do sistema e dos gastos com material e mão de obra, quanto pela possibilidade de implementação de sistemas que processem as informações em tempo real. Portanto, neste trabalho de mestrado foi construída uma arquitetura dedicada em hardware para processar as operações básicas de morfologia matemática binária em tempo real através do uso de dispositivos de lógica programável de alta capacidade. Por meio desta arquitetura é possível processar imagens binárias de 512x512 pixels e também reprogramar a forma e o tamanho do elemento estruturante utilizado nas operações morfológicas. Também, com esta arquitetura é possível se realizar outras aplicações mais complexas em processamento de imagens. O desempenho do hardware desenvolvido demonstrou que esta arquitetura é uma alternativa viável e eficiente.

O capítulo 2 fornece uma visão geral sobre dispositivos de lógica programável complexa (CPLDs), inserindo-os no contexto das demais metodologias alternativas existentes para projeto de hardware.

O capítulo 3 apresenta uma revisão sobre morfologia matemática binária, destacando-se algumas de suas aplicações importantes em processamento de imagens. Também, as definições básicas para as operações de dilatação, erosão, abertura e fechamento são estendidas para imagens em níveis de cinza.

No capítulo 4, apresenta-se uma arquitetura desenvolvida através de um CPLD da Altera, para aquisição, armazenamento e exibição de imagens monocromáticas. Esta arquitetura fez parte da fase inicial deste trabalho, que teve como objetivo a familiarização com projetos utilizando CPLDs e o domínio do processamento do sinal de vídeo fornecido por sistemas convencionais de vídeo analógico. Também, esta serviu de base à arquitetura desenvolvida que foi o objetivo do trabalho de mestrado.

O capítulo 5 apresenta uma revisão sobre arquiteturas existentes para morfologia matemática. Também, as arquiteturas mais relevantes para este trabalho são discutidas pormenorizadamente.

No capítulo 6, apresentam-se de forma minuciosa, o projeto e a implementação de uma arquitetura *pipeline* dedicada para dilatação e erosão de imagens binárias em tempo real utilizando CPLDs. Os resultados obtidos são apresentados e comparados com outras formas de implementação.

No capítulo 7, apresentam-se as conclusões e sugestões para trabalhos futuros.

2 DISPOSITIVOS DE LÓGICA PROGRAMÁVEL (PLDs)

2.1 Considerações Iniciais

Embora o objetivo deste capítulo seja fornecer uma visão geral sobre dispositivos de lógica programável de alta capacidade, é importante discernir onde estes se inserem no contexto das demais metodologias alternativas para projeto de hardware.

2.1.1 Metodologias de Projeto de Circuitos Integrados

Estimulado pelo desenvolvimento de novos tipos de dispositivos lógicos programáveis [PLDs (*Programmable logic devices*)] mais sofisticados, o processo de projeto de circuitos digitais têm mudado radicalmente nos últimos anos (BROWN e ROSE, 1996). Também, de acordo com (CHAN e MOURAD, 1994), o projeto de circuitos digitais têm sofrido várias evoluções nas últimas décadas. Os componentes dos circuitos evoluíram de transistores individuais a circuitos integrados de integração em escala muito alta [VLSI (*Very large scale integration*)]. As ferramentas CAD (*Computer aided design*) têm acelerado o ciclo de projeto. Não é mais necessário montar diferentes componentes ou desenhar portas lógicas individuais. As linguagens de descrição de hardware [HDLs (*Hardware description languages*)] têm facilitado a descrição de projetos complexos. Ferramentas de síntese lógica automática estão disponíveis para criar circuitos que são rapidamente mapeados na tecnologia em questão. Além das mudanças na tecnologia e técnicas de projeto de VLSI, o ciclo de vida dos produtos modernos está se tornando mais curto em relação ao ciclo de projeto. Assim, há a necessidade de uma rápida prototipação.

As implementações de circuitos podem ser agrupadas em duas categorias principais: circuitos completamente *customizados* e *semicustomizados*, conforme ilustrado na figura 2.1.

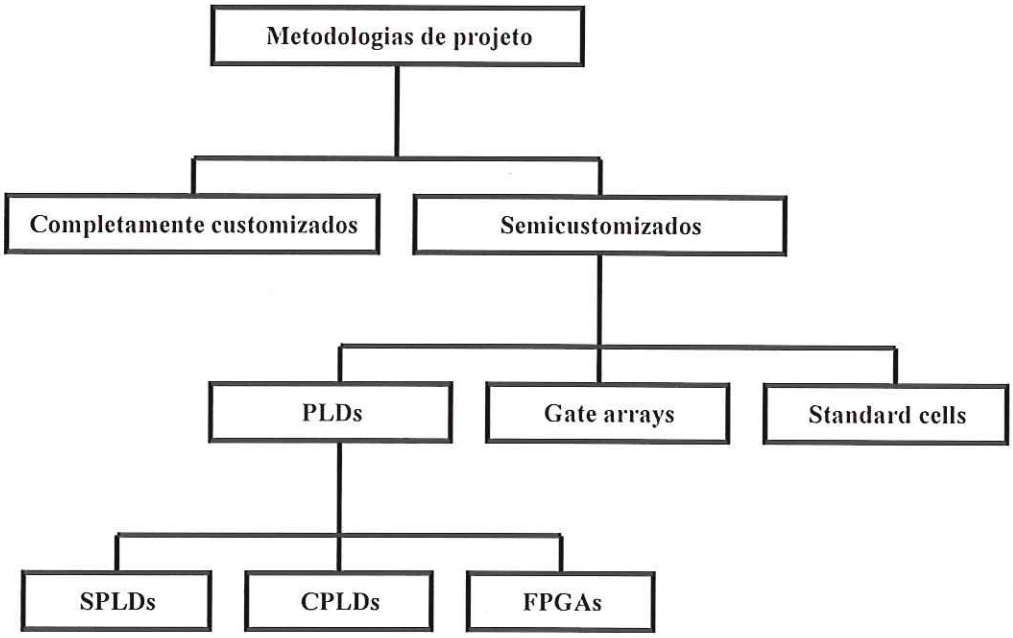


FIGURA 2.1 – Metodologias de projeto.

De acordo com a figura 2.1, tem-se:

1. **CIs (*Circuitos integrados*) completamente *customizados*:** Neste estilo de projeto cada função lógica primitiva é manualmente projetada e otimizada, logo, exige-se máscaras específicas para cada projeto. Uma vez que o projetista controla todos os estágios de *layout* do *chip*, é possível se obter uma máxima flexibilidade de projeto e um alto desempenho. O CI resultante é mais compacto, oferece maior desempenho e um baixo consumo de energia. Em contraposição, o tempo de desenvolvimento é longo e os custos são extremamente altos. Para aplicações que requerem grandes volumes de produção, estes CIs fornecem uma alternativa de baixo custo.

2. **MPGAs (*Mask-programmable gate arrays*):** Nesta implementação o projeto é geralmente facilitado e agilizado pela existência de uma biblioteca de células. O projeto é mapeado em um arranjo de transistores pré-fabricados numa pastilha. As interconexões são *customizadas* e feitas durante o processo de fabricação. Logo, há a necessidade de máscaras específicas para as interconexões entre os blocos. O desempenho obtido pode ser muito bom e são mais densos que os PLDs. Pelo fato do arranjo de transistores ser produzido em massa, os custos de fabricação são os menores se comparados a CIs completamente *customizados* e *standard cells* (LANDIS, 1996).

3. **Standard cells:** Como no caso de MPGAs, a tarefa de projeto é facilitada pelo uso de módulos pré-projetados. Os módulos, *standard cells*, são frequentemente salvos em uma base de dados. O processo de projeto físico é automatizado por ferramentas CAD. Comparados a CIs *customizados*, os circuitos implementados em *standard cells* são menos eficientes em tamanho e desempenho, entretanto, seu custo de desenvolvimento é mais baixo e o tempo de projeto é menor.

4. **PLDs (*Programmable logic devices*)**: Termo geral que se refere a qualquer tipo de circuito integrado usado para se implementar circuitos digitais onde o *chip* pode ser configurado e reconfigurado pelo usuário final por meio de um *software* específico fornecido pelo seu fabricante. Logo, os projetistas podem produzir CIs ASICs (*Application specific integrated circuits*) sem a necessidade de passar por um processo de fabricação. Também, podem ser reprogramados no próprio sistema pela tecnologia ISP (*In-system programmability*), facilitando-se assim possíveis mudanças no projeto. Possuem como característica principal um curto ciclo de projeto se comparados a CIs completamente *customizados* ou *semicustomizados*. Todas as vantagens supracitadas aliadas a um baixo custo inicial e às altas capacidades lógicas existentes para PLDs, popularizaram seu uso grandemente nos últimos anos (SALCIC e SMAILAGIC, 1998).

Os PLDs se dividem basicamente em:

1. **SPLDs (*Simple PLDs*)**. Como exemplo, pode-se citar os dispositivos lógicos programáveis PLA (*Programmable logic array*), atualmente obsoletos, e os PAL (*Programmable array logic*).

2. **HCPLDs (*High capacity PLDs*)**: simples acrônimo para ambos, CPLDs (*Complex PLDs*) e FPGAs (*Field programmable gate arrays*).

2.2 Evolução dos PLDs

O primeiro tipo de *chip* programável pelo usuário que podia implementar circuitos lógicos foi a PROM (*Programmable read-only memory*), onde as linhas de endereçamento podiam ser usadas como entradas do circuito lógico e as linhas de dados como saídas. As funções lógicas, entretanto, raramente requerem mais do que alguns termos produto, e uma PROM contém um decodificador completo para seus endereços de entrada, logo, representam uma arquitetura ineficiente para a realização de circuitos lógicos e raramente são utilizadas na prática. Posteriormente foram desenvolvidos os PLAs (*Programmable logic arrays*) para a implementação de circuitos lógicos. Um PLA possui dois níveis de portas lógicas: um plano de portas AND seguido por um plano de portas OR, ambos programáveis. Cada saída do plano AND pode corresponder a qualquer termo produto de suas entradas. Similarmente, cada saída do plano OR pode ser configurada para produzir a soma lógica de quaisquer saídas do plano AND. Tal estrutura é bem adequada para a implementação de funções lógicas na forma de soma de produtos. Também, são bastante versáteis, uma vez que ambos os termos, AND e OR, podem ter várias entradas. Na figura 2.2, tem-se o esquema simplificado de um PLA onde as interconexões programáveis são representadas pelos **x**'s (KATZ, 1993).

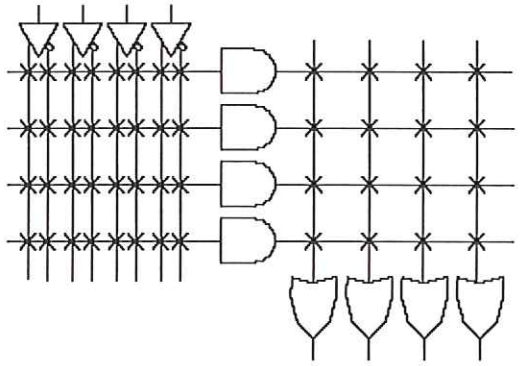


FIGURA 2.2 – Circuito simplificado de um PLA.

Quando os PLAs foram introduzidos no início dos anos 70 pela Philips, seus principais problemas eram o alto custo de fabricação e o pobre desempenho de velocidade. Ambas as desvantagens eram devidas aos dois níveis de lógica configurável, pois os planos lógicos eram difíceis de se fabricar e introduziam atrasos significantes de propagação. Para superar estas dificuldades, os dispositivos PALs (*Programmable array logic*) foram desenvolvidos em 1978 pela Monolithic Memories Inc (HOROWITS e HILL, 1989). Os dispositivos PALs apresentam um único nível de programação, consistindo-se de um plano AND programável que alimenta portas OR fixas. Na figura 2.3, tem-se um exemplo de um dispositivo PAL.

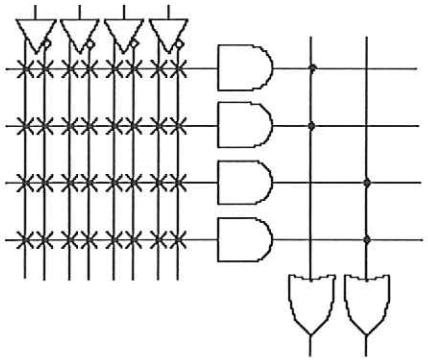


FIGURA 2.3 – Circuito simplificado de um dispositivo PAL.

Para compensar a falta de generalidade devida ao plano OR fixo, muitas variantes de dispositivos PALs são produzidas com diferentes números de entradas e saídas, e vários tamanhos de portas OR. Também, com eles é possível a implementação de circuitos sequenciais, pois geralmente possuem *flip-flops* conectados às saídas das portas OR. Os dispositivos PALs tiveram um profundo efeito no projeto de hardware digital e são a base para dispositivos lógicos programáveis mais complexos. Há outras variantes de arquiteturas semelhantes às dos dispositivos PALs com acrônimos diferentes. Todos os pequenos PLDs, incluindo PLAs, PALs e demais dispositivos similares, são agrupados na categoria de SPLDs, possuindo como características mais importantes um baixo custo e um alto desempenho. Há duas tecnologias de programação disponíveis para SPLDs, bipolar e CMOS (*Complementary metal oxide semiconductor*). A primeira utiliza fusíveis para as interconexões, sendo portanto, programável uma única vez, ou seja, caracterizando um dispositivo OTP (*One time programmable*), e a última utiliza transistores MOS de *gate* flutuante.

Com o avanço da tecnologia, foi possível produzir dispositivos com maior capacidade do que à dos SPLDs. A dificuldade para se aumentar a capacidade dos SPLDs esbarra no fato da estrutura do plano lógico programável crescer muito rapidamente em tamanho à medida que o número de entradas aumenta. Uma maneira de se prover dispositivos lógicos programáveis mais complexos baseados em arquiteturas de SPLDs, consiste-se em se integrar vários SPLDs em um único *chip* e fornecer interconexões programáveis para conectar seus blocos constituintes.

Os CPLDs foram originalmente desenvolvidos pela Altera, primeiramente em sua família de *chips* conhecida como *Classic* EPLDs (*Erasable programmable logic devices*) e posteriormente em três séries adicionais: MAX 5000, MAX 7000 e MAX 9000. Com o rápido crescimento do mercado de PLDs de alta capacidade, outros fabricantes também desenvolveram dispositivos na categoria de CPLDs. Existe atualmente uma variedade muito grande de opções disponíveis.

Os *chips* lógicos para propósitos gerais de mais alta capacidade disponíveis hoje em dia são os tradicionais *gate arrays* ou MPGAs. Os *gate arrays* motivaram o desenvolvimento dos FPGAs. Assim como estes, os FPGAs são formados por um arranjo bidimensional de blocos lógicos cercados por blocos de I/O programáveis e possuem recursos de interconexões que podem ser programados pelo usuário final. Também, possuem capacidade superior à dos CPLDs. Logo, oferecem as vantagens de ambos. Os FPGAs foram introduzidos pela Xilinx em 1985 e foram responsáveis pela mudança na maneira de se projetar circuitos digitais. Existem diversos fabricantes produzindo FPGAs atualmente. Os três aspectos principais que definem a arquitetura de um FPGA são: tecnologia de programação, arquitetura das células e estrutura de roteamento.

2.3 Tecnologias de Programação

O primeiro tipo de comutador programável pelo usuário final foi o fusível. Esta tecnologia era usada nos PLAs e ainda hoje é utilizada para se fabricar SPLDs bipolares (BROWN e ROSE, 1996; KNAPP, 2000). Um fusível é uma conexão metálica que pode ser programada (conexão aberta) fazendo-se passar pela mesma uma corrente de um determinado valor. Portanto, sua tecnologia é denominada OTP. Para dispositivos de alta capacidade, onde a tecnologia CMOS prevalece, foram desenvolvidas diferentes abordagens para a implementação de comutadores programáveis. Para CPLDs, a tecnologia principal utilizada é a de transistores com *gate* flutuante, que são os mesmos utilizados em memórias EPROM (*Erasable programmable read only memory*) e EEPROM (*Electrically erasable programmable read only memory*). Para FPGAs, as tecnologias mais comuns são SRAM (*Static RAM*) e *antifuse*.

As tecnologias EPROM e EEPROM são geralmente utilizadas em SPLDs e CPLDs. As EPLDs da família MAX 5000 da Altera e as da Xilinx, utilizam células EPROM em suas tecnologias de programação. Um transistor EPROM se parece com um transistor MOS a menos de um segundo *gate*, isolado por uma camada de óxido, inserido entre o *gate* de controle e o canal. Sua célula é tão pequena quanto à de um *antifuse*.

A programação de um transistor EPROM de canal *n* com gate flutuante é realizada da seguinte maneira: aplicando-se uma voltagem maior que 12V ao *gate* de controle da célula, e uma voltagem menor ao dreno, aparecerá um forte campo elétrico fazendo com que elétrons originados na fonte e indo na direção do dreno ganhem energia suficiente para saltarem sobre o *gate* flutuante e se aprisionarem[1] nele. Este *gate* se comporta como um capacitor com constante de tempo secular. Os elétrons aprisionados elevam o *threshold* de voltagem, ou seja, a voltagem no *gate* de controle através da qual o transistor começa a conduzir. Assim, o transistor permanece cortado mesmo com uma diferença de potencial normal aplicada ao seu *gate* de controle. Uma vez que o *gate* isolado não é acessível eletricamente, ele só poderá ser descarregado por radiação de luz ultravioleta, fazendo-se com que as cargas armazenadas se desprendam por fotocondução. A programação de um transistor EEPROM é similar à descrita anteriormente, mudando-se apenas o mecanismo de apagamento, que é feito por um campo elétrico aplicado entre o *gate* de controle e a fonte. Este método de apagamento é mais rápido que o utilizado por transistores EPROMs e o *chip* não precisa ser removido do sistema. A figura 2.4 ilustra o processo de programação e apagamento de uma célula EPROM.

---

[1] Este processo é conhecido como *hot-electron injection* ou *avalanche injection*. Algumas vezes, a tecnologia EPROM é chamada de FAMOS (*Floating-gate Avalanche MOS*).

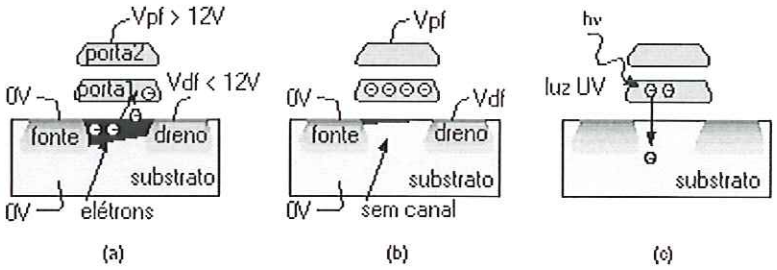
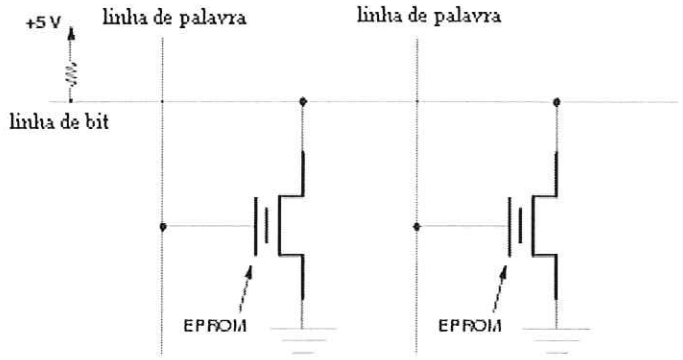


FIGURA 2.4 – Processo de programação e de apagamento de uma célula EPROM. Em (a), tem-se o processo conhecido como *hot electron injection*. Em (b), tem-se a célula já programada e em (c) os elétrons recebem energia suficiente para retornarem ao substrato.

A programação de um transistor EPROM ou EEPROM é mantida mesmo quando a alimentação for desligada, assim, evita-se a necessidade de reprogramar o dispositivo quando a alimentação for religada. Apesar da célula EEPROM ser maior que a célula EPROM, ela oferece a vantagem de poder ser apagada eletricamente. Também, pode ser reprogramada no próprio circuito (ISP). A figura 2.5 ilustra como transistores EPROMs podem ser conectados em um plano AND de um CPLD (BROWN e ROSE, 1996; ROSE et al.,1993).



F IGURA 2.5 – Comutadores programáveis baseados na tecnologia EPROM.

As conexões programáveis de FPGAs baseados na tecnologia de programação SRAM são feitas por transistores de passagem ou multiplexadores que são controlados por células SRAMs distribuídas pelo *chip*. A vantagem desta tecnologia é que ela permite uma rápida reconfiguração no próprio circuito. A principal desvantagem é o tamanho da célula SRAM. Por causa de sua volatilidade, as configurações das células devem ser restabelecidas sempre que o dispositivo for religado, necessitando-se portanto de um dispositivo de memória externo. Uma vez que elas podem ser reconfiguradas facilmente, suas configurações podem ser mudadas para consertar *bugs* ou para fazer atualizações. Deste modo, os FPGAs baseados nesta tecnologia fornecem um meio ideal para prototipação (HAUCK, 1998). Também, podem ser usados em situações onde se exigem diversos tipos de configurações. A figura 2.6 mostra um exemplo de comutadores programáveis controlados por células SRAMs conectando dois blocos lógicos.

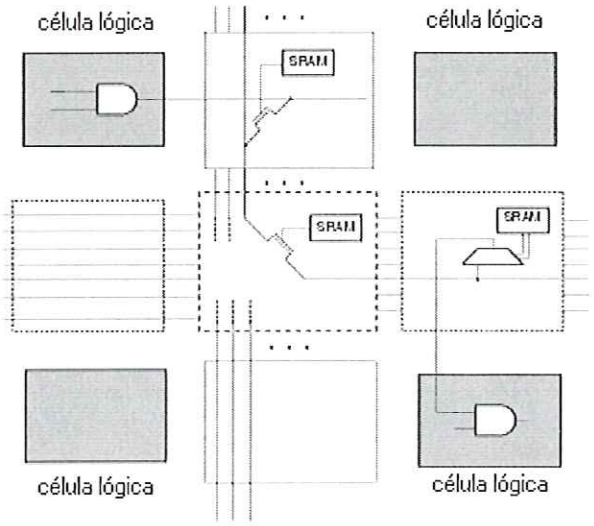


FIGURA 2.6 – Comutadores programáveis controlados por SRAMs.

O outro tipo de comutador programável utilizado em FPGAs é o *antifuse*. Os *antifuses* são dispositivos OTP. Quando programados oferecem uma baixa resistência, e quando não, comportam-se como um circuito aberto. Logo, possuem um comportamento antagônico ao de um fusível normal. Os *antifuses* possuem resistência e capacitância

parasitas menores que às de um transistor de passagem. Logo, reduzem os atrasos RC no roteamento (GREENE et al., 1993). Apesar das vantagens dos *antifuses* em relação às demais tecnologias, estes necessitam de transistores ocupando largas áreas no dispositivo.

Estes dispositivos se enquadram em duas categorias, *amorphous silicon* e *dieletric*. O primeiro tipo possui uma camada de silício amorfo entre duas camadas de metal, que quando submetida a uma determinada corrente se torna condutiva. Oferece alguns problemas em relação ao segundo. A Segunda categoria se baseia em um dielétrico colocado entre duas camadas condutoras, *N+ diffusion* e *polysilicon*. Submetido a uma determinada voltagem, este dielétrico se rompe permitindo a condução do dispositivo. O *antifuse* PLICE (*Programmable low-impedance circuit element*) da Actel possui um dielétrico constituído por três camadas, chamado ONO (*Oxide-nitride-oxide*). Quando o *antifuse* está desprogramado este dielétrico evita a passagem de corrente entre as camadas condutoras. Ao se aplicar um pulso de 16V através do dielétrico, este se derrete permitindo a formação de um canal condutor entre as camadas de difusão e de silício. O processo descrito anteriormente está ilustrado na figura 2.7.

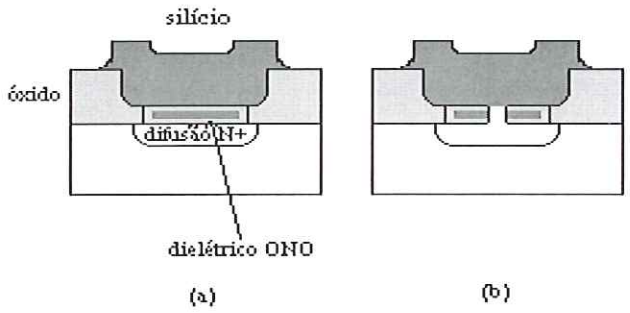


FIGURA 2.7 - *Antifuse* PLICE da Actel. (a) Antes da programação. (b) Após a programação.

Para se finalizar esta seção, abordar-se-á sucintamente sobre a tecnologia FLASH. Basicamente esta tecnologia possui as características de apagamento da tecnologia EEPROM com uma célula de tamanho comparável à tecnologia EPROM, entretanto, seu custo é menor. A diferença principal entre uma célula FLASH e uma célula EPROM está na espessura da camada de óxido entre o substrato e o *gate* flutuante. Ela é de

aproximadamente 100 ângstroms para a tecnologia FLASH e maior que 150 ângstroms para a tecnologia EPROM. Em suma, as características das memórias FLASH são: não volátil, apagável eletricamente por setor ou totalmente no circuito, grande densidade, alta velocidade de acesso e baixo custo. A tabela 2.1 fornece um sumário das tecnologias de programação discutidas nesta seção.

Todos os comutadores programáveis sobreditos ocuparão determinadas áreas no dispositivo e apresentarão resistências e capacitâncias que limitarão o desempenho do dispositivo se comparado a um *gate array* tradicional. Também, os PLDs baseados na tecnologia EEPROM ou FLASH consumirão mais energia do que os baseados em SRAM ou *Antifuse*.

TABELA 2.1 – Sumário das tecnologias de programação.

NOME	REPROGRAMAÇÃO	VOLATILIDADE	TECNOLOGIA
Fuse	Não	Não	Bipolar
EPROM	Sim, fora do circuito	Não	UVCMOS
EEPROM	Sim, no circuito	Não	EECMOS
SRAM	Sim, no circuito	Sim	CMOS
Antifuse	Não	Não	CMOS+

Comercialmente a tecnologia *fuse* é utilizada pela Texas Instruments. A tecnologia EPROM é utilizada pela Xilinx, Altera, AMD, Cypress, Philips, Atmel, ICT e WSI. A tecnologia EEPROM é utilizada pela Altera, AMD e Lattice. A tecnologia SRAM é utilizada pela Xilinx, Altera, Actel, Lucent, Motorola e DynaChip. A tecnologia *antifuse* é utilizada pela Actel e QuickLogic. A tecnologia FLASH é utilizada pela Cypress.

2.4 CPLDs

Os CPLDs são formados por múltiplos blocos lógicos que se comunicam através de uma estrutura programável de interconexão global. Cada bloco lógico em um CPLD é similar a um SPLD, entretanto, sua estrutura interna é mais sofisticada. Tipicamente, cada bloco lógico contém de 4 a 16 macrocélulas, dependendo de sua arquitetura. Uma macrocélula, nos CPLDs mais modernos, compreende um circuito lógico combinacional formado por soma de produtos mais um flip-flop opcional. Pode ter um grande número de entradas, entretanto, a complexidade de sua função lógica é limitada. As macrocélulas dentro de um bloco lógico geralmente podem ser totalmente conectadas. Dependendo da família e fornecedor, nem todos os CPLDs podem ser totalmente conectados entre seus blocos lógicos. A figura 2.8 mostra a arquitetura básica de um CPLD.

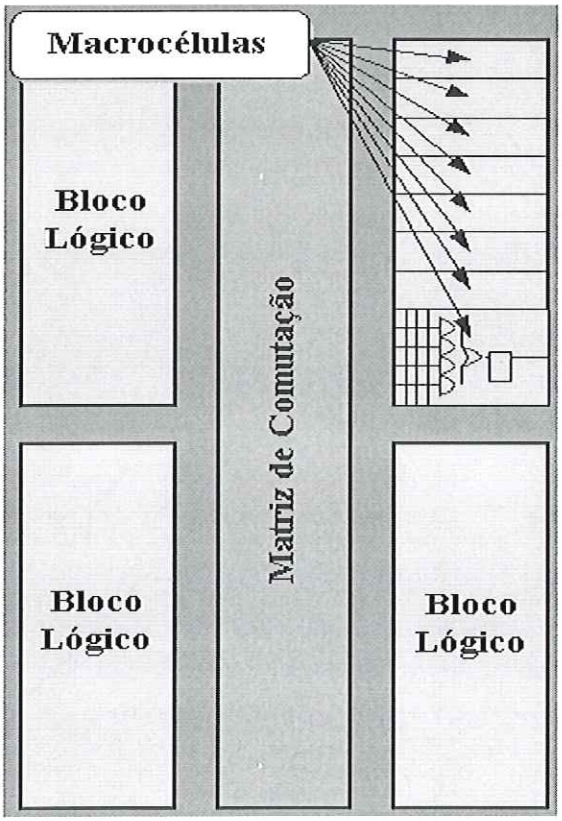


FIGURA 2.8 – Arquitetura básica de um CPLD.

As principais diferenças entre as arquiteturas de CPLDs são: número de termos-produto por macrocélula, se estes podem ser emprestados para outras macrocélulas e se a matriz programável de comutação global pode ser totalmente ou parcialmente conectada. Em algumas arquiteturas, quando o número de termos-produto solicitados excedem a capacidade de uma macrocélula, termos adicionais de outras macrocélulas são emprestados, logo, dependendo da arquitetura, estas macrocélulas não podem ser mais utilizadas. Isto aumenta o atraso de propagação interno. Uma outra diferença nas arquiteturas é o número de conexões dentro da matriz programável de comutação. O número de conexões nesta implicará o quão fácil um projeto se ajustará num dado dispositivo. Para uma matriz de comutação com todas as possibilidades de conexões, um projeto será roteado mesmo com a maioria dos recursos sendo usados e após os pinos de I/O já terem sido fixados. Neste tipo de arquitetura os atrasos internos são fixos e previsíveis. Para as arquiteturas onde a matriz programável é parcialmente povoada, haverá problemas com projetos complexos. Também, será difícil realizar uma mudança no projeto mantendo os pinos de I/O fixados, logo, isto implicará numa mudança de *layout* do circuito. Neste caso, os atrasos internos não serão facilmente previsíveis. Na subseção seguinte, tem-se um exemplo comercial de CPLD. A arquitetura escolhida foi à da série MAX 7000 da Altera por ser uma das mais popularizadas e possuir características similares à outras arquiteturas categorizadas como CPLDs.

2.4.1 Família "MAX 7000" da Altera

A arquitetura geral da série MAX 7000 da Altera está ilustrada na figura 2.9. Ela é formada por um arranjo de blocos chamados LABs (*Logic array blocks*), blocos de entrada e saída (I/Os) e um arranjo de interconexão programável denomidado PIA (*Programmable interconnect array*). O PIA é capaz de conectar qualquer entrada e saída de um LAB a qualquer outro LAB. Também, os blocos de I/Os são conectados diretamente ao PIA e aos LABs.

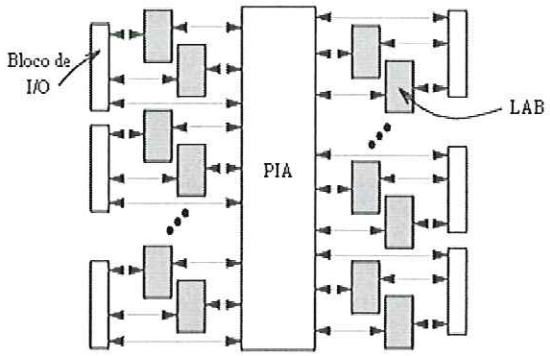


FIGURA 2.9 – Arquitetura da família MAX 7000 da Altera.

Conforme já mencionado, um LAB pode ser imaginado como sendo uma estrutura complexa similar a um SPLD, dessa forma, o *chip* é constituído por um arranjo de SPLDs. As tecnologias disponíveis para esta série são: EPROM e EEPROM. Em 1996, a Altera disponibilizou a série 7000S com recursos de reprogramação no próprio circuito. A arquitetura de um LAB pode ser vista na figura 2.10.

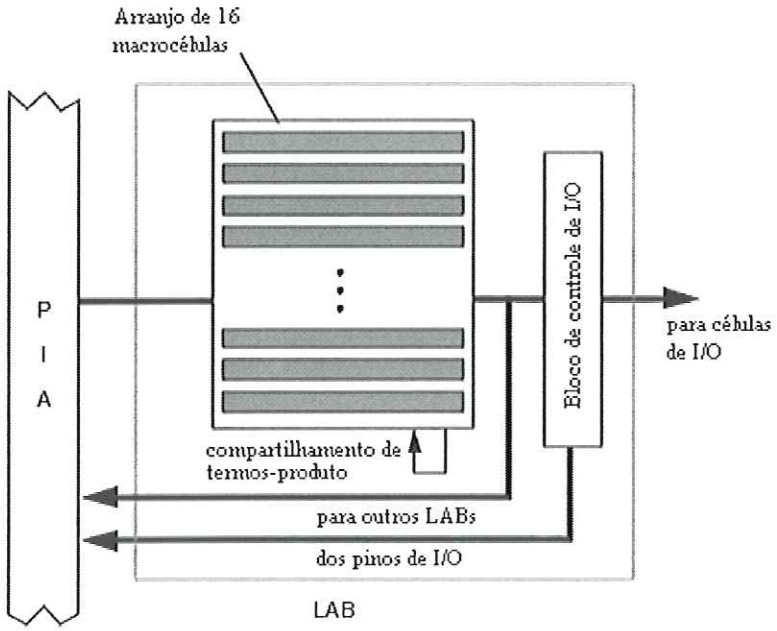


FIGURA 2.10 – Arquitetura de um LAB.

Para cada LAB há dois conjuntos de oito macrocélulas, onde cada macrocélula é formada por termos-produto programáveis que alimentam uma porta OR e um flip-flop. Os *flip-flops* podem ser configurados como: D, JK, T, SR ou podem ser transparentes. Conforme ilustrado na figura 2.11, o número de entradas para a porta OR é variável. A porta OR pode ser alimentada por quaisquer ou todas as saídas dos termos-produto e também pode ter até 15 termos-produto extras de outras macrocélulas dentro do mesmo LAB. Esta flexibilidade permite uma maior eficiência de área, pois as funções lógicas típicas não necessitam mais do que cinco termos-produto e a arquitetura suporta funções mais complexas quando necessário.

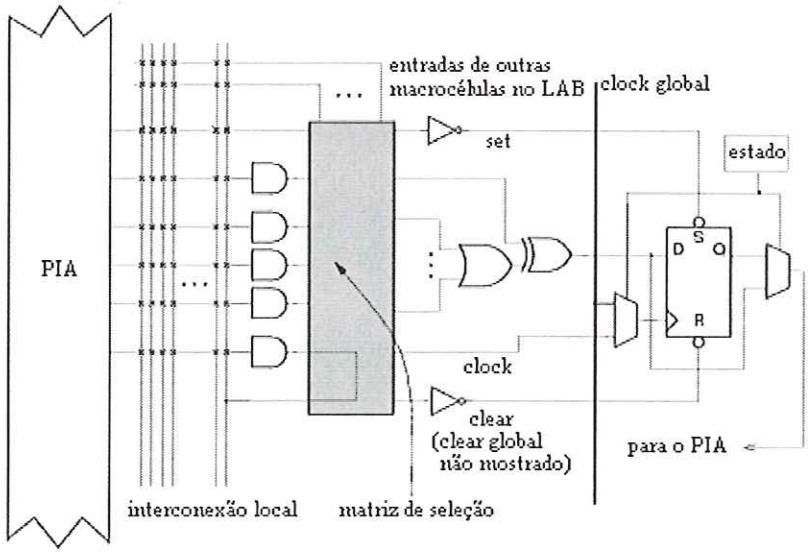


FIGURA 2.11 – Macrocélula da família MAX 7000.

Além da Altera, diversos fabricantes produzem dispositivos categorizados como CPLDs. Entre eles, pode-se citar: Xilinx, Lattice, AMD e ICT. Apesar das diferenças arquiteturais de cada fabricante, os dispositivos possuem características similares entre si.

2.4.2 Aplicações de CPLDs

Os CPLDs fornecem um caminho de migração natural para projetistas de SPLDs que procuram altas densidades. Permitem uma grande integração de circuitos lógicos, reduzindo-se assim o consumo de energia e aumentando-se a confiabilidade do sistema. Devido ao seu bom desempenho de velocidade pino a pino, geralmente são utilizados em projetos orientados a controle. As várias entradas de suas macrocélulas faz com que sejam adequados para máquinas de estados complexas e de alto desempenho. Também podem ser usados para prototipação de sistemas digitais, controladores de vídeo, controladores de rede, UARTs, controle de *cache*, entre outros. Enfim, com CPLDs é possível realizar a implementação de circuitos complexos que explorem um grande número de portas AND/OR e que não necessitem de um número grande de *flip-flops*. A utilização de CPLDs com recursos de reprogramação no sistema permite reconfigurar o hardware sem a necessidade de seu desligamento. Como exemplo, pode-se modificar um protocolo para um circuito de comunicação sem o desligamento do sistema.

2.5 FPGAs

Os FPGAs combinam a arquitetura dos *gate arrays* com a programabilidade dos PLDs. Um FPGA é formado basicamente por um arranjo de blocos lógicos, circundados por blocos de I/Os programáveis, e são conectados via interconexões programáveis. Sua arquitetura é completamente diferente dos CPLDs e tipicamente oferecem uma maior capacidade lógica. Entretanto, seu desempenho é inferior ao dos CPLDs. Nos FPGAs, a lógica a ser implementada usa múltiplos níveis de portas com números de entradas inferiores aos dos SPLDs. Um bloco lógico de um FPGA pode ser tão simples quanto um transistor ou tão complexo quanto um microprocessador. Com FPGAs é possível implementar-se uma variedade muito grande de funções lógicas combinacionais e sequenciais. A arquitetura de roteamento de um FPGA é formada tipicamente por segmentos metalizados de tamanhos diversificados que podem ser interconectados através de comutadores programáveis eletricamente. A distribuição destes afeta significativamente a densidade e o desempenho obtidos por um FPGA (CHOW et al., 1999a; ROSE et al.,

1993). As tecnologias de programação mais comumente utilizadas para se implementar comutadores programáveis num FPGA são: SRAM e *Antifuse*. Em ambos os casos, o comutador ocupará uma área maior e apresentará resistência e capacitância parasitas mais altas do que as de um típico contato metálico utilizado na *customização* de um MPGA. Quanto à arquitetura, os blocos lógicos de um FPGA se classificam em: blocos de granularidade grossa e blocos de granularidade fina.

Geralmente, as arquiteturas de granularidade grossa são formadas por blocos lógicos grandes contendo dois ou mais *flip-flops* e duas ou mais *look-up tables*. Por exemplo, uma *look-up table* de quatro entradas pode ser imaginada como sendo uma ROM de 16x1. Assim, a tabela verdade de uma função lógica de *K*-entradas é armazenada em uma memória de 2^K x 1. As linhas de endereçamento da memória funcionarão como entradas e a saída fornecerá o valor da função lógica. A vantagem das *look-up tables* é sua alta funcionalidade. Uma LUT (*Look-up table*) pode implementar qualquer função de *K* entradas, assim, há 2^n funções, onde: $n=2^K$. A desvantagem é que se tornam grandes para mais do que cinco entradas. Em (BROWN, 1996), um estudo da complexidade de um bloco lógico de FPGA, assumindo este como uma LUT de *K* entradas, indicou que para se otimizar a área, o bloco deve possuir cerca de 4 entradas. Também, para se obter um bom desempenho de velocidade, uma LUT com cinco entradas representa a melhor solução. A desvantagem dos blocos de granularidade grossa é o desperdício de seus recursos para se implementar funções mais simples.

As arquiteturas de granularidade fina contém um grande número de blocos lógicos mais simples. Estes blocos geralmente são formados por portas lógicas básicas ou multiplexadores de quatro entradas e um flip-flop. A principal vantagem de se usar blocos de granularidade fina é que estes são melhor aproveitados. Sua principal desvantagem é que exigem um número muito grande de segmentos e comutadores programáveis para as interconexões. É preciso mais células lógicas para se implementar uma função que seria implementada com poucas células em uma arquitetura com blocos de granularidade grossa. Logo, isto implicará em atrasos e desperdício de área.

As arquiteturas de FPGAs podem ser classificadas genericamente em quatro categorias disponíveis comercialmente: arranjo simétrico, baseada em linhas, PLD hierárquico e mar de portas. Estas categorias se diferenciam pelo tipo de tecnologia de programação utilizada, na arquitetura dos blocos lógicos e na estrutura da arquitetura de roteamento. A figura 2.12 ilustra as quatro classes arquiteturais de FPGAs referidas.

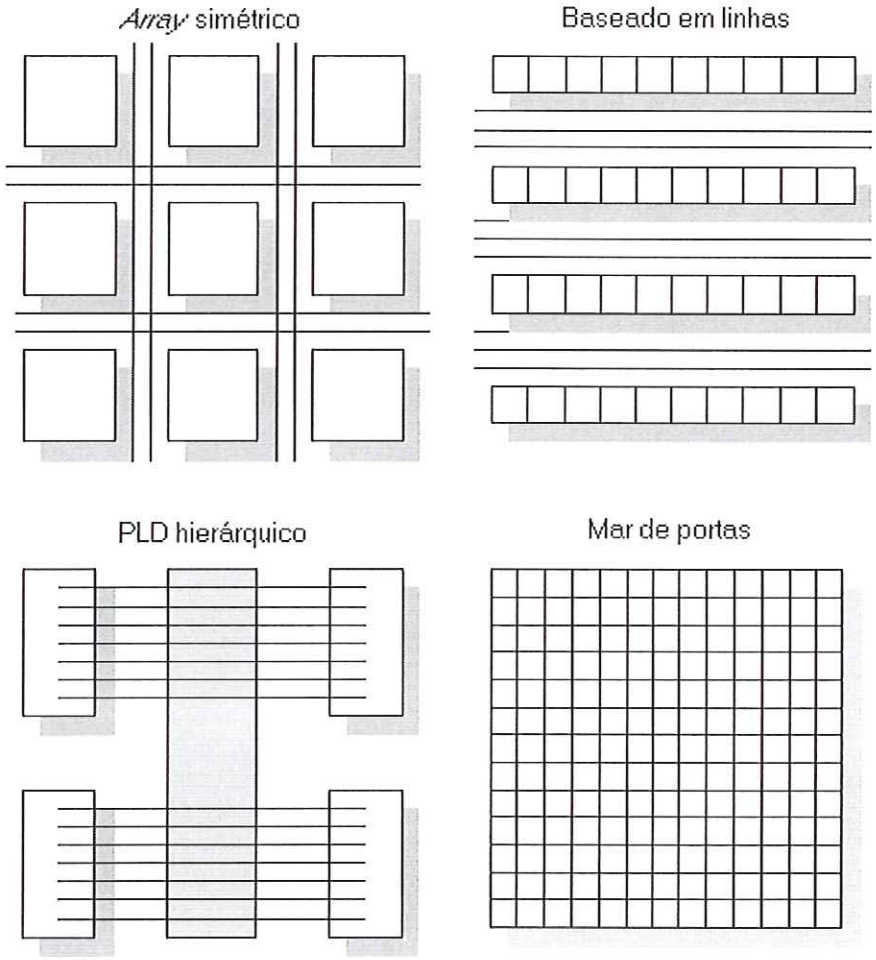


FIGURA 2.12 – Classes arquiteturais de FPGAs.

Na subseção seguinte, tem-se um exemplo prático de uma arquitetura comercial de FPGA. A arquitetura a ser descrita será à da família XC4000 da Xilinx, considerando-se que é uma das mais popularizadas.

2.5.1 Série XC4000 da Xilinx

A estrutura básica dos FPGAs da Xilinx é baseada em *array*, significando que cada *chip* é constituído por um arranjo bidimensional de blocos lógicos que podem ser interconectados por canais horizontais e verticais, de acordo com a figura 2.13.

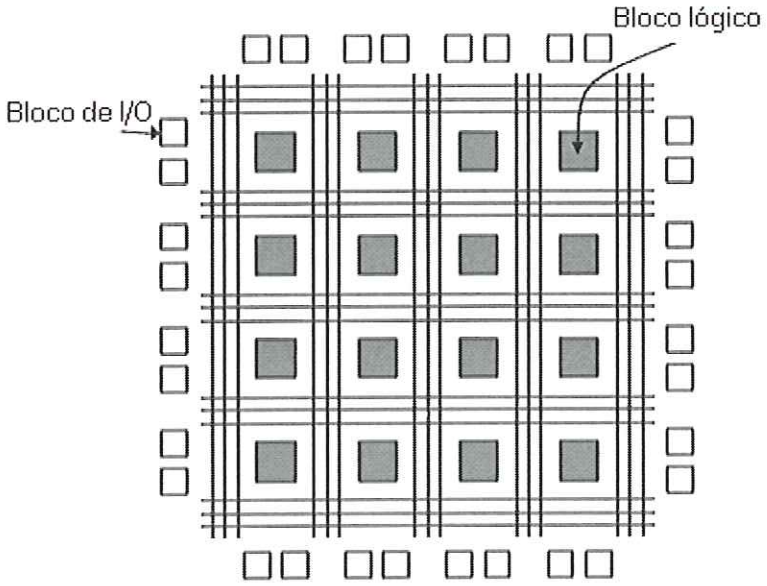


FIGURA 2.13 – Estrutura de um FPGA.

O bloco lógico da série XC4000 é chamado de CLB (*Configurable logic block*) e é baseado em LUTs. Cada CLB contém três LUTs separadas, conforme ilustrado na figura 2.14. Há duas LUTs de quatro entradas que alimentam uma LUT de três entradas. Este arranjo permite a implementação de diversas possibilidades de funções lógicas. Cada CLB contém dois *flip-flops*. Os diferentes tamanhos de LUTs no CLB dão ao bloco uma característica de heterogeneidade, ou seja, permitem um melhor compromisso entre desempenho e densidade lógica. Também, as duas conexões não programáveis das LUTs de quatro entradas para a LUT de três entradas implicam num maior desempenho, considerando-se que não há comutadores programáveis nestas interconexões. Todavia, a inflexibilidade das conexões não programáveis reduz a densidade lógica, uma vez que a LUT de três entradas pode não ser utilizada.

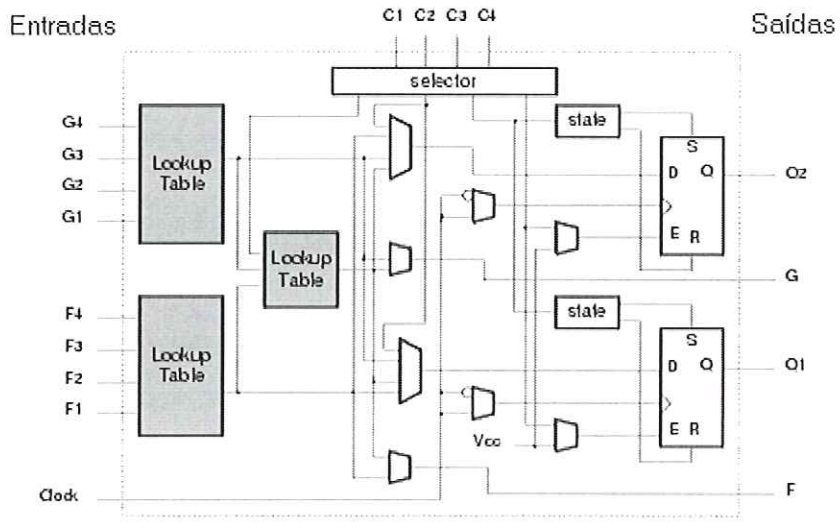


FIGURA 2.14 – CLB da série XC4000 da Xilinx.

Os *chips* da série XC4000 possuem características orientadas a sistemas. Por exemplo, cada CLB contém circuitos que permitem a implementação eficiente de operações aritméticas, e também as LUTs podem ser configuradas como células RAM de leitura/escrita. Além disso, cada *chip* possui planos AND de muitas entradas na periferia do arranjo de blocos lógicos para facilitar a implementação de circuitos como decodificadores. A estrutura de interconexão desta família é disposta em canais horizontais e verticais. Cada canal possui alguns segmentos de trilhas curtos que se estendem por um único CLB, segmentos mais longos que se estendem por dois CLBs e segmentos muito longos que atravessam a largura ou altura total do *chip*. Comutadores programáveis são disponíveis para conectar as entradas e saídas dos CLBs aos segmentos de trilhas, ou conectar um segmento de trilha a outro. A figura 2.15 ilustra uma pequena seção de um canal de roteamento de um dispositivo da série XC4000. Nesta figura, pode-se ver os segmentos de um canal horizontal. Um importante ponto a ser considerado neste tipo de interconexão é que os sinais devem passar por comutadores para alcançar um CLB e o número total de comutadores atravessados depende do conjunto particular de segmentos utilizados. Deste

modo, o desempenho de velocidade de um circuito implementado depende em parte de como os segmentos de trilhas são alocados a sinais individuais pelas ferramentas CAD.

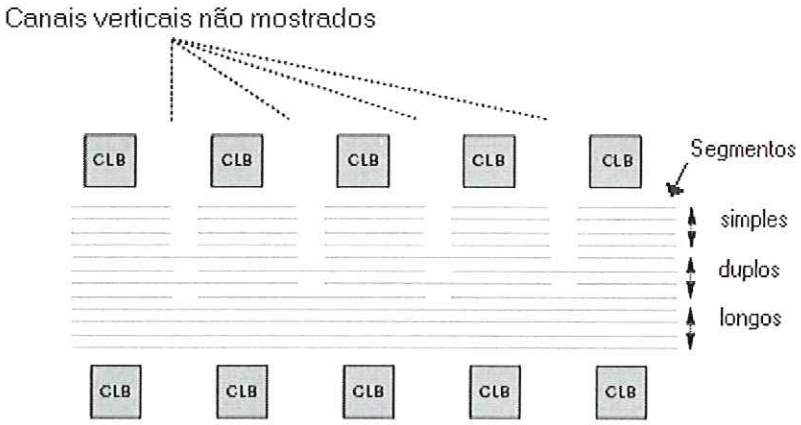


FIGURA 2.15 – Segmentos de trilha da série XC4000.

2.5.2 Aplicações de FPGAs

Os FPGAs podem ser usados em uma faixa muito ampla de aplicações e nas mais distintas áreas. Como exemplos, pode-se destacar: integração de múltiplos SPLDs, controladores de dispositivos, rápida prototipação de projetos para serem posteriormente implementados em *gate arrays*, emulação de grandes sistemas de hardware, coprocessadores, criptografia, reconhecimento de padrões, processamento de imagens, processamento de sinais, compressão de dados, arquiteturas paralelas, sistemas de comunicação, entre outros (BROWN e ROSE, 1996; BOURIDANE et al., 1999; CHAN e MOURAD, 1994; DICK e HARRIS, 1998; GUŠTIN, 1999; HAUCK, 1998; SOLDEK e MANTIUK, 1999; THOMAS et al., 1999; VASSÁNYI, 1997 e VILLASENOR e SMITH, 1997).

2.5.2.1 Computação Reconfigurável

Os sistemas reprogramáveis baseados em FPGAs estão revolucionando o projeto de sistemas computacionais. Sistemas computacionais reconfiguráveis são plataformas cujas arquiteturas podem ser modificadas por software em tempo real para se adequar à uma dada aplicação específica. Para se obter um máximo rendimento, um algoritmo deve ser implementado em hardware. Os sistemas reconfiguráveis se utilizam das partes programáveis dos FPGAs para executar os algoritmos ao invés de compilá-los para a execução numa CPU normal. Estes sistemas tiram vantagem do paralelismo da aplicação enquanto reduzem o *overhead* gerado pelas operações de carga, armazenamento, desvios e decodificação de instruções. De acordo com (TURLEY, 1997), os microprocessadores apresentam diversas limitações e não são adequados para aplicações orientadas a *bits*. Por serem elementos destinados a executar processamento genérico, não são tão eficientes quanto um *chip* ASIC para realizar uma tarefa específica. Dessa forma, o hardware reconfigurável combina a versatilidade dos microprocessadores com o desempenho do hardware dedicado. Um microprocessador é fixado na dimensão espacial e flexível na temporal, assim, suas capacidades nunca mudam, mas a parte do *chip* em uso muda no decorrer do tempo para executar uma diferente função. Sistemas de computação reconfigurável são flexíveis em ambas as dimensões. Logo, sua lógica pode mudar no transcorrer do tempo para se adequar à necessidade em questão. Portanto, sistemas de computação reconfigurável são máquinas que se aproveitam dos aspectos reconfiguráveis dos FPGAs para implementar um algoritmo. Todavia, este novo paradigma requer um modo diferente de se pensar para resolver os problemas, assim, confunde e mistura os limites de hardware e de software. Por se tratar de um novo campo, ainda há muitos obstáculos a serem contornados para que esta tecnologia possa ser amplamente adotada, deste modo, há a necessidade do surgimento de arquiteturas mais sofisticadas para FPGAs e, segundo (ZHANG e NG, 2000), do aparecimento de novas ferramentas de desenvolvimento que possam fazer a ponte entre a especificação de um projeto e sua realização em hardware para sistemas de reconfiguração dinâmica.

2.6 Processo de Projeto para PLDs

Embora os primeiros projetos para PLDs eram gerados em grande parte manualmente, a complexidade dos dispositivos atuais exige o emprego de ferramentas CAD. Nesta seção, objetiva-se fornecer uma visão geral sobre os passos e as operações necessárias para se realizar um projeto num PLD. Na figura 2.16, pode-se ver a sequência típica de operações necessárias para ir desde a descrição de um projeto até a programação do *chip*. Os fornecedores comerciais de ferramentas CAD e de companhias de PLDs oferecem ferramentas apropriadas para projetos. Entretanto, as ferramentas EDAs (*Electronic design automation*) tradicionais não suportam todas as etapas do processo de projeto, fornecem-se somente opções de entrada, operações de simulação e uma interface para ferramentas específicas de posicionamento e de roteamento do chip.

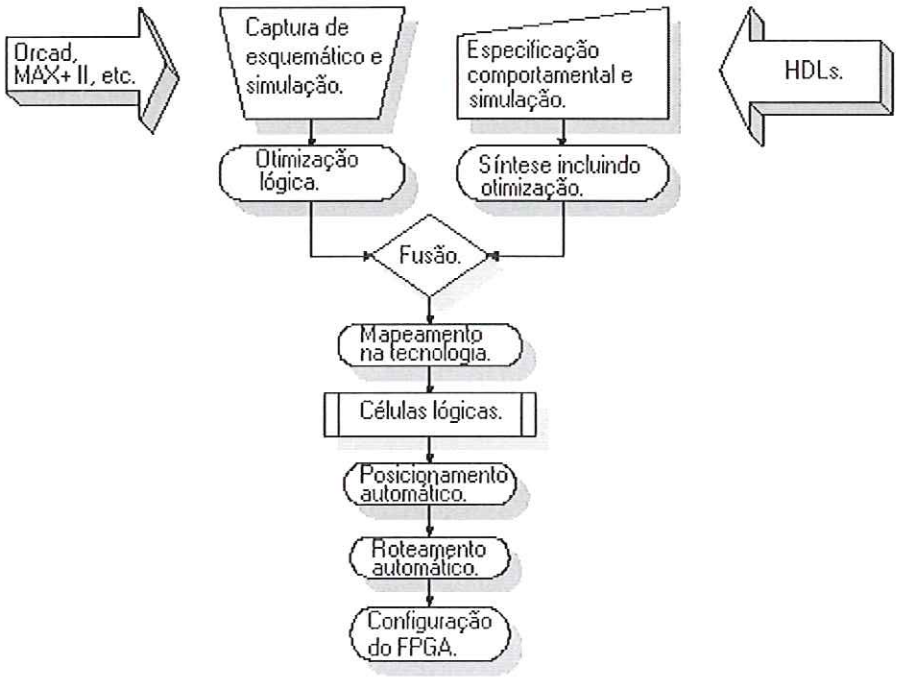


FIGURA 2.16 – Processo de projeto para PLDs.

Há uma variedade muito grande de ferramentas disponíveis para a realização do processo de entrada de um projeto, conforme ilustrado pelas setas da figura 2.16. Alguns projetistas preferem usar pacotes de captura de esquemático enquanto outros preferem especificar um projeto através de linguagens de descrição de hardware (HDLs) ou misturando-se ambos os métodos. Uma outra alternativa para a entrada de um projeto, consiste na utilização de diagramas de tempo contendo formas de onda desejadas para as entradas e para as saídas. Também, é possível se utilizar *cores* para a entrada de um projeto, reduzindo-se assim o tempo de desenvolvimento e de verificação. *Cores* são funções pré-definidas especificamente implementadas e verificadas em lógica programável. Independentemente do método utilizado para a entrada do projeto, o próximo passo consiste-se em converter o projeto em um formato padrão que seja processado por uma ferramenta de otimização lógica. O objetivo da otimização lógica é minimizar as expressões booleanas e eliminar redundâncias, portanto, minimizando-se a área do circuito final. Também nesta etapa, realiza-se uma verificação do projeto. Em seguida, o software mapeia as portas lógicas básicas em blocos lógicos do PLD de destino. Após a lógica ter sido particionada em vários blocos, o software procura pela melhor localização para posicionar os blocos lógicos entre todas as possibilidades. O objetivo principal é reduzir a quantidade de recursos de roteamento requeridos e maximizar o desempenho do sistema. Após o posicionamento, um roteador aloca os segmentos e comutadores disponíveis para a interconexão dos blocos lógicos. Quando o processo de posicionamento e de roteamento é completado, o software cria um arquivo de programação binário necessário para a configuração do dispositivo. A verificação de um projeto ocorre em vários níveis e passos durante as etapas de implementação. É possível realizar uma simulação funcional em conjunto com o processo de entrada, mas antes do posicionamento e do roteamento, para se verificar o funcionamento lógico. Depois de completados os processos de posicionamento e de roteamento, valores exatos de tempos podem ser utilizados para determinar o desempenho do chip. Também é possível obter-se resultados de simulações funcionais mais precisos, pois são incorporadas informações de atrasos ao arquivo *netlist*. Alternativamente, utilizando-se técnicas de verificação no sistema, o projeto pode ser verificado em alta velocidade.

2.7 Considerações Finais

Inicialmente os PLDs eram usados somente para prototipação de projetos digitais. Com o amadurecimento destes dispositivos e das ferramentas CAD, os PLDs já são utilizados em diversas aplicações complexas. Atualmente já se dispõe de dispositivos lógicos programáveis com mais de 4 milhões de portas lógicas, com blocos de DSP (*Digital Signal Processing*) e de memória interiores (ALTERA, 2003). Assim, levando-se também em consideração o baixo custo destes dispositivos, a possibilidade de programação e reprogramação em campo pelo usuário, e o curto ciclo de projeto em relação a outras metodologias tradicionais, pode-se esperar que a lógica programável se consolidará como a forma dominante de projeto e de implementação de sistemas digitais. Além disso, à medida que as arquiteturas e as ferramentas CAD evoluírem, as desvantagens dos PLDs em relação aos MPGAs diminuirão (ALI, 1996; BROWN e ROSE, 1996 e SALCIC e SMAILAGIC, 1998). Também é possível que o sucesso dos FPGAs no mundo digital seja reproduzido no domínio analógico com os FPAAs (*Field programmable analog arrays*).

Conforme a tecnologia se escala, a área da lógica decresce, entretanto, o efeito relativo dos atrasos devido às conexões aumenta. Portanto, faz-se necessário o surgimento de novas estruturas de roteamento. Também, através do desenvolvimento de arquiteturas com blocos lógicos não homogêneos é possível se obter um melhor compromisso entre área e desempenho. É importante também ao projetista conhecer a arquitetura do dispositivo a ser utilizado para que se possa otimizar o desempenho do sistema. Além disso, uma boa descrição de projeto pode otimizar a sintetização lógica.

Enquanto os sistemas reconfiguráveis oferecem possibilidades de implementações de alto desempenho, ainda há muitos problemas a serem resolvidos. O software de mapeamento é muito lento e possui uma baixa qualidade. Também a velocidade dos chips reconfiguráveis é mais lenta que à dos microprocessadores. Com o aparecimento de FPGAs que possam ser reconfigurados em tempo real, com unidades de processamento aritmético, memórias e outros blocos de propósitos especiais, pode-se acreditar que estes dispositivos

se tornarão a pedra fundamental de muitos sistemas computacionais no futuro. Atualmente já existem dispositivos lógicos programáveis ópticos, assim, com o amadurecimento destes será possível reconfigurar sistemas dinâmicos em tempo real.

2.7.1 CPLDs x FPGAs

Os dispositivos lógicos programáveis podem ser classificados basicamente quanto ao número de portas lógicas e quanto ao tipo de estrutura de interconexão utilizada. A figura 2.17 fornece um resumo dos tipos de PLDs mais comumente encontrados comercialmente (ALTERA, 2002).

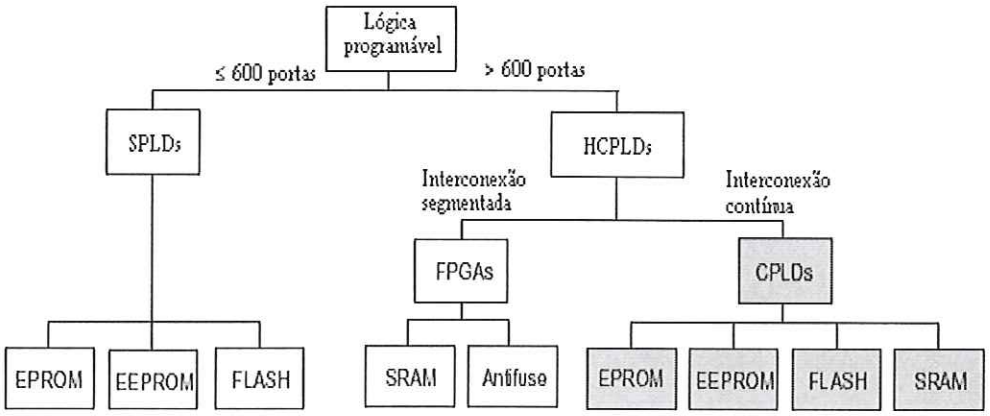


FIGURA 2.17 – PLDs comerciais.

Os CPLDs e os FPGAs utilizam diferentes estruturas de interconexão. Os CPLDs utilizam uma estrutura de interconexão contínua enquanto os FPGAs utilizam uma estrutura de interconexão segmentada, conforme a figura 2.18. Considerando-se que as resistências e capacitâncias de todas interconexões num CPLD são fixas, os atrasos entre quaisquer duas células lógicas no dispositivo são previsíveis. No caso de um FPGA, o número de segmentos necessários para interconectar os sinais não é constante e nem previsível. Então, os atrasos não podem ser quantificados até que o posicionamento e o roteamento tenham

sido completados. As estruturas de interconexão afetam as seguintes características do dispositivo: previsibilidade de desempenho, desempenho no sistema e utilização lógica.

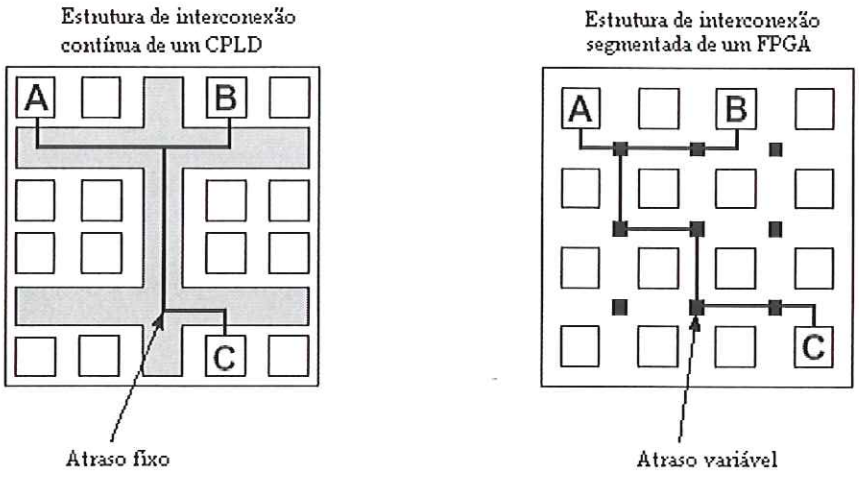


FIGURA 2.18 – Esquema de roteamento de CPLDs e de FPGAs.

Em suma, se comparados a FPGAs, CPLDs oferecem atrasos de interconexão previsíveis, alto desempenho, eficiente utilização lógica e podem ser beneficiados por avanços no processo tecnológico. Também, o software de desenvolvimento para CPLDs possui as seguintes vantagens: síntese lógica automática e tempos menores de compilação. Entretanto, a escolha ótima entre um dispositivo e outro para um projeto fica subordinada à aplicação.

2.7.2 Vantagens da Programação em Sistema (ISP)

PLDs com recursos de programação no sistema podem ajudar a acelerar o tempo de desenvolvimento de um sistema digital e simplificar o fluxo de produção, logo, os custos de projeto podem ser minimizados. Também, ao contrário de PLDs tradicionais que requerem o uso de soquetes, PLDs com tecnologia ISP podem ser soldados diretamente na placa, evitando-se assim possíveis danos ao chip e acelerando-se o ciclo de projeto. Ainda,

é possível programar diversos chips simultaneamente usando a mesma interface. Enquanto o projeto do chip é realizado, a placa de circuito impresso pode ser fabricada em paralelo, reduzindo-se deste modo o tempo de produção. Dispositivos ISP com suporte de migração vertical dão aos projetistas uma flexibilidade adicional, permitindo-se migrar para dispositivos mais densos sem a necessidade de se modificar a placa. Comparando-se a PLDs tradicionais, o tempo para se completar o ciclo de projeto pode ser reduzido em até 50%. Além disso, PLDs tradicionais requerem um programador específico que pode custar em média 4000 dólares.

A implementação da tecnologia ISP pode ser realizada, entre outras opções, através de uma interface denominada JTAG (*Joint test action group*), desenvolvida nos anos 80 segundo o padrão IEEE 1149.1-1990. Esta interface permite testar e programar um PLD pela porta serial ou paralela. Sua conexão com o chip é feita basicamente por quatro pinos dedicados.

Em suma, dispositivos que suportam a tecnologia ISP oferecem muitas vantagens sobre PLDs tradicionais. Os custos são menores e o tempo necessário para executar o ciclo de projeto é reduzido significantemente (ALTERA, 1998).

3 MORFOLOGIA MATEMÁTICA BINÁRIA

3.1 Considerações Iniciais

A palavra *morfologia* é composta das palavras gregas morphê (forma) e logos (ciência) (FACON, 1996), significando o estudo da forma ou estrutura. Em linguística a morfologia estuda a estrutura das palavras. Segundo (GONZALEZ e WOODS, 1992), em biologia, a morfologia denota uma área que lida com a forma e a estrutura de animais e plantas. Por exemplo, a forma de uma folha pode ser usada para identificar uma planta.

A morfologia matemática teve suas origens em 1964 nas pesquisas conjuntas de G. Matheron e J. Serra. Neste mesmo período foi criado o centro de morfologia matemática na Escola de Minas de Paris em Fontainebleu (França) (DOUGHERTY, 1992; SERRA, 1982; WEEKS Jr., 1996).

A morfologia matemática é uma área não linear em processamento de imagens, podendo ser utilizada como uma ferramenta para extração de componentes de imagens que sejam úteis na representação e na descrição da forma de uma determinada região, como por exemplo, fronteiras, esqueletos e o fecho convexo. Também pode ser usada para pré e pós-processamento, como por exemplo, filtragem morfológica, afinamento e poda. Pode ser aplicada em diversas áreas, entre elas, biologia, metalografia, medicina, visão robótica, controle de qualidade, reconhecimento de padrões, entre outras. A aplicação dos conceitos de morfologia matemática a problemas de visão por computador segue sendo um tema de investigação em nível mundial. A linguagem da morfologia matemática é a teoria de conjuntos. Os conjuntos em morfologia matemática representam as formas dos objetos numa imagem. Como exemplo, pode-se considerar todos os pixels pretos em uma imagem

binária como sendo uma descrição completa dessa imagem. Tais conjuntos são membros do espaço bidimensional de números inteiros Z^2. Cada elemento do conjunto consiste-se de um vetor bidimensional cujas coordenadas são as coordenadas (x, y) dos pixels pretos (por convenção) da imagem. Na figura 3.1, pode-se ver um exemplo (COSTA & CÉSAR Jr., 2000; DOUGHERTY, 1992; FACON, 1996; GONZALEZ e WOODS, 1992; HARALICK et al.,1987; MARQUES FILHO e VIEIRA NETO, 1999; WEEKS Jr., 1996).

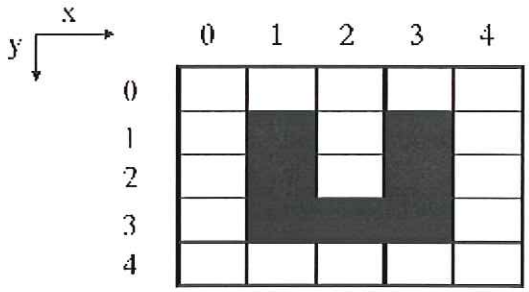


FIGURA 3.1 – Descrição de uma imagem binária através do conjunto formado por seu sistema de coordenadas. O objeto em forma de U pode ser representado completamente pelo conjunto A={(1,1),(1,2),(1,3),(2,3),(3,3),(3,2),(3,1)}.

Imagens digitais em níveis de cinza podem ser representadas por conjuntos cujos componentes estejam em Z^3. Nesse caso, dois componentes de cada elemento do conjunto se referem às coordenadas do pixel enquanto o terceiro corresponde ao valor discreto de intensidade. Conjuntos em espaços de maiores dimensões podem conter outros atributos de imagens, como por exemplo, cor e componentes que variem com o tempo.

3.2 Dilatação e Erosão

3.2.1 Definições Básicas

Antes de se definir as operações básicas morfológicas de dilatação e de erosão é importante relembrar algumas definições básicas referentes à teoria de conjuntos. Logo, pode-se destacar:

3.2.1.1 Translação

Sejam A e B conjuntos de Z^2 com componentes $a=(a_1,a_2)$ e $b=(b_1,b_2)$. A translação de A por $x=(x_1,x_2)$ é definida como:

$$(A)_x = \{c \mid c = a + x, \textit{para } a \in A\} \qquad (3.1)$$

3.2.1.2 Reflexão

A reflexão de B é definida como:

$$\hat{B} = \{x \mid x = -b, \textit{para } b \in B\} \qquad (3.2)$$

3.2.1.3 Complemento

O complemento do conjunto A é definido como:

$$A^c = \{x \mid x \notin A\} \qquad (3.3)$$

3.2.1.4 Diferença

A diferença entre dois conjuntos A e B é definida como:

$$A - B = \{x \mid x \in A, x \notin B\} = A \cap B^c \qquad (3.4)$$

Na figura 3.2 é possível se observar as ilustrações das definições anteriores.

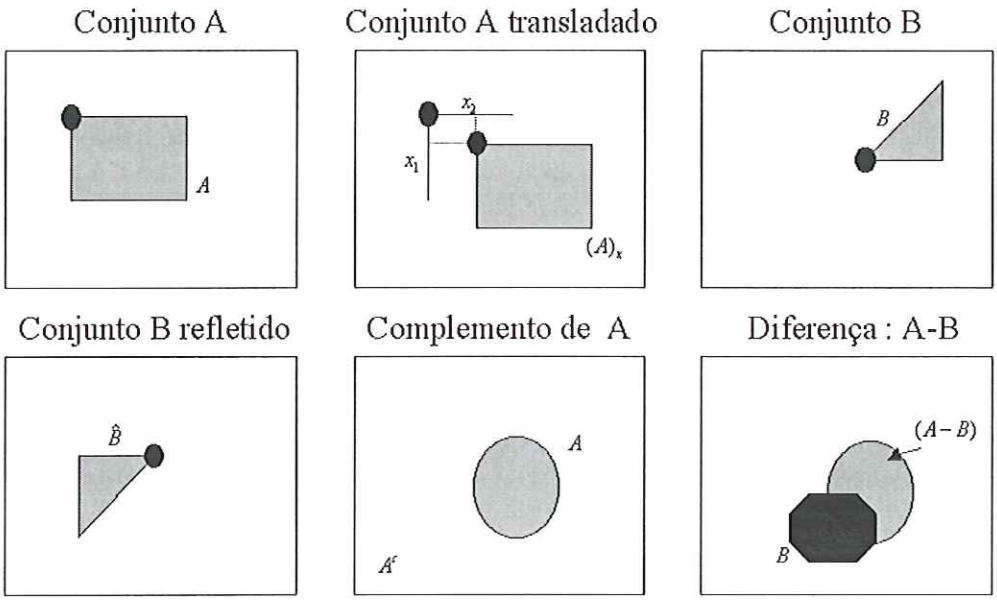


FIGURA 3.2 – Exemplos de operações básicas entre conjuntos.

3.2.2 Dilatação

Sejam A e B conjuntos de Z^2 e $\varnothing$ um conjunto vazio, logo, a dilatação de A por B é definida como:

$$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \varnothing\} \qquad (3.5)$$

Assim, o processo de dilatação começa na obtenção da reflexão de B em torno de sua origem, seguido da translação dessa reflexão por x. A dilatação de A por B é então, o

conjunto de todos os deslocamentos x tais que a reflexão de B e A sobreponham-se em pelo menos um elemento não nulo. A equação (3.5) pode ser reescrita como:

$$A \oplus B = \{x \mid [(\hat{B})_x \cap A] \subseteq A\} \qquad (3.6)$$

O conjunto B da equação (3.6) denota o elemento estruturante da operação morfológica em questão.

Pode-se observar que o processo acima é análogo ao processo de convolução, uma vez que é necessário inverter B em relação à sua origem e deslocá-lo em relação à imagem A (conjunto). Na figura 3.3, tem-se um exemplo ilustrativo.

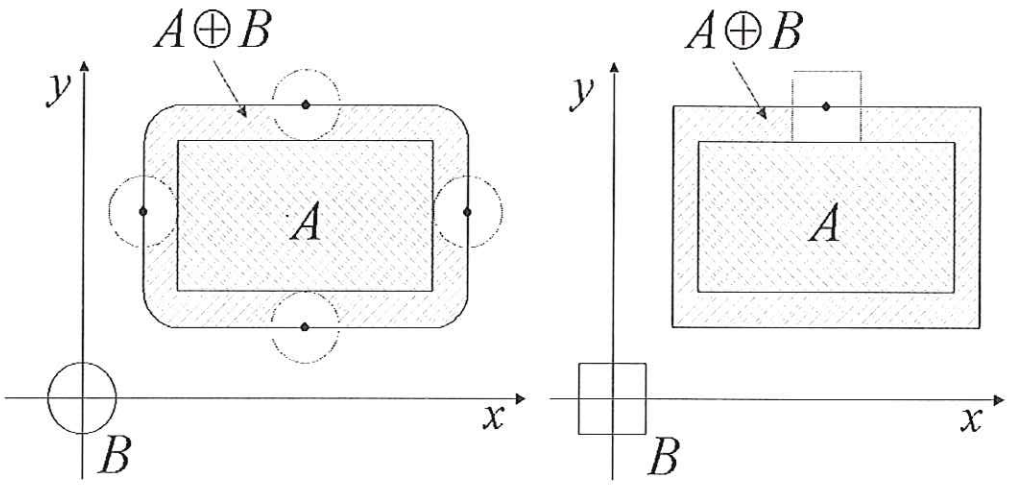


FIGURA 3.3 – Ilustração do processo de dilatação do conjunto A (quadrado) utilizando-se dois elementos estruturantes diferentes.

3.2.2.1 Adição de Minkowski

A adição de Minkowski de dois conjuntos é definida como:

$$A \oplus B = \bigcup_{b \in B} (A)_b \qquad (3.7)$$

Ou seja, na equação (3.7) pode-se observar que em vez de se transladar o elemento estruturante, deve-se transladar a imagem em relação às posições permitidas pelo elemento estruturante. Os deslocamentos são realizados em relação ao ponto central de B. Ao contrário das definições anteriores de dilatação, neste caso não há necessidade de verificação, conseguindo-se assim um maior desempenho computacional.

Exemplo 3.1: Dilatação do conjunto A pelo elemento estruturante B utilizando-se a definição de Minkowski dada pela equação (3.7).

$$A \oplus B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \oplus \begin{Bmatrix} 1 \\ 1 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cup \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Uma vez que a dilatação é uma transformação comutativa, tem-se:

$$A \oplus B = \cup \{B + a \mid a \in A\} \qquad (3.8)$$

Assim sendo, pode-se efetuar a união de todos os deslocamentos de B em relação aos elementos válidos de A.

Exemplo 3.2: Na figura 3.4 tem-se um exemplo ilustrativo da dilatação do conjunto A pelo elemento estruturante B utilizando a equação (3.8).

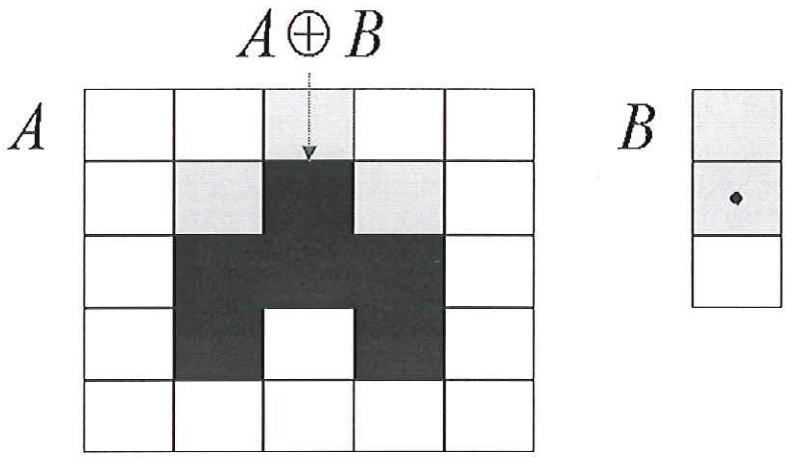


FIGURA 3.4 – Ilustração do processo de dilatação utilizando a equação (3.8). Em cinza, para efeitos didáticos, tem-se os pixels adicionados após a operação de dilatação.

Outras definições de dilatação podem ser encontradas em (FACON, 1996 e SERRA, 1986), entretanto, todas são equivalentes.

3.2.3 Erosão

Sejam A e B conjuntos de Z^2, logo, a erosão de A por B é definida como:

$$A \Theta B = \{x \mid (B)_x \subseteq A\} \qquad (3.9)$$

Deste modo, a erosão de A por B é o conjunto de todos os pontos x, tais que B, quando transladado de x, fique contido em A. Como no caso da dilatação, a equação (3.9) não é a única definição de erosão. Todavia, é usualmente favorecida em implementações morfológicas, assim como a primeira definição de dilatação, equação (3.5), pois apresenta a vantagem de ser mais intuitiva que as outras quando o elemento estruturante B for visto

como uma máscara de convolução. A figura 3.5 ilustra a erosão de um retângulo A por um elemento estruturante circular.

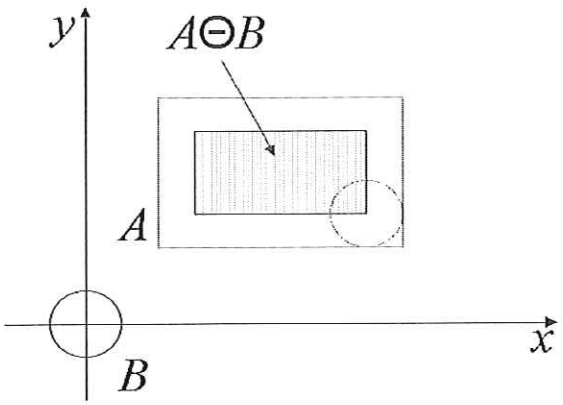


FIGURA 3.5 – Processo de erosão do conjunto A pelo elemento estruturante B. A área hachurada representa o conjunto erodido.

3.2.3.1 Subtração de Minkowski

A subtração de Minkowski de dois conjuntos é definida como:

$$A\Theta(-B) = \cap\{A + b \mid b \in B\} \qquad (3.10)$$

Onde: -B={-b|b∈B} é a rotação de 180° de B em relação à sua origem. A subtração de Minkowski é a erosão pelo elemento estruturante –B. Portanto, da equação (3.10), vem:

$$A\Theta B = \cap\{A - b \mid b \in B\} \qquad (3.11)$$

Exemplo 3.3: Erosão do conjunto A pelo elemento estruturante B utilizando-se a definição dada pela equação (3.11).

$$A\Theta B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Theta \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cap \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Assim como na dilatação dada pela equação (3.7), também na erosão dada pela equação (3.11) o processo é executado sem verificação, obtendo-se assim um melhor desempenho computacional.

3.2.4 Propriedade Dual da Dilatação e da Erosão

A dilatação e a erosão são operações duais em relação à complementação e a reflexão de conjuntos, ou seja:

$$(A\Theta B)^c = A^c \oplus \hat{B} \qquad (3.12)$$

Prova:

$$(A\Theta B)^c = \{x \mid (B)_x \subseteq A\}^c,$$

$$(A\Theta B)^c = \{x \mid (B)_x \cap A^c = \varnothing\}^c, \therefore$$

$$(A\Theta B)^c = \{x \mid (B)_x \cap A^c \neq \varnothing\} = (A^c \oplus \hat{B})$$

3.2.5 Efeitos da Dilatação e da Erosão

A erosão modifica o conjunto original e este fica menor em todos os casos. Faz desaparecer conjuntos inferiores ao elemento estruturante, aumenta furos internos e separa conjuntos.

A dilatação também modifica o conjunto original e este fica maior em todos os casos. Preenche furos inferiores ao elemento estruturante e conecta conjuntos separados.

Na figuras 3.6 e 3.7, pode-se confirmar o que foi sobredito. Os exemplos foram implementados no software Matlab 6.0.

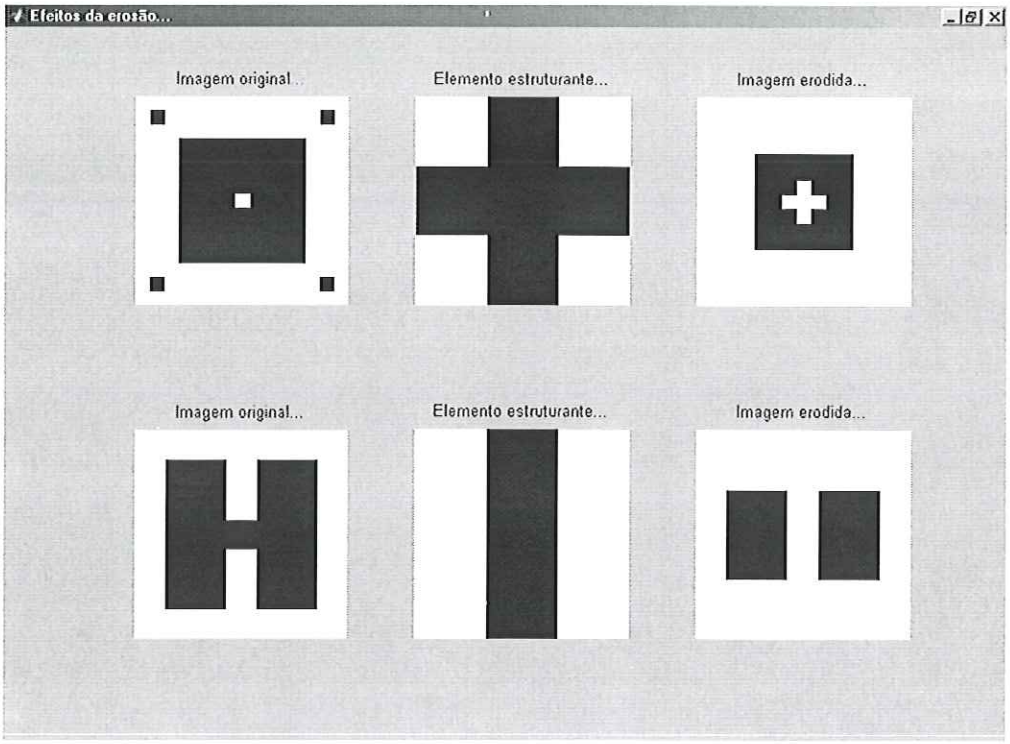


FIGURA 3.6 - Efeitos da erosão por um elemento estruturante cruz.

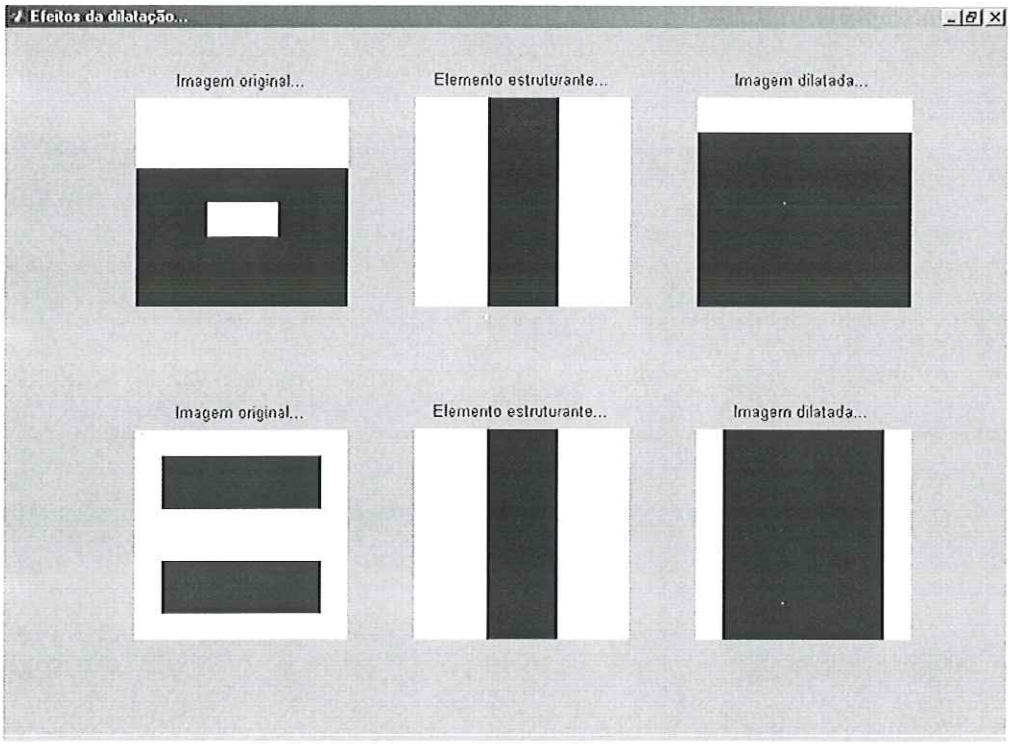


FIGURA 3.7 - Efeitos da dilatação por um elemento estruturante unidimensional.

3.3 Abertura e Fechamento

Como foi visto anteriormente, a dilatação expande uma imagem, enquanto, a erosão a encolhe. A abertura, em geral, suaviza o contorno de uma imagem, quebra istmos estreitos e elimina proeminências delgadas. O fechamento funde pequenas quebras, alarga golfos estreitos, elimina pequenos orifícios e preenche *gaps* no contorno.

A abertura de um conjunto A por um elemento estruturante B, é definida como:

$$A \circ B = (A \Theta B) \oplus B \qquad (3.13)$$

O fechamento de A por B é definido como:

$$A \bullet B = (A \oplus B) \Theta B \qquad (3.14)$$

Na figura 3.8, tem-se um exemplo de operação de abertura e de fechamento, implementado no software Matlab 6.0, para um elemento estruturante circular.

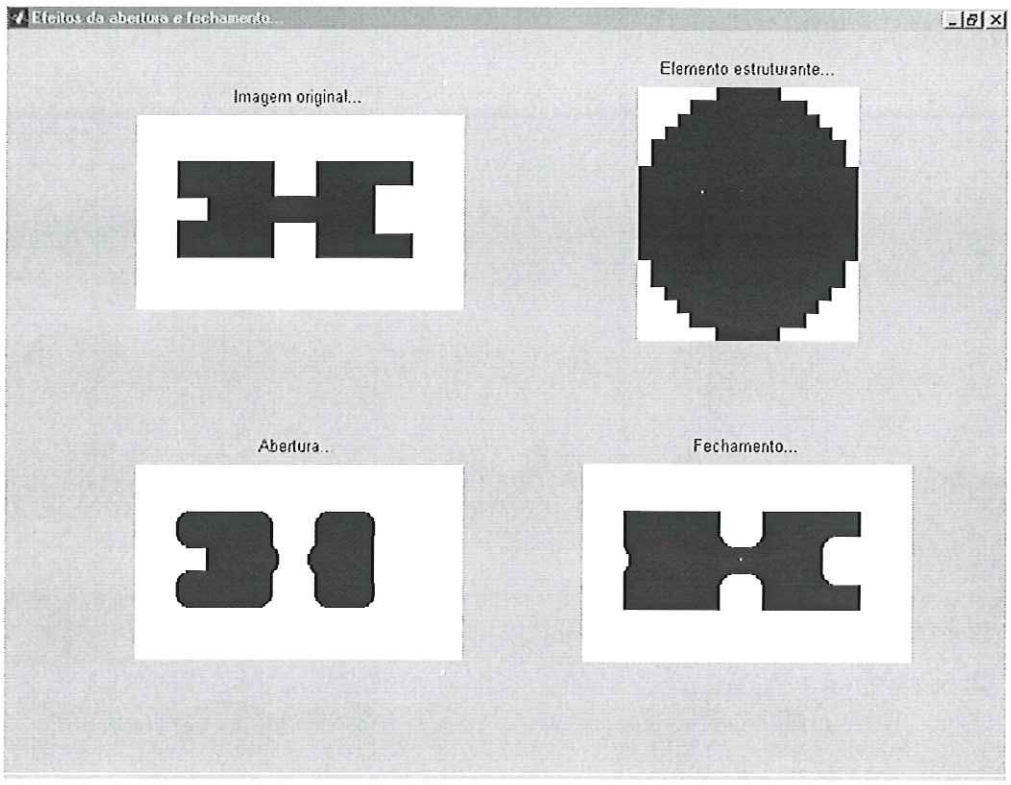


FIGURA 3.8 – Exemplo de abertura e de fechamento binário. A operação de abertura suavizou os contornos da imagem pelo interior enquanto a operação de fechamento suavizou seus contornos pelo exterior.

3.3.1 Interpretações Geométricas da Abertura e do Fechamento

3.3.1.1 Ajuste Geométrico da Operação de Abertura

A abertura de A por B é obtida através da união de todas as translações de B que caibam em A, ou seja, a abertura pode ser expressada como um processo de ajuste, tal que:

$$A \circ B = \cup\{(B)_x \mid (B)_x \subset A\} \qquad (3.15)$$

A figura 3.9 ilustra o processo realizado pela equação (3.15).

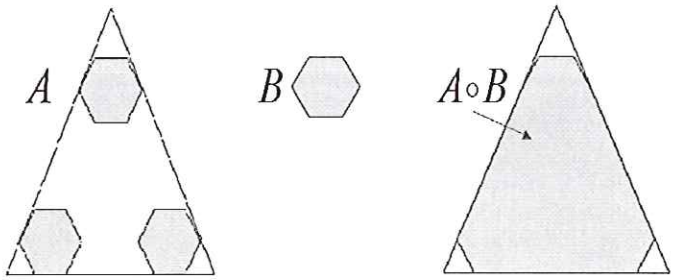


FIGURA 3.9 – Interpretação geométrica da operação de abertura.

3.3.1.2 Interpretação Geométrica da Operação de Fechamento

Geometricamente, um ponto z é um elemento de A fechado por B se, e somente se, $(B)_x \cap A \neq \varnothing$, para qualquer translação de $(B)_x$ que contenha z. A figura 3.10 ilustra essa propriedade.

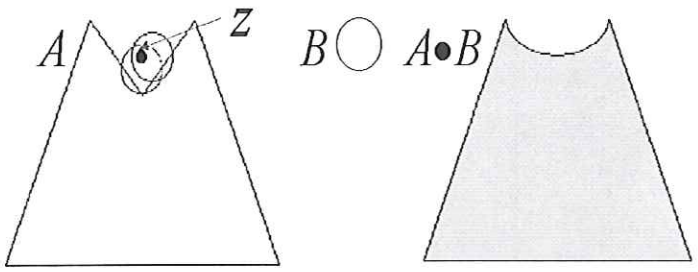


FIGURA 3.10 – Interpretação geométrica da operação de fechamento.

3.3.1.3 Propriedades da Abertura e do Fechamento

Como no caso da dilatação e da erosão, a abertura e o fechamento são duais em relação à complementação e reflexão de conjuntos:

$$(A \bullet B)^c = (A^c \circ \hat{B}) \qquad (3.16)$$

A operação de abertura satisfaz as seguintes propriedades:

- É uma transformação anti-extensiva, ou seja, o resultado da abertura é um subconjunto do conjunto original.
- É uma transformação crescente. Por exemplo, se C está contido em D, então a abertura de C por B estará contida na abertura de D por B.
- É uma transformação idempotente, logo, a abertura de A por B seguida da abertura por B não causará nenhum efeito.

A operação de fechamento satisfaz as seguintes propriedades:

- É uma transformação extensiva, ou seja, o conjunto original é um subconjunto do resultado produzido pelo fechamento.
- Também é uma transformação crescente.
- Também é uma transformação idempotente.

Valendo-se das idéias anteriores, é possível definir um filtro morfológico de acordo com a equação (3.17).

$$(A \circ B) \bullet B \qquad (3.17)$$

A figura 3.11 mostra um exemplo de aplicação para o filtro morfológico definido na equação (3.17). A imagem original, constituída por um retângulo, foi corrompida pelo ruído sal e pimenta gerado através do software Matlab 6.0. O objetivo deste filtro é remover o ruído, contudo, sem distorcer a imagem original.

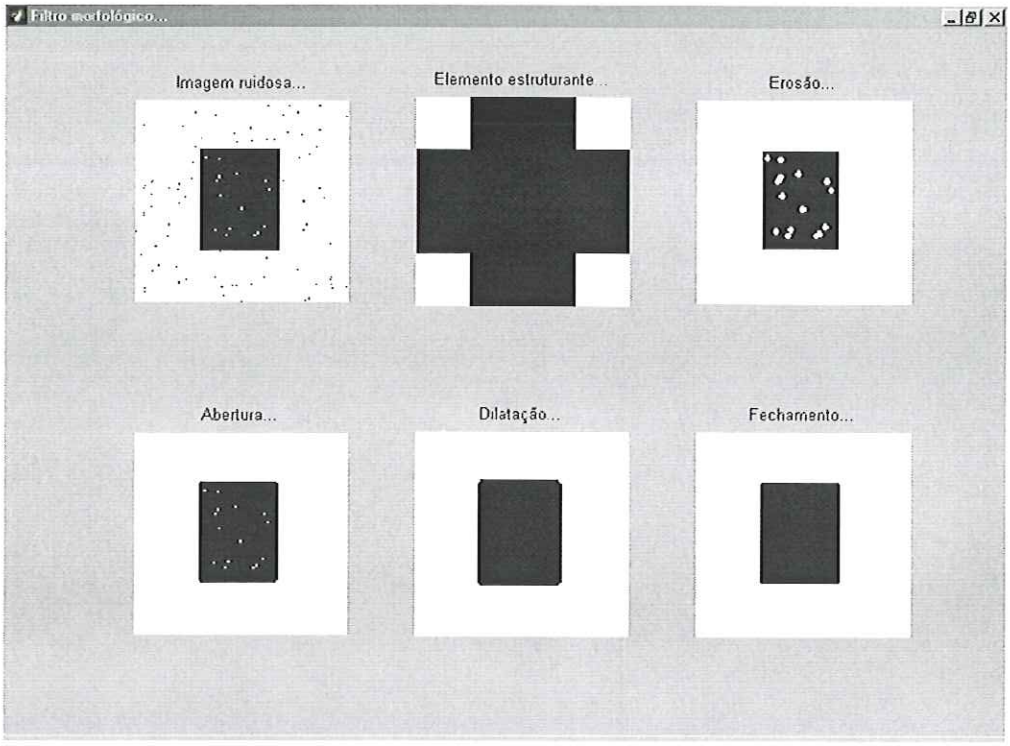


FIGURA 3.11 – Filtragem morfológica. A imagem original, constituída por um retângulo, foi corrompida pelo ruído sal e pimenta gerado pelo software matlab 6.0. Para remover o ruído foi utilizado o processo de filtragem morfológica descrito pela equação (3.17). Utilizou-se para esta finalidade um elemento estruturante cruz de dimensão superior ao maior aglomerado de ruído. Pela figura é possível constatar que o ruído foi totalmente removido e a imagem original foi completamente restaurada.

3.4 Transformada *Hit or Miss*

A transformada hom (*hit or miss*) é uma ferramenta básica para a detecção de formas. É usada para testar ao mesmo tempo as partes internas e externas de um conjunto. É definida como:

$$A \hom B = (A \Theta B_1) \cap (A^c \Theta B_2), B_1 \cap \mathrm{B}_2 = \varnothing \qquad (3.18)$$

O conjunto A hom B, contém todos os pontos nos quais, simultaneamente, B_1 encontrou um casamento (*hit*) em A e B_2 em A^c. A figura 3.12 ilustra o uso da equação (3.18) para detectar a posição do subconjunto X do conjunto A.

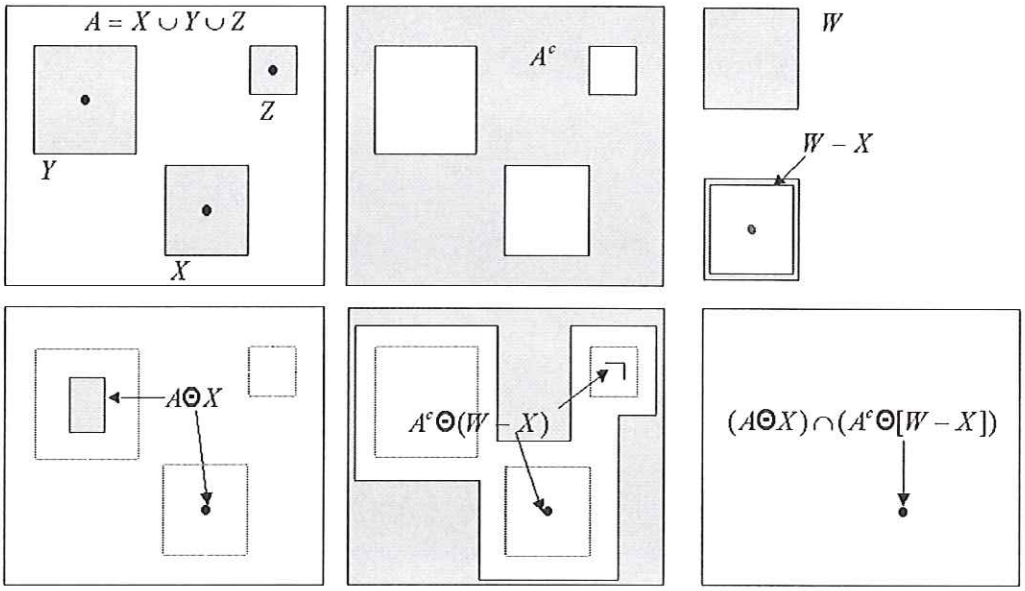


FIGURA 3.12- Exemplo da Transformada *Hit or Miss*. O conjunto A representa a imagem original com três subconjuntos constituintes. O objetivo deste exemplo foi detectar a posição do subconjunto X pertencente ao conjunto inicial. Logo, complementando-se o conjunto A e definindo-se uma janela de dimensões maiores que X, W, então, após subtrair esta de X, obteve-se o elemento estruturante fundo local W-X. Portanto, para se detectar a posição desejada foi realizada uma erosão de A por X e simultaneamente uma erosão de A^c por W-X. Assim, após se efetuar a intersecção destes dois últimos resultados foi possível detectar a coordenada central de X.

3.5 Algoritmos Morfológicos Básicos

Nesta seção serão considerados alguns aspectos práticos para a morfologia matemática. No caso de imagens binárias, a principal aplicação de morfologia é a extração de componentes da imagem que sejam úteis na representação e na descrição de formas. Também, abordar-se-á sobre o mecanismo de funcionamento de alguns algoritmos utilizados para pré e pós processamento.

3.5.1 Extração de Fronteiras

A fronteira de um conjunto A, denotada por β(A), pode ser obtida através da erosão de A por B, seguido da diferença de conjuntos entre A e sua erosão. Ou seja,

$$\beta(A) = A - (A \Theta B) \qquad (3.19)$$

A figura 3.13 ilustra o mecanismo de extração de fronteiras.

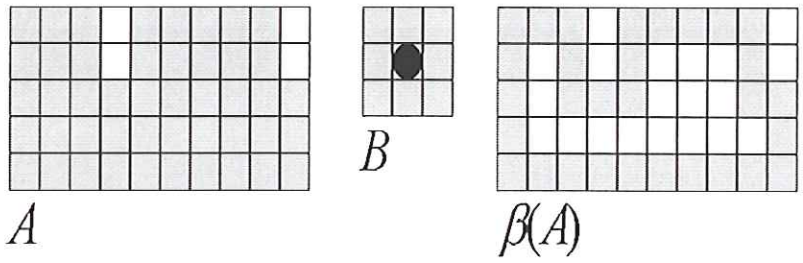


FIGURA 3.13 – Obtenção da fronteira de A segundo a equação (3.19).

3.5.2 Preenchimento de Regiões

Este algoritmo é baseado em dilatação de conjuntos, complementação e interseções. Na figura 3.14, A denota o conjunto contendo um subconjunto cujos elementos são pontos de fronteira conectados-de-8 de uma região. A partir de um ponto p dentro da fronteira, o objetivo é preencher completamente a região. Logo, atribui-se 1 ao ponto p e inicia-se o seguinte procedimento que preencherá a região com 1:

$$X_k = (X_{k-1} \oplus B) \cap A^c, k = 1,2,3... \qquad (3.20)$$

Onde X_o=p e B é o elemento estruturante simétrico mostrado na figura 3.14. O algoritmo converge na iteração k se X_k=X_{k-1}. A união dos conjuntos X_k e A contém o conjunto preenchido e sua fronteira. O processo de dilatação descrito na equação (3.20) preencheria completamente a área se deixasse de ser verificado, entretanto, a intersecção

com A^c a cada passo limita o resultado dentro da região de interesse (dilatação condicional).

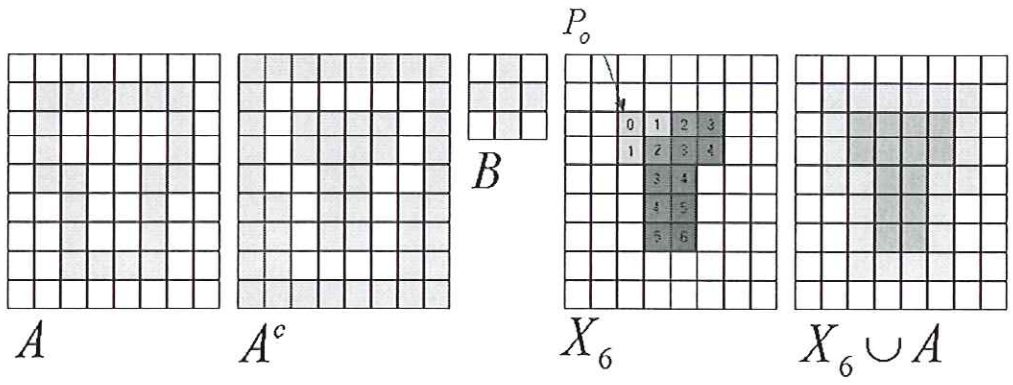


FIGURA 3.14 – Ilustração do processo de preenchimento de regiões descrito pela equação (3.20). Aqui, P_0 denota a semente inicial que crescerá conforme as iterações do algoritmo e preencherá completamente a região. O subconjunto X_6 representa o resultado da última iteração em que não houve mais mudanças em relação à iteração anterior. Os pixels numerados representam as iterações. Logo, a união de A com X_6 determina o resultado final.

3.5.3 Extração de Componentes Conectados

Este procedimento é utilizado em aplicações de análise automática de imagens binárias. Assim, seja Y um componente conectado contido num conjunto A e assuma que um ponto p de Y seja conhecido. Logo, a expressão iterativa dada pela equação (3.21) leva a todos os elementos de Y.

$$X_k = (X_{k-1} \oplus B) \cap A, k = 1,2,3... \qquad (3.21)$$

Na figura 3.15, tem-se um exemplo.

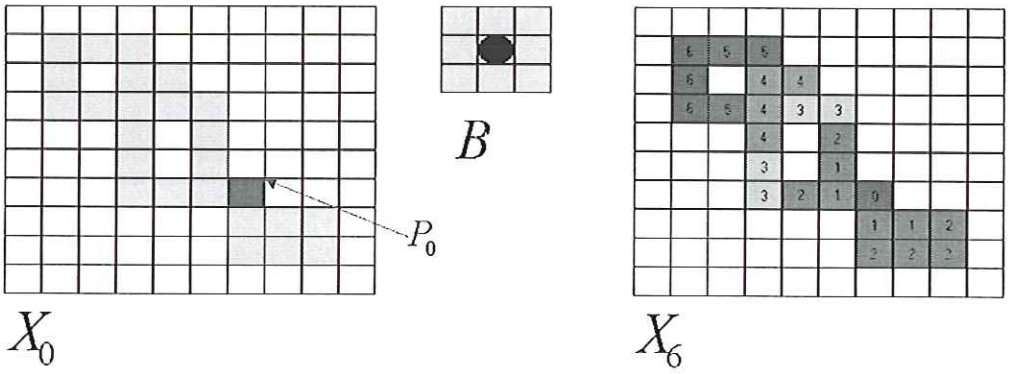


FIGURA 3.15 – Ilustração do algoritmo de extração de componentes conectados. Aqui, P_0 denota o pixel inicial pertencente a fronteira do conjunto original. B é um elemento estruturante adequado. A convergência ocorre em $X_k=X_{k-1}$. Neste exemplo a convergência ocorreu na Sexta iteração. A intersecção com o conjunto original faz com que as dilatações centradas em torno dos componentes rotulados com 0 sejam eliminadas. Isto é feito porque todos os elementos procurados estão rotulados com 1 (componente conectado).

3.5.4 Fecho Convexo

O casco convexo H de um conjunto S é o menor conjunto convexo que ainda contém S. É usado para particionar uma fronteira em segmentos significativos. Algumas vezes é usado para a descrição de objetos (SONKA et al., 1993). O procedimento morfológico para determinar o fecho convexo é:

- Aplique iterativamente a transformada hom a A com B^1; quando não houverem mais mudanças, realize a união com A e chame o resultado de D^1. O procedimento é então repetido com B^2 até que não ocorram mais mudanças, e assim por diante. A união dos quatro D's resultantes constitui o fecho convexo de A. A equação (3.22) apresenta o procedimento para calcular o fecho convexo.

$$X_k^i = (X \text{ hom } B^i) \cup A, i = 1,2,3,4 \text{ e } k = 1,2,3,...$$

$$X_0^i = A,$$

$$D^i = X_{conv}^i,$$

$$C(A) = \bigcup_{i=1}^{4} D^i \qquad (3.22)$$

A figura 3.16 ilustra o procedimento dado pela equação (3.22).

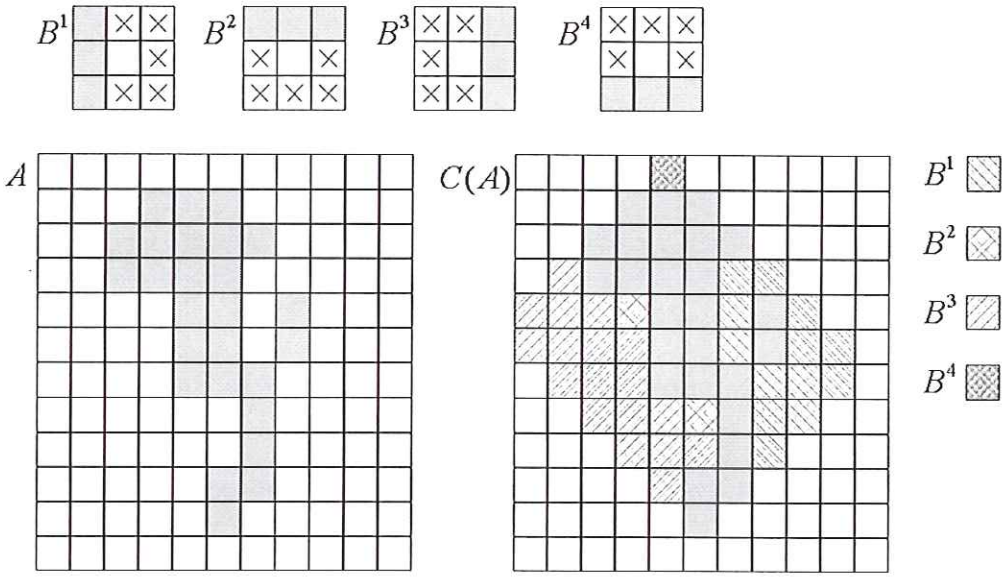


FIGURA 3.16 – Ilustração do algoritmo de fecho convexo. Os B's representam os quatro elementos estruturantes utilizados. Os X's significam uma condição irrelevante, podendo ser 0 ou 1. B^i é uma rotação de B^{i-1} de 90°. C(A) denota o casco convexo do conjunto A mostrando a contribuição de cada elemento estruturante utilizado.

3.5.5 Afinamento

O afinamento de um conjunto A por um elemento estruturante B é definido como:

$$A \otimes B = A - (A \text{ hom } B) = A \cap (A \text{ hom } B)^c \qquad (3.23)$$

A operação dual correspondente do afinamento é o espessamento. Usualmente, para se obter o espessamento, afina-se o fundo da imagem e o resultado é então complementado. Também, os elementos estruturantes usados para o espessamento tem o mesmo formato dos usados para o afinamento, porém, com os 1's e os 0's trocados. Na figura 3.17, pode-se ver um exemplo de afinamento. Neste exemplo, o afinamento do conjunto A é baseado em uma sequência de elementos estruturantes, tais que, $\{B\}=\{B^1,B^2,...,B^n\}$, onde B^i é uma versão rotacionada de B^{i-1}. Assim, $A\otimes\{B\}=((...((A\otimes B^1)\otimes B^2)...)\otimes B^n)$, ou seja, o processo se consiste em afinar o conjunto A por uma passada com B^1, seguida de uma passada por B^2 e assim por diante, até que A tenha sido afinado por uma passada de B^n. Este processo deve ser repetido até a idempotência ser atingida.

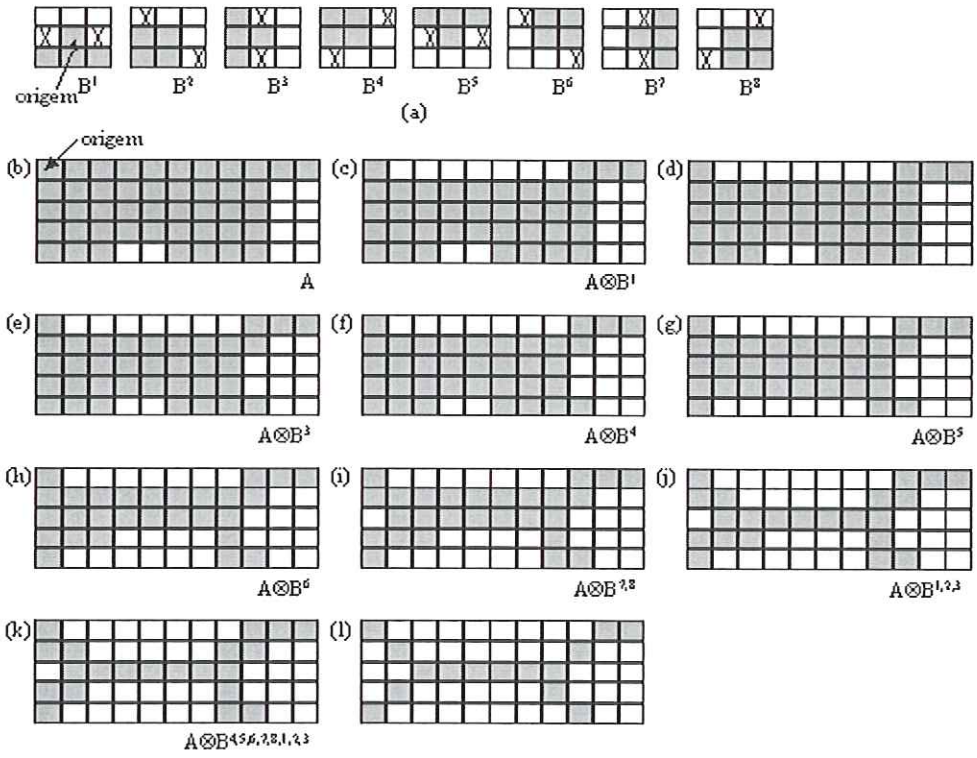


FIGURA 3.17 – Ilustração do algoritmo de afinamento. (a) Sequência de elementos estruturantes usados para o processo de afinamento; (b) conjunto original A; (c) resultado do afinamento com o primeiro elemento; (d)-(i) resultados do afinamento com os próximos sete elementos restantes (não ocorreu mudanças entre o sétimo e oitavo elemento); (j) resultado após a utilização do primeiro elemento pela segunda vez (não ocorreu mudanças para os próximos dois elementos); (k) resultado final; (l) conversão para conectividade m.

3.5.6 Esqueletos

O esqueleto de um conjunto A pode ser obtido pelo seguinte procedimento:

$$S(A) = \bigcup_{k=0}^{K} S_k(A), \text{ onde:}$$

$$S_k(A) = \{(A \Theta kB) - [(A \Theta kB) \circ B]\} \text{ e}$$

$$K = \max\{k \mid (A \Theta kB) \neq \varnothing\} \qquad (3.24)$$

Para se reconstruir a imagem original, deve-se executar o seguinte procedimento:

$$A = \bigcup_{k=0}^{K} (S_k(A) \oplus kB) \qquad (3.25)$$

A formulação dada pela equação (3.24) é conhecida como fórmula de Lantuejoul. Este processo pode ser exemplificado passo a passo com a ajuda da figura 3.18.

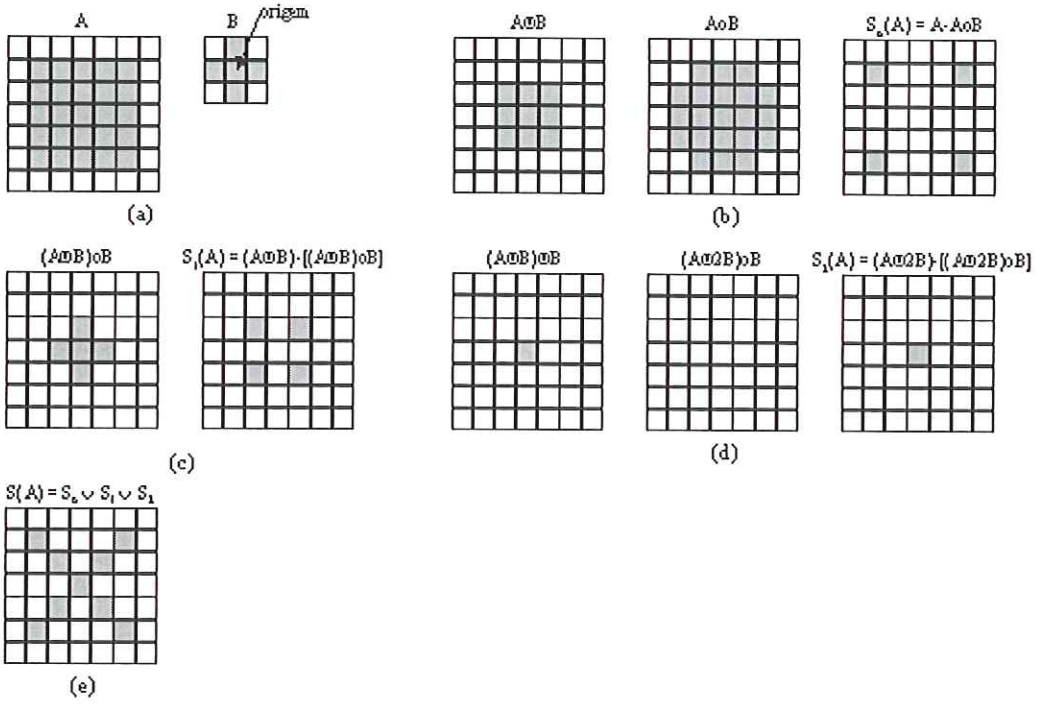


FIGURA 3.18 – Ilustração do algoritmo para determinação de esqueleto morfológico. (a) Conjunto inicial A (quadrado) e elemento estruturante usado para o processo de esqueletização de A; (b) passo k=0 do algoritmo descrito pela equação (3.24); (c) passo k=1; (d) passo k=2 onde o resultado das k erosões sucessivas de A ainda é diferente de vazio; (e) esqueleto resultante de A.

3.5.7 Poda

Um algoritmo de poda para pós-processamento é usado para remover componentes parasitas criados pelos métodos de afinamento e de esqueletização devido à deformidades nos segmentos do conjunto original. Por exemplo, estes componentes poderiam prejudicar o processo de reconhecimento automático de caracteres escritos à mão através da análise de forma do esqueleto de cada caracter. Ao contrário do afinamento, o processo de poda não é idempotente, ou seja, o fato de continuar o processo pode resultar numa redução ou numa grande diminuição ou até numa destruição parcial da imagem afinada. Portanto, o número de ciclos no processo de poda deve ser predeterminado. A equação 3.26 descreve o algoritmo morfológico usado para o processo de poda, onde X_1 denota o conjunto original afinado pela sequência de elementos estruturantes B; X_2 contém os pontos terminais de

X_1; X_3 representa o resultado da dilatação de X_2 por H (elemento estruturante 3x3 de 1's) usando-se o conjunto A como delimitador, previnindo-se assim a criação de pontos de valor 1 fora da região de interesse, entretanto, em cenários mais complexos esta abordagem pode recriar a extremidade de ramificações espúrias; X_4 fornece o resultado final.

$$X_1 = A \otimes \{B\},$$

$$X_2 = \bigcup_{k=1}^{8} (X_1 \text{ hom } B^k),$$

$$X_3 = (X_2 \oplus H) \cap A,$$

$$X_4 = X_1 \cup X_3 \qquad (3.26)$$

Na figura 3.19, tem-se um exemplo de remoção de componentes parasitas de comprimento igual a um pixel num esqueleto de um caracter pelos elementos estruturantes detetores de pontos terminais indicados pela sequência {B}.

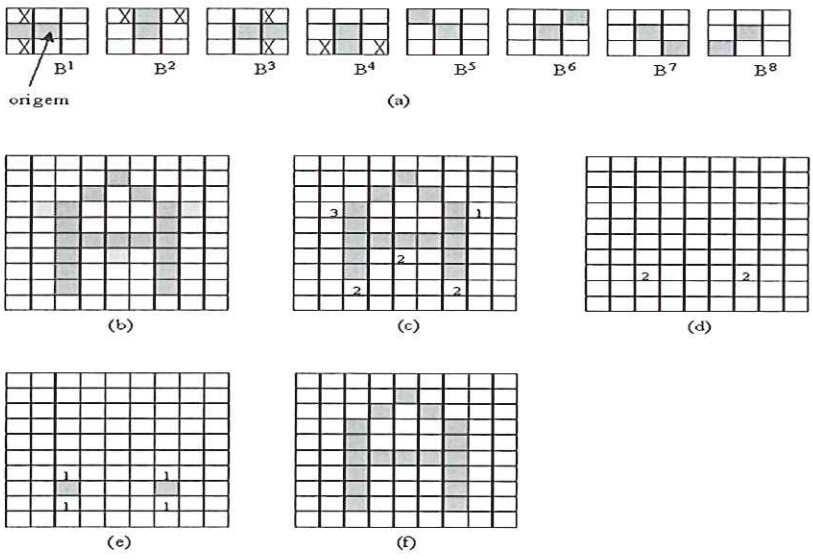


FIGURA 3.19 – Ilustração do algoritmo de poda. (a) Sequência dos oito elementos estruturantes comumente usados para poda; (b) conjunto X_0 formado pelo esqueleto de um caracter A com pontos espúrios representados pelos tons mais claros; (c) conjunto X_1 após uma iteração de A⊗{B}, pois, considerou-se que os ramos parasitas tem comprimento unitário (um pixel); (d) conjunto X_2 representando os pontos terminais de X_1; (e) conjunto X_3 formado por uma dilatação condicionada a X_0 de X_2 por H; (f) imagem podada.

3.6 Extensões Para Imagens em Níveis de Cinza

Serão apresentadas nesta seção as operações de dilatação, erosão, abertura e fechamento estendidas para imagens em níveis de cinza. Ao longo da discussão, f(x,y) corresponderá a uma imagem (função) digital de entrada e b(x,y) corresponderá ao elemento estruturante, ou seja, uma subimagem digital. Assume-se também, que essas funções sejam discretas, isto é, se Z denotar os números inteiros, assume-se que (x,y) pertençam a ZxZ e f e b sejam funções que atribuam um nível de cinza (um número real R) a cada par distinto de coordenadas (x,y). Se os níveis de cinza também forem inteiros, Z deve substituir R.

3.6.1 Dilatação

A dilatação em níveis de cinza de f por b, denotada, f ⊕ b, é definida como:

$$(f \oplus b)(s,t) = \max\{f(s-x,t-y)+b(x,y) \mid (s-x),(t-y) \in D_f;(x,y) \in D_b\} \quad (3.27)$$

Na equação 3.27, D_f e D_b são os domínios de f e b, respectivamente. Como antes, b é o elemento estruturante do processo morfológico. Aqui, diferentemente do caso binário, f é deslocada, no lugar do elemento estruturante b. Também, este processo pode ser realizado por b sendo refletido em relação a sua origem e deslocado pela imagem f. Visto que a dilatação se baseia na escolha do valor máximo de $f+b$ em uma vizinhança definida pela forma do elemento estruturante, o efeito geral da dilatação de uma imagem em níveis de cinza se desdobra em dois:

- se todos os valores do elemento estruturante forem positivos, a imagem resultante tende a ser mais clara que a imagem de entrada.
- detalhes escuros são reduzidos ou eliminados, dependendo de como seus valores e formatos estejam relacionados com o elemento estruturante utilizado.

3.6.2 Erosão

A erosão em níveis de cinza, denotada por f Θ b, é definida como:

$$(f\Theta b)(s,t) = \min\{f(s+x,t+y) - b(x,y) \mid (s+x),(t+y) \in D_f; (x,y) \in D_b\} \quad (3.28)$$

Na equação 3.28, D_f e D_b são os domínios de f e b, respectivamente. Aqui também, f é deslocada, e não o elemento estruturante. Também, este processo pode ser realizado por b sendo deslocado pela imagem f. A erosão se baseia na escolha do valor mínimo de f-b em uma vizinhança definida pela forma do elemento estruturante. Existem dois efeitos gerais da erosão de uma imagem:

- se todos os elementos do elemento estruturante forem positivos, a imagem de saída tende a ser mais escura que a imagem de entrada.
- o efeito de detalhes claros na imagem de entrada que forem menor em área que o elemento estruturante é reduzido, sendo que o grau dessa redução é determinado pelos valores dos níveis de cinza em torno do detalhe claro e pela forma e valores de amplitude do próprio elemento estruturante.

3.6.3 Abertura e Fechamento

As expressões de abertura e de fechamento de imagens em níveis de cinza possuem a mesma forma que as análogas no caso binário. A abertura de uma imagem f por uma imagem b (elemento estruturante), denotada por f∘ b, é dada por:

$$f \circ b = (f\Theta b) \oplus b \quad (3.29)$$

Como no caso binário, a abertura é simplesmente a erosão de f por b seguida da dilatação do resultado por b. De maneira similar, o fechamento de f por b, denotado por f• b, é dado por:

$$f \bullet \mathrm{b} = (f \oplus b)\Theta b \qquad (3.30)$$

Na prática, as operações de abertura são usualmente aplicadas na remoção de pequenos (em relação ao tamanho do elemento estruturante) detalhes claros, enquanto não altera os níveis de cinza em geral nem os grandes elementos claros. A erosão inicial não só remove os pequenos detalhes como também escurece a imagem. A dilatação subseqüente novamente aumenta a claridade da imagem sem reintroduzir os detalhes removidos pela erosão. O fechamento é geralmente usado na remoção de detalhes escuros em uma imagem, enquanto deixa os elementos claros relativamente inalterados. A dilatação inicial remove os detalhes escuros e clareia a imagem, enquanto a erosão subseqüente escurece a imagem sem reintroduzir os detalhes removidos pela dilatação.

3.7 Considerações Finais

Em processamento de imagens, a morfologia matemática que representa um ramo de processamento não linear permite processar imagens com objetivos de realce, segmentação, detecção de bordas, esqueletização, afinamento, análise de formas, compressão, entre outros. A morfologia matemática fornece uma aproximação ao processamento de imagens digitais que é baseada na forma. Apropriadamente usadas, as operações de morfologia matemática tendem a simplificar os dados da imagem preservando suas características de forma essenciais e eliminando irrelevâncias. De acordo com (GONZÁLEZ et al., 2002), a morfologia matemática pode resolver algumas características indesejáveis do processamento de imagens linear clássico, uma vez que os filtros lineares geram uma distorção espacial na imagem original. O formato e o tamanho do elemento estruturante possibilitam testar e quantificar de que maneira o elemento estruturante está ou não está contido na imagem. Assim, a saída da operação morfológica depende essencialmente da forma do elemento estruturante (GASTERATOS, 2002). Também, a decomposição do elemento estruturante reduz de forma linear o número de operações envolvidas nos processos morfológicos. Outra grande vantagem da morfologia matemática é sua simplicidade de implementação, onde, através de suas operações básicas, dilatação e erosão, por composição é possível realizar muitos outros operadores poderosos,

diferentemente das demais técnicas de processamento de imagens que na maioria dos casos não se aproveitam das ferramentas já existentes. Para propósitos de identificação de objetos ou peças defeituosas, requeridos em aplicações de visão industriais, as operações de morfologia matemática são mais úteis que as de convolução empregadas em processamento de sinais, pois os operadores morfológicos se relacionam diretamente com a forma. Outras aproximações para a morfologia matemática são a morfologia matemática *soft* (KOSKINEN et al., 1991) e a morfologia matemática *fuzzy* (BLOCH e MAITRE, 1995).

Através da esqueletização morfológica é possível reduzir a complexidade do processo de identificação de objetos, pois o número de pixels requeridos para representar o objeto é reduzido. Em termos de compressão, o processo de esqueletização produz estruturas mais compactadas que as obtidas pelo processo de afinamento. Todavia, a estrutura produzida pela esqueletização é quase sempre descontínua e geralmente não reflete a geometria do objeto. Para o processo de afinamento, tem-se exatamente o inverso, pois este conserva a homotopia (conservação do número de conectividade).

3.7.1 Desvantagens

A qualidade das operações morfológicas dependem de seleções apropriadas de elementos estruturantes (IMAGING RESEARCH INC, 1998). Muitas vezes, a escolha de um elemento estruturante adequado ou a escolha de uma classe de elementos estruturantes adequados para uma determinada aplicação não é uma tarefa trivial na prática. Também, os processos morfológicos são todos computacionalmente intensivos. Para se obter um elevado desempenho computacional nos processamentos morfológicos, estes devem ser implementados em hardware.

4 SISTEMA DE AQUISIÇÃO, ARMAZENAMENTO E EXIBIÇÃO DE IMAGENS MONOCROMÁTICAS

4.1 Considerações Iniciais

Neste capítulo será descrito o funcionamento de um sistema que utiliza um CPLD para aquisição e armazenamento de imagens em tons de cinza. O sistema foi desenvolvido como exercício para dominar o projeto lógico com CPLDs e para se familiarizar com o sinal de vídeo composto e o processo de aquisição e armazenamento deste sinal. Também, este servirá de base à arquitetura desenvolvida no capítulo 6. As imagens digitais são obtidas através da amostragem de um sinal de vídeo fornecido por uma câmera CCD (*Charge-coupled device*). Antes de se iniciar a descrição do projeto é importante compreender o formato do sinal de vídeo composto a ser utilizado. Na subseção seguinte será feita uma abordagem mais aprofundada sobre este sinal.

4.1.1 Sinal de Vídeo Composto

Ao sinal de vídeo acrescido dos pulsos de apagamento e de sincronismo dá-se o nome de *sinal de vídeo composto* (SENATORI e SUKYS, 1987). O sinal de vídeo adotado neste projeto obedece às especificações do padrão convencional NTSC (*National Television Systems Committee*), definido em 1952, ainda em uso principalmente na América do Norte e Japão (TEKALP, 1995). O sinal NTSC possui varredura entrelaçada (2:1), 262,5 linhas por campo (525 linhas por quadro) e frequência de varredura vertical (f_v) de 60 campos por segundo (30 quadros por segundo). Assim, a frequência horizontal de varredura (f_h) é: 525x30=15,75 kHz, logo, para se varrer cada linha horizontal são

necessários 63,5 μs (1H). A figura 4.1 ilustra o processo de varredura entrelaçada onde as linhas sólidas pertencem ao campo ímpar e as linhas tracejadas ao campo par.

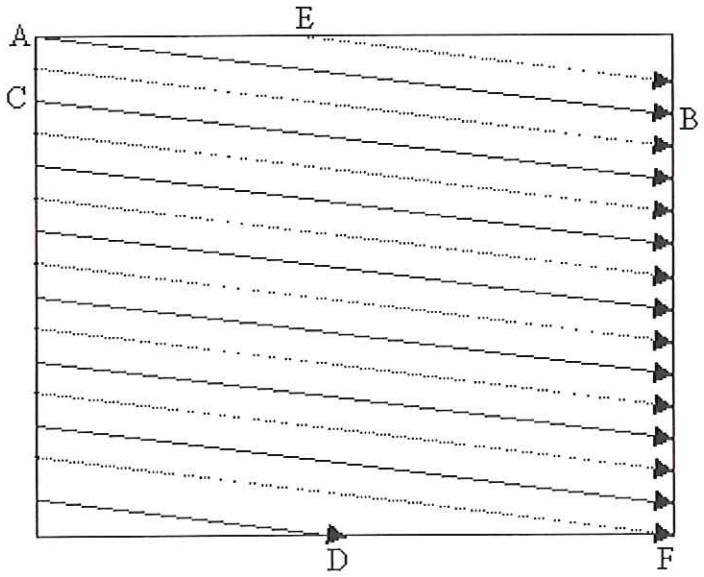


FIGURA 4.1 – Processo de varredura entrelaçada. De A a B, tem-se um período conhecido como traço horizontal, enquanto de B a C o período é chamado de retraço horizontal. De forma análoga, de D a E e de F a A, tem-se um período de retraço vertical. O traço vertical corresponde à ação do feixe de se deslocar de cima para baixo. O intercalamento é possível graças à posição especial do ponto inicial de varredura em cada campo.

Baseando-se na figura 4.1, para fazer o feixe eletrônico se deslocar da esquerda para a direita uniformemente, deve-se aplicar à bobina defletora horizontal do cinescópio uma tensão dente-de-serra adequada. Concomitantemente, o mesmo processo deve ocorrer à bobina defletora vertical para que o feixe possa se deslocar de cima para baixo. À medida que o feixe se movimenta da esquerda para a direita, sua intensidade varia de acordo com o sinal de vídeo. Quando o feixe atinge o lado direito da tela, perfazendo uma linha, ele deve voltar ao lado esquerdo para recomeçar uma nova linha. Durante esse intervalo de tempo, o feixe não deve ser visível, portanto, é aplicado à grade da válvula de imagem um pulso de apagamento horizontal que permanece num nível mais preto que o preto. Depois que o feixe, sob a ação da varredura vertical, completa um campo, ele deve ser novamente apagado para retornar à parte superior da tela. Novamente, a grade da válvula de imagem é polarizada negativamente, agora durante um intervalo de tempo superior ao que se verifica no caso de apagamento horizontal. Assim, segundo o padrão NTSC, durante o primeiro

campo, o feixe de elétrons cobre 247,5 linhas ativas e o retorno vertical desse campo cobre 15 linhas inativas que permanecerão apagadas no monitor. O mesmo ocorre ao segundo campo, portanto, cada um deles perfaz 247,5 + 15 = 262,5 linhas. Para que este tipo de varredura possa ser realizável, é necessário que a frequência de varredura horizontal seja 262,5 vezes maior que a vertical. Deste modo, assegura-se que a proporcionalidade entre as frequências seja um número fracionário de maneira a garantir meia linha no final e início de campos consecutivos. Também, o sinal dente-de-serra vertical deve ser o mesmo para os dois campos, o que é facilmente obtido por meio de um único gerador de varredura. Os sinais dente-de-serra para as varreduras horizontal e vertical são obtidos de osciladores distintos, e para que se estabeleça um perfeito sincronismo na tela, os dois osciladores são sincronizados por sinais específicos emitidos pela câmera.

O sinal de vídeo puro leva apenas informações referentes à luz e sombra de uma imagem. Todavia, essas informações são insuficientes, uma vez que há a necessidade de se comandar os osciladores de sincronismo no monitor. Suponha-se, por exemplo, que uma imagem qualquer esteja sendo explorada na câmera conforme a figura 4.1. Quando o feixe descreve a linha A-B, recolhem-se as informações de imagem, que possui forma arbitrária, mas cuja amplitude se situa entre o nível de preto e o nível de branco. Ao atingir o ponto B o feixe deve retornar ao lado esquerdo e recomeçar a varredura da linha adjacente. Entretanto, no final da linha a intensidade luminosa do ponto explorado pode ser qualquer, inclusive a mais clara. Assim, nesse instante deve-se aplicar à grade da válvula de imagem o impulso de extinção horizontal levando a mesma ao nível de preto e, portanto, apagando o feixe. Para que o feixe seja comandado de volta e possa recomeçar no ponto C é necessário acrescentar aos sinais de vídeo e extinção o impulso de sincronismo horizontal. Esse impulso ocorre na região abaixo do nível de preto conhecida como mais preto que o preto. A figura 4.2 mostra o formato de uma linha de vídeo adicionada dos pulsos de extinção e de sincronismo horizontal. O nível de preto corresponde a uma tensão de 0 volts e o nível de branco tem amplitude de 0,7 volts. Os níveis de cinza ficam entre estes dois extremos. O nível de sincronismo possui amplitude de –0,3 volts. Na prática é muito difícil obter um sinal de extinção vertical, ou seja, que num tempo infinitamente curto levasse a grade de controle do tubo à tensão correspondente ao nível de preto. Para se contornar esse

inconveniente, o sinal de vídeo deve permanecer durante certo intervalo de tempo no nível de preto e em seguida deve-se aplicar o impulso de sincronismo. Com isso, consegue-se fazer com que os impulsos sejam sempre aplicados no mesmo instante. Consequentemente, o sinal de vídeo apresenta um ligeiro degrau que é conhecido como pórtico ou bordo. O pórtico de início do impulso de sincronismo é conhecido como pórtico anterior e o pórtico do final do impulso denomina-se pórtico posterior. A duração do pórtico anterior é de 0,02H e a do pórtico posterior é de 0,06H. A largura do pulso de sincronismo corresponde a 0,08H.

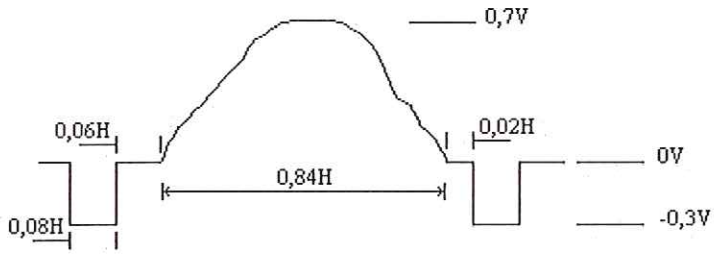


FIGURA 4.2 – Formato de uma linha de sinal de vídeo composto. O sinal de vídeo composto possui amplitude de $1V_{pp}$, onde o nível de branco corresponde a 0,7V, o nível de preto a 0V e o nível de sincronismo a –0,3V, ficando os níveis de cinza entre os níves de preto e de branco. O período de extinção horizontal é de 0,16H, enquanto o período de sincronismo horizontal é de 0,08H. O pórtico anterior possui período de 0,02H e o pórtico posterior de 0,06H. Considerando-se que o período de uma linha do sinal é de 1H (63,5 μs), o período de informações de vídeo será de 0,84H (53,34 μs).

Ao terminar a exploração da última linha da parte inferior da imagem, o feixe deve ser apagado e reconduzido à parte superior da tela. Nesse instante, são aplicados pulsos de extinção e de sincronismo vertical. Os impulsos de sincronismo vertical são precedidos e sucedidos por 6 impulsos de duração menor, denominados impulsos equalizadores. Os impulsos equalizadores tem dupla função, sendo uma delas a de manter o sincronismo horizontal enquanto o feixe está sendo reconduzido para cima, pois, se nesse intervalo de tempo em que não há exploração de linha não forem aplicados impulsos de sincronismo ao oscilador horizontal, ele perderá o sincronismo. A outra função relevante dos impulsos equalizadores é a de manter o intercalamento perfeito entre dois campos sucessivos (KUHN, 1996; NINCE, 1991; SCHURE, 1964 e SENATORI e SUKYS, 1987). A duração

de um impulso equalizador é de 2,54 μs e o intervalo entre dois impulsos é de 0,5 H. O formato final do sinal de vídeo composto padrão está ilustrado na figura 4.3.

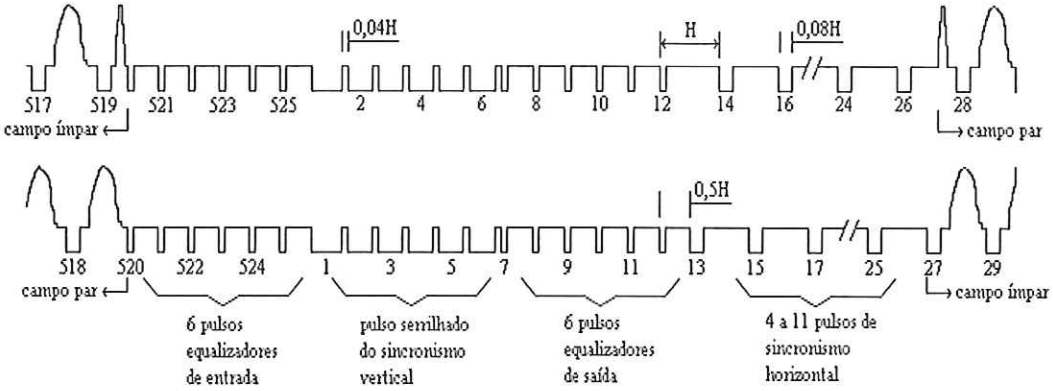


FIGURA 4.3 – Formato do sinal padrão de vídeo composto.

De acordo com a figura 4.3, o intervalo entre o início do apagamento vertical e o início do pulso serrilhado deve conter 6 pulsos, correspondentes aos números de 520 a 525, a fim de se sincronizar o oscilador horizontal do monitor. No final do 1° campo o sincronismo horizontal é feito com os pulsos 521, 523 e 525, e no fim do 2° campo com os pulsos 520, 522 e 524. Esses pulsos ficam afastados de ½ H e sua largura dever ser a metade daquela exibida por um pulso de sincronismo horizontal, a fim de se manter a mesma tensão residual na rede integradora do sistema de sincronismo vertical. Por esse motivo, estes pulsos recebem o nome de pulsos equalizadores, pois realmente equalizam as cargas residuais da rede integradora para ambos os campos. O oscilador horizontal não sofre influência de dois pulsos equalizadores sucessivos porque não há a possibilidade do mesmo ser sincronizado por um pulso que ocorra muito antes (meio período) do momento adequado para o retorno. O pulso serrilhado de sincronismo vertical deverá conter 6 descontinuidades correspondentes aos números de 1 a 6, com frequência igual à dos pulsos equalizadores e largura suficientemente pequena, a fim de se reduzir o tempo de subida dos pulsos integrados na saída da rede, minimizando-se assim o efeito de denteamento do sinal. No intervalo de 3H imediatamente após o fim do pulso serrilhado, devem existir mais 6 pulsos equalizadores (de 7 a 12) idênticos aos anteriores. O gerador que fornece os pulsos

equalizadores de entrada produz também os de saída, além de produzir o serrilhado no pulso vertical. Logo após os pulsos equalizadores de saída, é reiniciado o envio de pulsos de sincronismo horizontal, cuja quantidade pode variar de 5 a 9 antes do término do apagamento vertical. Haverá, desse modo, uma distância de H entre o pulso 14 e o último equalizador de saída para o 2° campo e de 0,5H entre o pulso 13 e o último equalizador de saída para o 1° campo.

4.2 Circuito de Aquisição, Armazenamento e Exibição de Imagens Monocromáticas

Nesta seção será descrito sucintamente o funcionamento do circuito de aquisição, armazenamento e exibição de imagens monocromáticas desenvolvido. O diagrama em blocos do sistema supracitado está ilustrado na figura 4.4. No Apêndice A, tem-se o circuito esquemático completo do referido projeto.

4.2.1 Separador de Sincronismo

O separador de sincronismo utilizado neste projeto foi o EL4581C da Elantec, cuja função é extrair as informações de sincronismo vertical, horizontal, pórtico posterior e de tipo de campo (par ou ímpar) de um sinal de vídeo composto padrão. Estes sinais são relevantes porque é através deles que a unidade de controle identificará o início de uma nova linha de vídeo ou de um novo campo, gerando-se assim os sinais de controle e os endereços para as memórias. Dessa forma, as operações de leitura ou escrita serão efetuadas nos tempos adequados. A figura 4.5 ilustra o formato dos sinais de saída deste chip dado um sinal de vídeo composto padrão de entrada (ELANTEC, 1993).

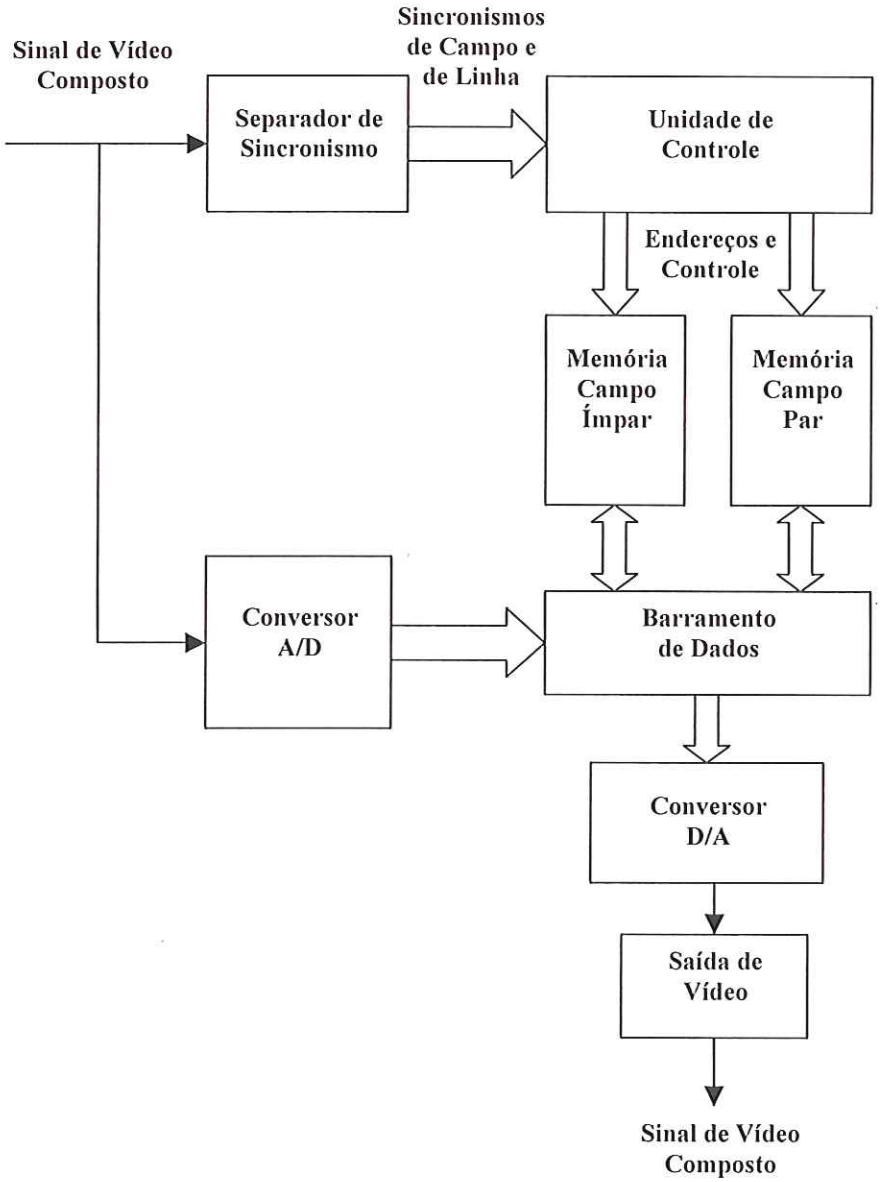


FIGURA 4.4 – Diagrama em blocos do circuito de aquisição, armazenamento e exibição de imagens monocromáticas.

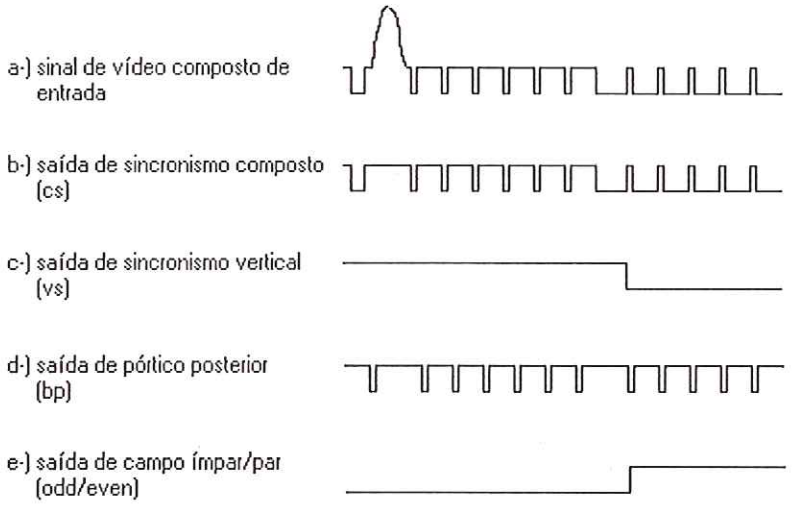


FIGURA 4.5 – Ilustração dos sinais extraídos do separador de sincronismo dado um sinal de vídeo composto de entrada. O sinal de entrada está representado em a-). Em b-), tem-se o sinal de sincronismo composto (cs) que reproduz todos os pulsos de sincronismo do sinal de entrada. Em c-), tem-se o sinal de sincronismo vertical (vs) cuja borda de descida coincide com a borda de subida do primeiro pulso serrilhado. Em d-), tem-se o sinal de pórtico posterior (bp) e em e-), tem-se o sinal de saída (odd/even) que será baixo quando o campo for par e alto quando o campo for ímpar.

4.2.2 Conversor A/D

Uma vez que a informação de uma linha de vídeo é de aproximadamente 53 μs, para se obter 512 amostras nesse tempo é necessário utilizar um conversor A/D capaz de realizar a conversão em aproximadamente 100 ns, o que equivale a 10 milhões de amostras por segundo. Assim, é preciso utilizar conversores analógico/digitais do tipo flash, que são os mais indicados para a digitalização de sinais de vídeo por permitirem altas taxas de conversão. Dessa forma, utilizou-se para este projeto o conversor A/D Bt218KP20 de 8 bits do tipo flash da BrookTree, projetado especificamente para aplicações de digitalização de sinais de vídeo que requerem taxas de conversão de até 20MSPS (CONEXANT, 1999). Na figura 4.6, tem-se o diagrama de tempos do conversor A/D Bt218. As informações digitalizadas são armazenadas nas memórias de acordo com os sinais de controle e endereços gerados pela unidade de controle.

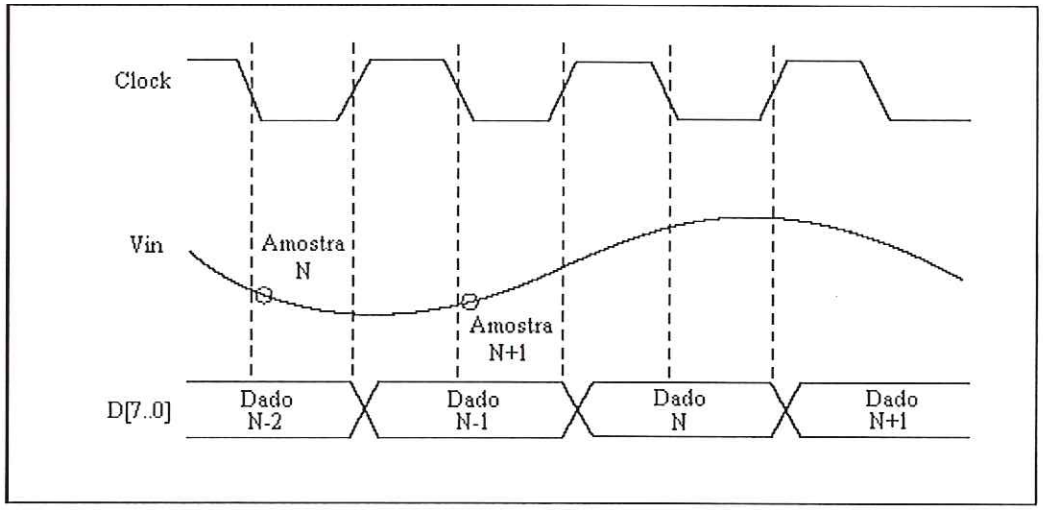
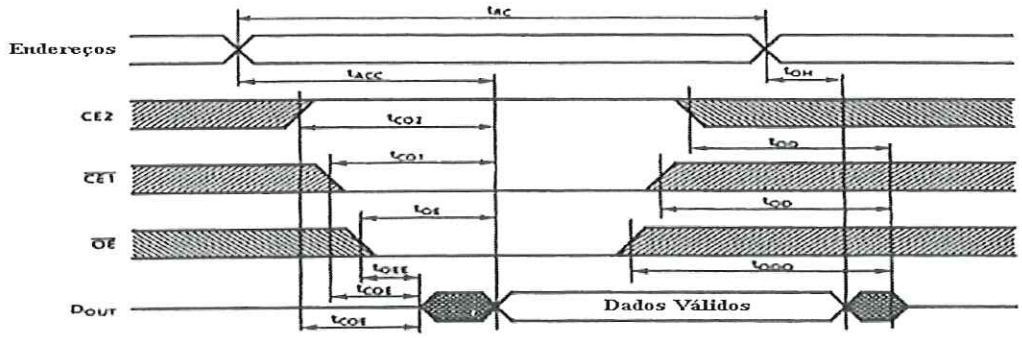


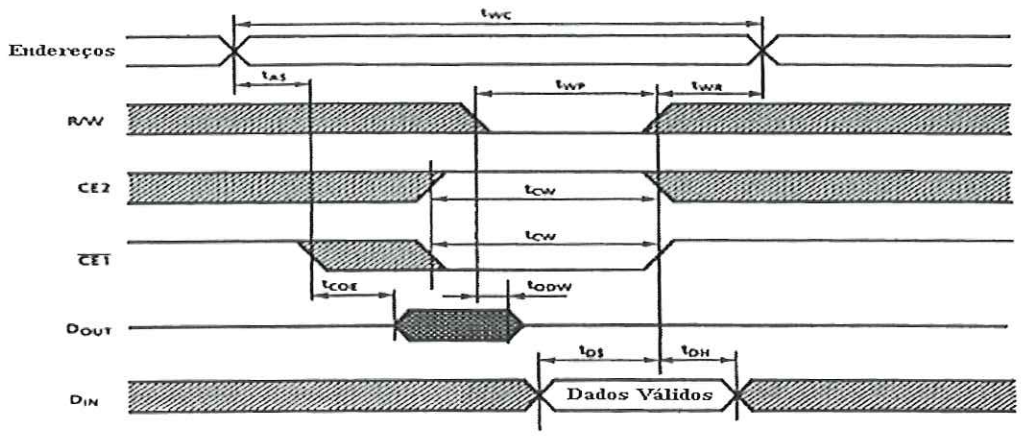
FIGURA 4.6 – Diagrama de tempos do conversor A/D Bt218. O sinal é amostrado na borda de descida do clock e os dados ficam disponíveis na segunda borda de subida.

4.2.3 Memórias

Considerando-se que o sinal de vídeo é entrelaçado, utilizou-se duas memórias RAMs estáticas para armazenar cada um dos campos. A memória RAM adotada foi a TC551001 de 70ns e 131072 palavras de 8 bits da Toshiba. Os diagramas de tempos dessa memória para os ciclos de leitura e de escrita estão ilustrados na figura 4.7 (TOSHIBA, 1997). A unidade de controle gera os sinais de controle e endereços à memória segundo os diagramas de tempo da figura 4.7, controlando assim as operações de escrita e leitura.



a-) Ciclo de leitura.



b-) Ciclo de escrita.

FIGURA 4.7 – Diagramas de tempo da memória RAM TC551001. a-) ciclo de leitura. b-) ciclo de escrita.

4.2.4 Unidade de Controle

Para se implementar a unidade de controle foi utilizado o CPLD EPM7128SLC84-7 da família MAX 7000S da Altera. Para se programar o CPLD, utilizou-se o software MAX+PLUS II, também da Altera. Na figura 4.8, tem-se o diagrama esquemático da unidade de controle. Nesta subseção será analisado o funcionamento de todos os blocos constituintes desta unidade.

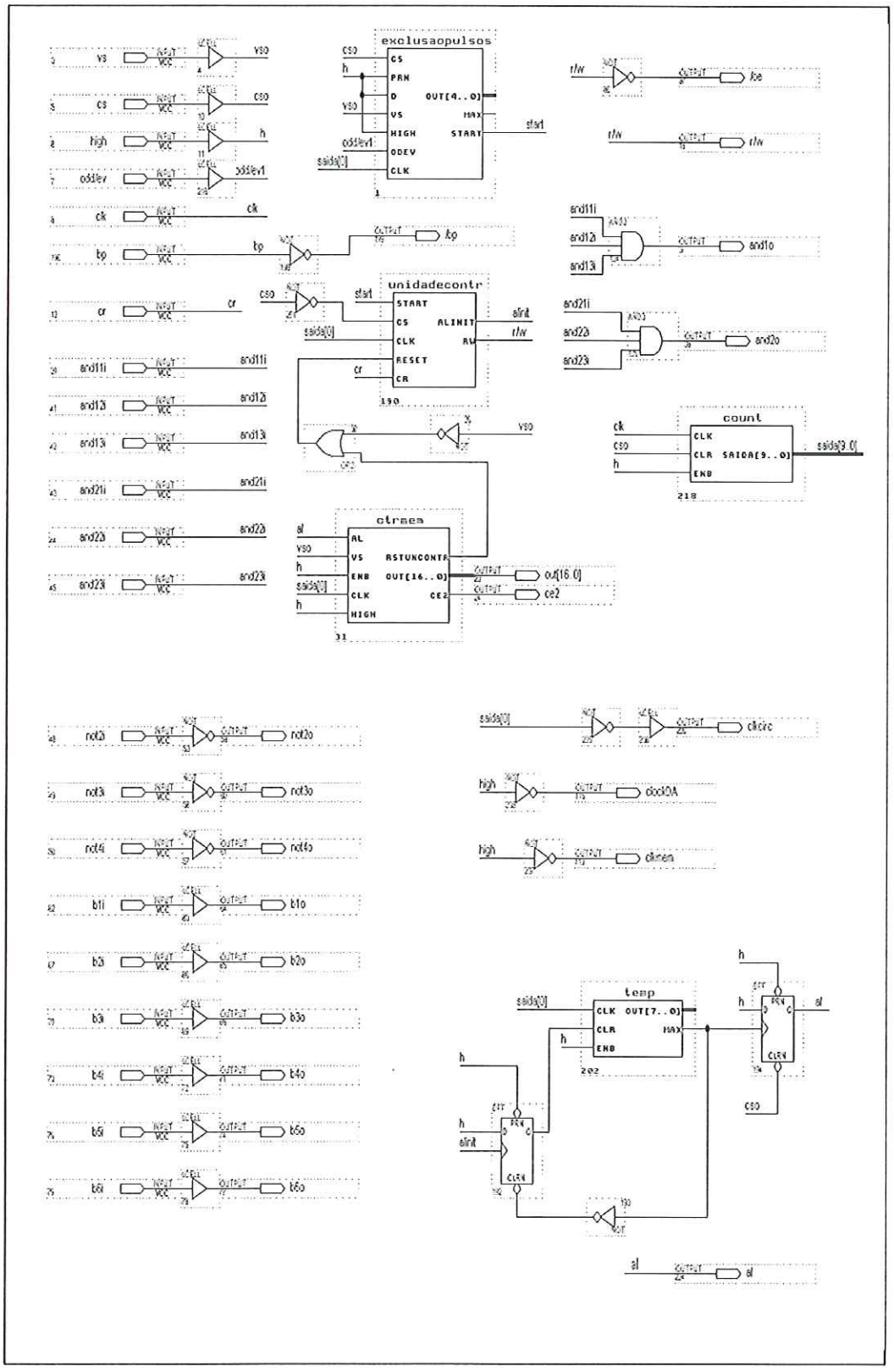


FIGURA 4.8 – Diagrama esquemático da unidade de controle.

4.2.4.1 Unidade de Exclusão de Pulsos de Sincronismo

O objetivo desta unidade é excluir os pulsos equalizadores e serrilhados fornecidos pelo sinal *cs*. Esta unidade está representada pelo símbolo *exclusaopulsos* na figura 4.8 e é composta por um contador, desenvolvido em HDL, responsável pela contagem dos pulsos de sincronismo após o início de cada campo e por uma máquina de estados, também desenvolvida em HDL, cuja função é sinalizar à unidade de controle interna *unidadecontr* que os pulsos de sincronismo foram excluídos através de um sinal interno denominado *start*. Na figura 4.9, tem-se o diagrama esquemático da unidade de exclusão de pulsos de sincronismo.

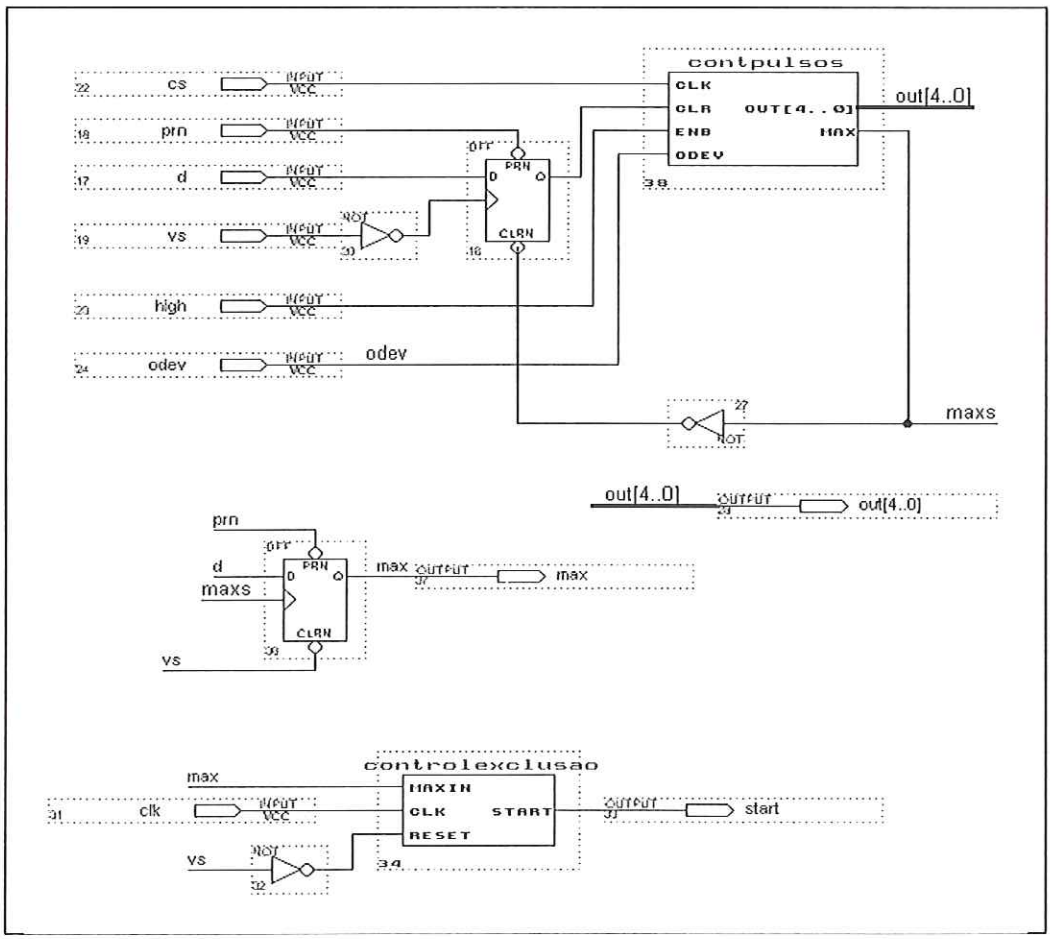


FIGURA 4.9 – Unidade de exclusão de pulsos de sincronismo.

Na borda de descida do pulso de sincronismo vertical, *vs*, a máquina de estados *controlexclusao* será resetada e seu estado inicial indicará o início da exclusão dos pulsos de sincronismo, estado *ie*. Enquanto o sinal *vs* estiver em nível lógico baixo, o sinal *start* também permanecerá em nível lógico baixo. Concomitantemente, após a descida do pulso de sincronismo vertical, o contador *contpulsos* será habilitado pelo *flip-flop* tipo D. Se o campo for par serão excluídos 15 pulsos, senão, serão excluídos 16. Após a subida do sinal *vs*, a máquina de estados mudará para o estado de exclusão de pulsos, estado *ex*. Neste mesmo instante, o sinal *start* passará para nível lógico alto, permanecendo neste nível até o contador *contpulsos* sinalizar à máquina de estados através do sinal *max*, indicando que a contagem atingiu seu valor máximo, fazendo o mesmo voltar para nível lógico zero e resetando o contador. No final da exclusão dos pulsos de sincronismo, a máquina de estados passará para o estado de fim de exclusão de pulsos, *fe*. No Apêndice B, tem-se o código HDL do contador de pulsos e no Apêndice C, tem-se o código HDL da máquina de estados *controlexclusao*.

4.2.4.2 Unidade de Controle Interna

A função desta unidade é gerar um sinal de disparo a um circuito de controle responsável por enviar às memórias um sinal denominado *al* que habilitará as mesmas para as operações de escrita ou leitura durante os períodos de linha ativa do sinal de vídeo. Este sinal é produzido por um circuito de controle formado por um contador de pulsos e dois *flip-flops* tipo D, sendo um deles disparado por um sinal denominado *alinit* gerado pela máquina de estados *unidadecontr*, implementada em HDL, que é resetada durante o pulso de sincronismo vertical e controlada pelos sinais *start* e *cs*. O estado inicial desta máquina de estados é denominado *vbi*, e corresponde ao período de apagamento vertical e início de um novo campo. Durante este estado o sinal *al* possuirá nível lógico baixo. Ainda, durante o período de apagamento vertical, quando o sinal *start* mudar para nível lógico alto, a máquina de estados mudará de seu estado atual, *vbi*, para um novo estado denominado *vb*, e o sinal *al* ainda permanecerá em nível lógico baixo. Quando o sinal *start* mudar novamente durante o estado *vb* para nível lógico baixo, a máquina de estados *unidadecontr* passará para um novo estado denominado *li* que corresponderá ao início de

uma linha ativa de vídeo e o sinal *al* continuará em nível lógico baixo. Somente quando o sinal *cs* passar para nível lógico baixo, o sinal de saída *alinit* da máquina de estados passará para nível lógico alto, habilitando através de um *flip-flop* tipo D um contador de pulsos denominado *temp* que ficará contando durante um período de 5 μs e excluindo o pulso *bp*. No final deste período o contador será resetado e produzirá um sinal de nível lógico alto que habilitará outro *flip-flop* tipo D que fará com que o sinal *al* passe para nível lógico alto, mantendo este estado até o próximo pulso de *cs* que resetará o *flip-flop*. Daí o ciclo se repetirá, pois a máquina de estados permanecerá no estado *li* até a chegada de um novo pulso de sincronismo vertical. O código HDL da máquina de estados *unidadecontr* pode ser visto no Apêndice D.

4.2.4.3 Unidade de Controle das Memórias

Esta unidade é responsável pela geração de endereços para as memórias e de um sinal de controle apropriado para desabilitar as mesmas no final de cada linha. Na figura 4.10, tem-se o circuito esquemático da unidade de controle *ctrmem* das memórias. Esta unidade constitui-se basicamente de dois *flip-flops* tipo D, um contador de linhas de vídeo denominado *contlin* e um contador de pixels denominado *contpix*. A concatenação das saídas *out* de ambos os contadores produzirá os endereços para as memórias. O contador *contlin* contará até 256 linhas e em seguida fornecerá um pulso de controle a um dos *flip-flops* resetando a unidade de controle unidadecontr, pois neste período as memórias deverão permanecer desabilitadas até o início de um novo campo, que ocorrerá logo após a borda de descida do sinal *vs*. De forma análoga, o contador de pixels, após efeturar a contagem de 512 pixels, enviará um sinal de controle *ep* ao outro flip-flop desabilitando as memórias até o início de uma nova linha ativa de vídeo. Os códigos escritos em HDL dos contadores de linhas de vídeo e de pixels podem ser encontrados respectivamente nos Apêndices E e F.

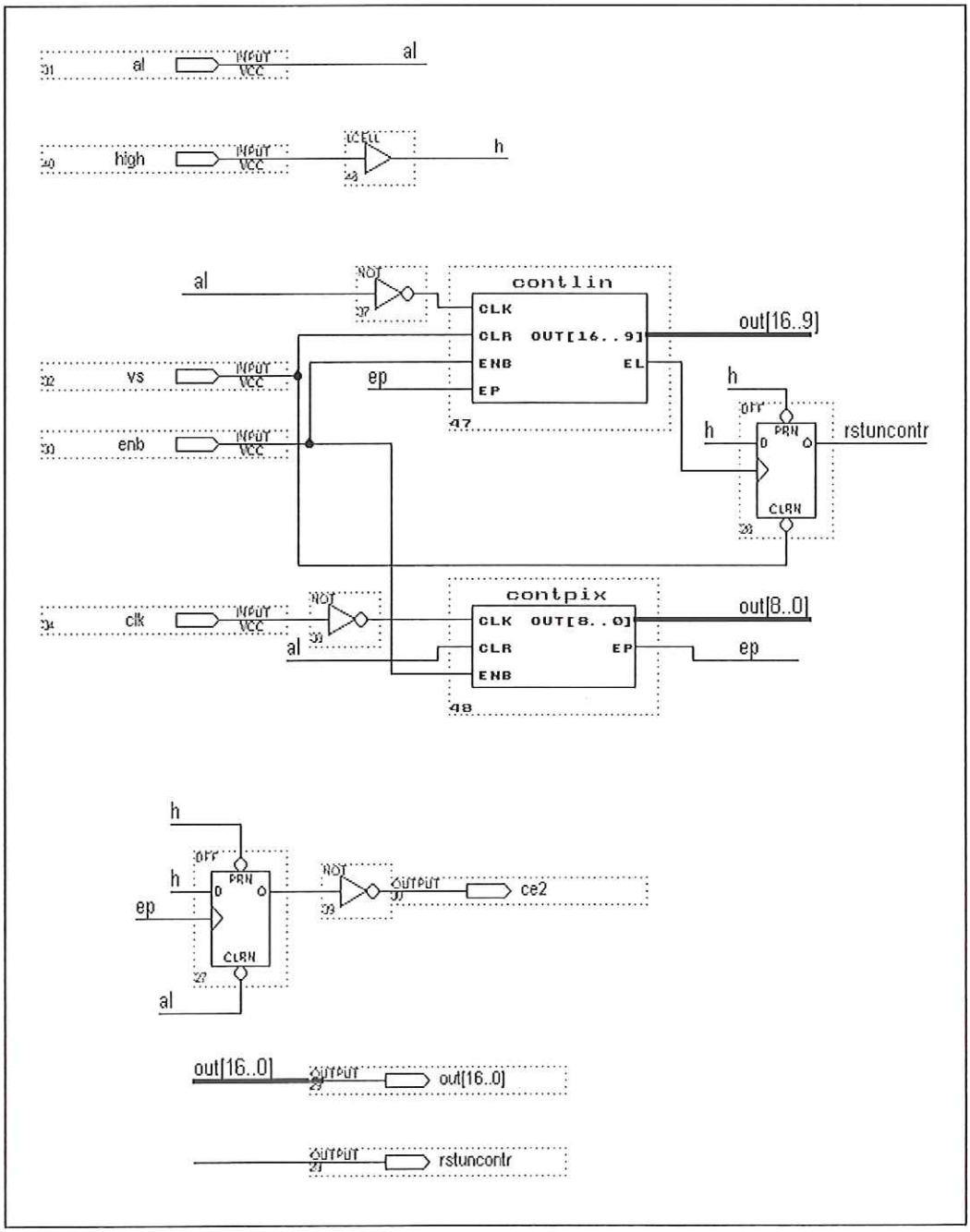


FIGURA 4.10 – Diagrama esquemático da unidade de controle das Memórias.

No Apêndice G, apresenta-se uma simulação da unidade de controle implementada através do software MAX+PLUS II para um sistema hipotético de 4 linhas de vídeo de 512 pixels.

4.2.5 Conversor D/A

O conversor D/A utilizado neste projeto foi o CA3338 de 8bits da Intersil, que utiliza uma rede R2R e é específico para aplicações que envolvem sinais de vídeo. Este conversor pode operar numa taxa de até 50 MSPS. O diagrama de tempos do conversor CA3338 pode ser visto na figura 4.11. Os dados de entrada serão transferidos para a saída quando o pino LE (*Latch Enable*) estiver em nível lógico baixo.

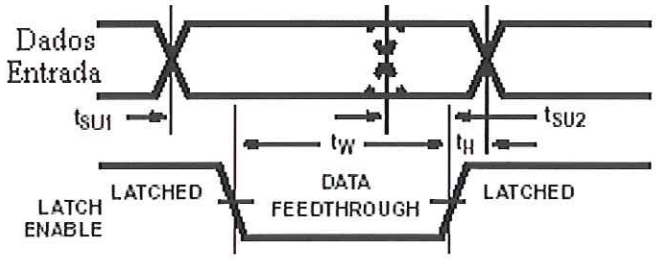


FIGURA 4.11 – Diagrama de tempos do conversor D/A CA3338.

4.2.6 Saída de Vídeo

O objetivo desta unidade será o de reconstituir o sinal de vídeo composto na saída, dados o sinal de saída do conversor D/A e o sinal de sincronismo composto proveniente do separador de sincronismo. Este circuito pode ser visto no diagrama esquemático do Apêndice A.

4.3 Considerações Finais

Todos os níveis de tensão utilizados neste projeto são compatíveis com o nível TTL. As informações técnicas dos diversos dispositivos utilizados neste projeto podem ser encontradas nos *datasheets* específicos de cada fabricante sobredito. As impedâncias de entrada e de saída do sinal de vídeo são de 75 ohms. Também, não foi abordado sobre as informações de cor do sinal de vídeo composto porque foi utilizado um sinal de vídeo fornecido através de uma câmera monocromática compatível com o formato RS170. O clock de 10 MHz do circuito foi obtido pela divisão por 2 da frequência fornecida por um módulo oscilador de cristal de 20 MHz. Através do bit menos significativo de um circuito contador, foi possível realizar esta tarefa. Para eliminar possíveis distorções na imagem na direção vertical devido à falta de sincronismo entre o clock e o início de cada linha de vídeo, tornou-se necessário resetar o circuito contador a cada início de linha de vídeo. No Apêndice H, apresentam-se duas fotos do sistema desenvolvido.

5 ARQUITETURAS PARA MORFOLOGIA MATEMÁTICA

5.1 Considerações Iniciais

Neste capítulo será realizada uma revisão sobre arquiteturas para a implementação das operações básicas de morfologia matemática em hardware, destacando pormenorizadamente as arquiteturas mais relacionadas com o trabalho desenvolvido.

5.2 Arquiteturas Existentes Para Morfologia Matemática

5.2.1 Arquiteturas de Propósito Geral para Processamento de Imagens

Segundo Gasteratos (2002 apud FLYNN, 1966) e (TANENBAUM, 1990), as arquiteturas de computadores são classificadas em quatro categorias principais:

- SISD (*Single Instruction, Single Data*): fluxo único de instruções e de dados. Esta é a máquina tradicional de Von Neumann. Ela tem apenas um fluxo de instruções, isto é, um programa, executado por uma única CPU e uma memória contendo seus dados.
- MISD (*Multiple Instruction, Single Data*): Esta envolve muitos processadores, que executam diferentes instruções sobre o mesmo dado. Esta categoria é considerada hipotética.
- SIMD (*Single Instruction, Multiple Data*): fluxo único de instruções e múltiplo de dados. Aqui, muitos processadores executam a mesma instrução para múltiplos conjuntos de dados em paralelo.

- MIMD (*Multiple Instruction, Multiple Data*): fluxo múltiplo de instruções e de dados. Aqui, CPUs diferentes executam programas diferentes, às vezes compartilhando alguma memória em comum. Para se reduzir os conflitos e aumentar o desempenho computacional, projetistas de sistemas multiprocessados acrescentaram a cada processador sua própria memória local.

Em processamento de imagens, devido a grande quantidade de dados envolvida, o paralelismo se faz necessário. As arquiteturas paralelas podem ser agrupadas em duas categorias principais: síncronas e MIMD. Entretanto, este agrupamento não é único e seus domínios podem ser confundidos. Assim, para a categoria síncrona, tem-se as seguintes arquiteturas: processadores vetoriais, SIMD e arquiteturas sistólicas. Para a categoria MIMD, tem-se: arquiteturas com memória distribuída e com memória compartilhada. Processadores vetoriais utilizam múltiplos elementos de processamento acoplados em múltiplos barramentos. Com eles é possível realizar operações lógicas ou aritméticas, por exemplo, a adição de dois vetores de entrada produzindo um vetor de saída como resultado. Em arquiteturas SIMD, uma unidade de controle difunde as instruções, que são executadas paralelamente por todos os processadores, cada qual usando seu próprio dado de sua própria memória (carregada durante a fase de inicialização). Estas arquiteturas são adequadas para processamento rápido de imagens em baixo nível, onde se enquadra a morfologia matemática. Como exemplo de arquitetura SIMD, pode-se citar o sistema AIS-5000, consistindo-se de um arranjo de até 1024 elementos de processamento com memória local e conexões com dois elementos vizinhos, figura 5.1a. Os elementos de processamento são programáveis e podem executar operações *booleanas*, aritméticas e de vizinhança. Na figura 5.1b, mostra-se um exemplo de erosão binária. Os elementos de processamento são programados através de uma tabela verdade, que é dada em hexadecimal. Na figura 5.1c, apresenta-se um exemplo de uma tabela verdade, o correspondente número hexadecimal e o elemento estruturante utilizado. Outros exemplos de arranjos paralelos são o Illiac IV, que contém um arranjo de 8x8 elementos de processamento e o MPP, contendo 16384 processadores em uma matriz de 128x128.

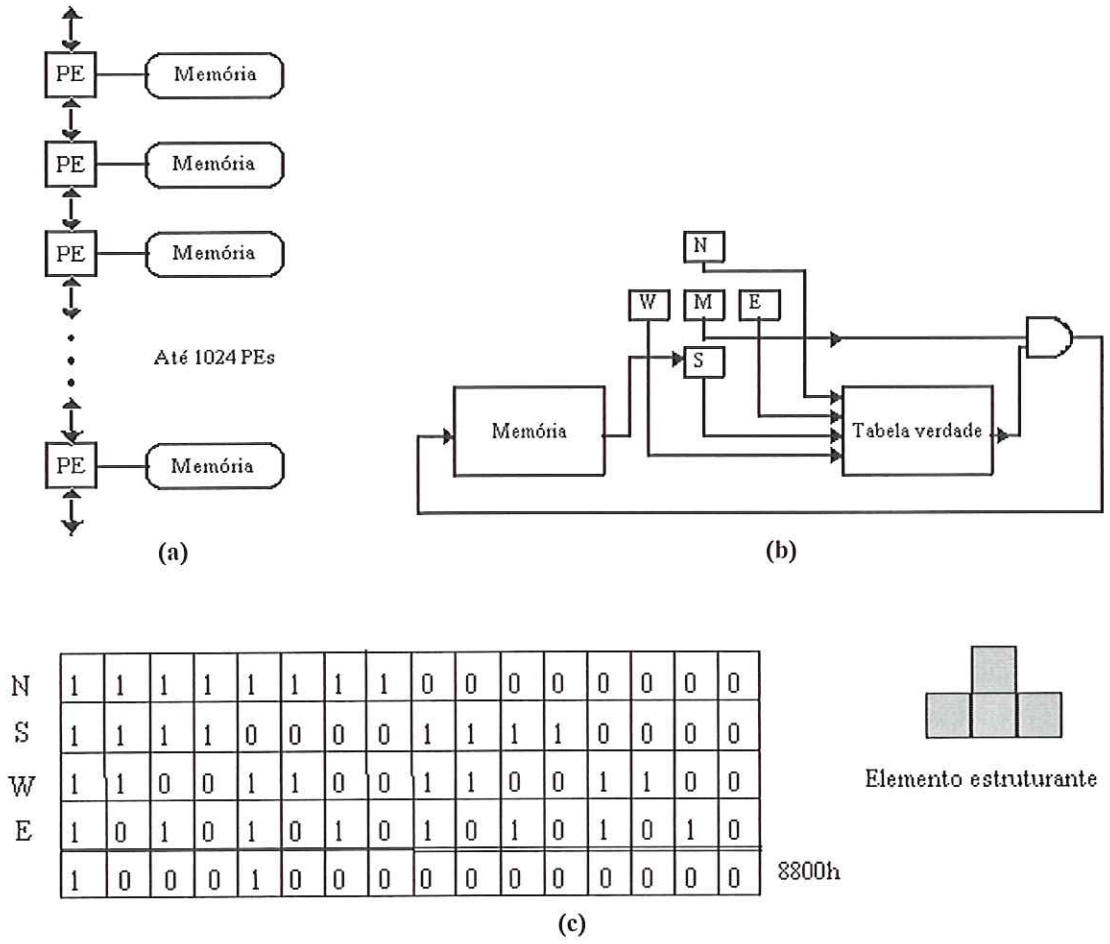


FIGURA 5.1 – Sistema AIS-5000. (a) Topologia do arranjo paralelo. (b) Implementação de erosão binária. (c) Elemento estruturante e tabela verdade utilizada no exemplo.

Arquiteturas sistólicas ou arranjos sistólicos, são as arquiteturas mais amplamente utilizadas para processamento de imagens em baixo nível. O arranjo é utilizado para implementar um certo algoritmo em hardware. Nesses arranjos, os elementos de processamento são rígidos. Os dados fluem pela memória de forma rítmica, passando pelos elementos de processamento e retornando novamente à memória. A sincronização dos dados é obtida através de um clock global, que controla todos os elementos de processamento.

As arquiteturas MIMD são divididas basicamente em arquiteturas de memória-distribuída (anel, malha, pirâmide e hipercubo) e arquiteturas de memória-compartilhada. As arquiteturas de memória-distribuída possuem diversos processadores interconectados de várias maneiras. Cada processador se comunica com sua própria memória local e também com processadores adjacentes. Na figura 5.2, apresentam-se alguns exemplos de arquiteturas com memória-distribuída. Na figura 5.2a, tem-se uma arquitetura com topologia em anel. Esta topologia é apropriada para um número limitado de nós e é adequada para algoritmos que não demandam grande quantidade de dados. Na figura 5.2b, apresenta-se uma arquitetura com topologia mais sofisticada que é a topologia em malha. Esta pode ser aplicada a algoritmos que envolvem matrizes, tais como processamento de imagens em 2D. Uma das desvantagens dessa topologia é a necessidade de comunicação entre dois nós que não pertençam a mesma vizinhança. Uma extensão para esta abordagem é a topologia piramidal. Esta é formada por diversos arranjos em malha, sobrepostos e de resoluções menores, figura 5.2c.

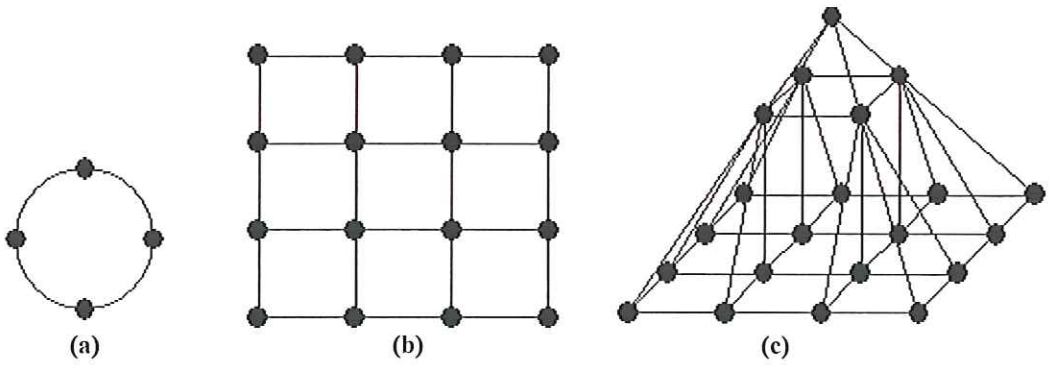


FIGURA 5.2 – Topologias de memória distribuída. (a) Topologia em anel. (b) Topologia em malha. (c) Topologia piramidal.

A topologia piramidal resolve o problema encontrado na topologia em malha, de acordo com a discussão anterior. Uma outra topologia mais complexa é a hipercubo. Nesta topologia, os nós são localizados nos cantos de um cubo de dimensão n. Uma máquina de dimensão d, construída nesta arquitetura, têm 2^d nós, com cada um deles sendo conectado aos seus d nós vizinhos. Na figura 5.3, apresentam-se alguns exemplos.

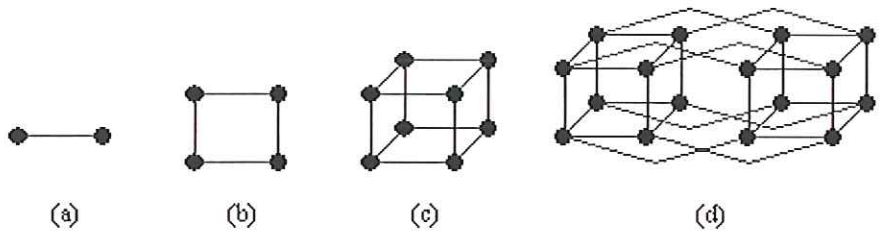
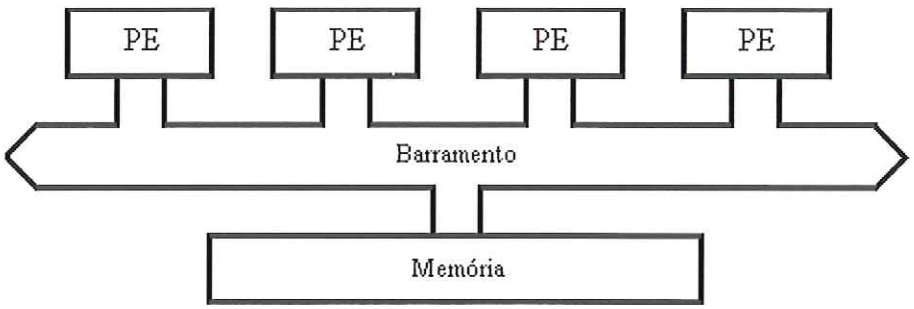
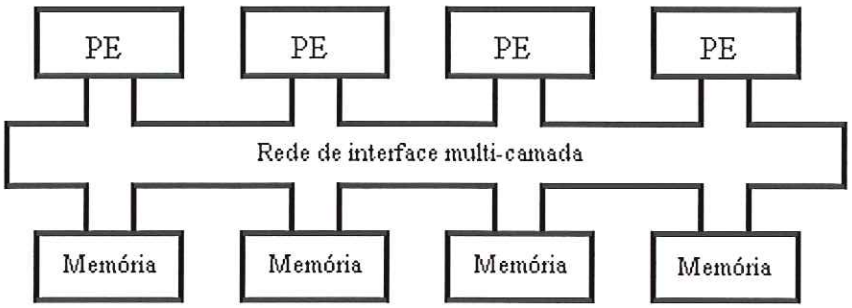


FIGURA 5.3 – Topologias hipercubo. (a) Hipercubo 1D. (b) Hipercubo 2D. (c) Hipercubo 3D. (d) Hipercubo 4D.

Em arquiteturas de memória-compartilhada, os elementos de processamento são controlados por uma memória comum. Basicamente, existem dois modos de se interfacear os elementos de processamento com a memória. O modo mais simples consiste de um barramento rápido onde os elementos de processamento e a memória são ligados, figura 5.4a. Estas máquinas podem suportar até 30 elementos de processamento. O modo mais sofisticado possui uma rede de interface multi-camada, que é responsável por interconectar os elementos de processamento com pedaços de memória, figura 5.4b. A principal vantagem de máquinas com memória-compartilhada é a facilidade de implementação. Elas são usadas em processamento de imagens em alto nível. Embora estas máquinas são aplicáveis para processamento paralelo de imagens, onde cada elemento de processamento é responsável por uma certa partícula de imagem, elas não são tão suficientes quanto as SIMD. Estas arquiteturas não são eficientes quanto à velocidade.



(a)



(b)

FIGURA 5.4 – Arquiteturas de memória-compartilhada. (a) Barramento rápido. (b) rede de interface multi-camada.

5.2.2 Estruturas de Hardware Especializadas para Processamento Morfológico de Imagens

Além das arquiteturas de propósito geral discutidas na seção anterior, outras arquiteturas particularmente dedicadas para processamento morfológico de imagens foram relatadas na literatura. As técnicas para implementação de estruturas de hardware especializadas para processamento morfológico podem ser classificadas nas seguintes categorias:

- Implementações digitais: técnica de decomposição por *threshold*, técnica *bit serial*, outras implementações de arranjos sistólicos e implementações assíncronas.
- Implementações analógicas.
- Implementações Ópticas.

A metodologia da técnica de decomposição por *threshold*, introduzida por (FITCH et al., 1985), levou ao desenvolvimento da teoria de filtros *stack*. Um filtro *stack* é qualquer filtro que pode ser definido usando-se a decomposição por *threshold* e uma função *booleana* positiva (PBF). De acordo com esta aproximação, um sinal multinível que obtém seus valores no conjunto {0, 1,..., M-1} é decomposto em M-1 sinais binários. Cada um

desses sinais é filtrado através de uma PBF. A mesma PBF é aplicada a cada sinal binário. A saída do filtro *stack* é composta pela soma de todos os binários após a filtragem. Uma PBF é uma função *booleana*, que é implementada como uma soma lógica de produtos lógicos. Na figura 5.5, ilustra-se a técnica de decomposição por *threshold*. Na figura, um sinal x unidimensional é decomposto em três sequências binárias, x_1, x_2 e x_3. Cada uma destas sequencias é filtrada por um filtro de *max* binário com uma janela de tamanho 3x1, isto é, uma porta OR de três entradas. Assim, as saídas binárias, y_1, y_2 e y_3 , são obtidas respectivamente de acordo com a fórmula: $y_i(t)=x_i(t-1)+x_i(t)+x_i(t+1)$, i=(1..3). Estas saídas são somadas por colunas e o resultado final, y, é obtido.

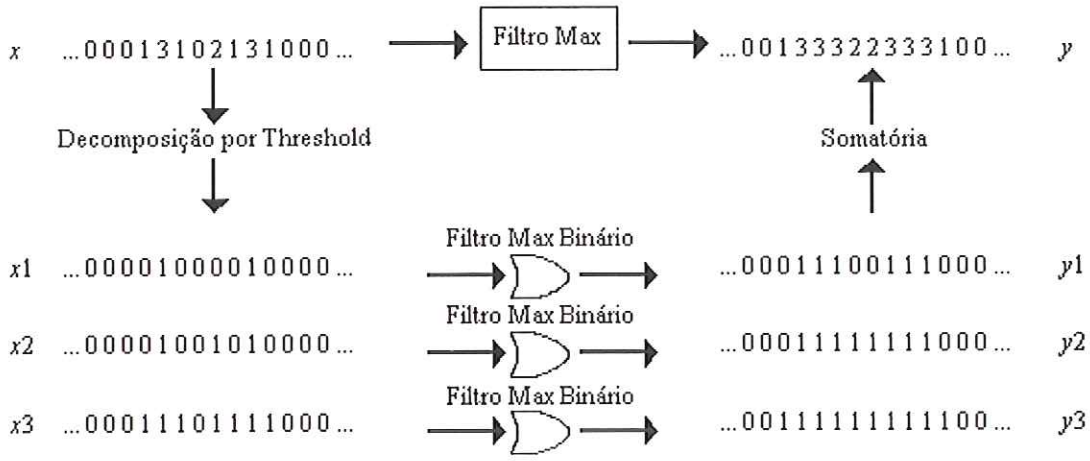


FIGURA 5.5 – Técnica de decomposição por *threshold* para um sinal unidimensional.

Esta técnica foi utilizada por (SHIH e MITCHELL, 1989) em morfologia clássica com a finalidade de decompor problemas complexos em problemas mais simples, ou seja, imagens em níveis de cinza são decompostas em múltiplas imagens binárias, e estas são processadas em paralelo por meio de um hardware dedicado, aumentando-se assim o desempenho do processamento morfológico. Este algoritmo, para o caso de dilatação em nível de cinza, funciona basicamente da seguinte maneira: primeiramente é preciso encontrar os valores máximos de níveis de cinza na imagem e no elemento estruturante. Assim, seja P o maior valor de intensidade na imagem e Q o maior valor de intensidade no elemento estruturante. A imagem deve ser decomposta em P imagens binárias e o elemento estruturante em $Q+1$ imagens binárias. Então, os pares de conjuntos binários resultantes da

decomposição devem ser inseridos no estágio de dilatação da seguinte forma: a primeira imagem binária e o Q_{th} elemento estruturante decomposto; a segunda imagem e o Q_{th} elemento estruturante, a segunda imagem e o elemento estruturante *(Q-1)*; a terceira imagem e o Q_{th} elemento estruturante, a terceira imagem e o $(Q-1)_{th}$ elemento estruturante, a terceira imagem e o $(Q-2)_{th}$ elemento estruturante, e assim por diante. Considerando-se que cada par é processado independentemente, estas operações podem ser implementadas em paralelo. Após a saída do estágio de dilatação, os conjuntos resultantes devem ser somados *bit* a *bit*, assim, obtendo-se P saídas. Se $P \geq Q$, então, acrescenta-se 1 a cada posição da $(Q+2)_{th}$ saída, acrescenta-se 2 a cada posição da $(Q+3)_{th}$ saída, e assim por diante. O resultado final é obtido selecionando-se os valores máximos de cada posição das P saídas e acrescentando-se Q a cada posição. Uma implementação em ASIC para dilatações em níveis de cinza, de acordo com esta técnica, é relatada em (ANDREADIS et al., 1996). Nesta implementação, as resoluções da imagem e do elemento estruturante são limitadas a 4 bits. Este é um dos principais problemas desta técnica, a complexidade do hardware cresce de acordo com o fator $O(2^{2b}+2^{b})$, onde b é a resolução dos pixels. Portanto, a aplicação desta técnica para imagens de altas resoluções se torna impraticável. Outras técnicas surgiram com o propósito de reduzir a complexidade do hardware envolvido, como por exemplo, a técnica híbrida. Nesta técnica, a decomposição por *threshold* não é aplicada a ambos, o elemento estruturante e a imagem, primeiramente é realizada uma operação morfológica entre o elemento estruturante e a imagem, e a partir do resultado desta operação se aplica a técnica de decomposição por *threshold*. Nesta técnica, a complexidade do hardware é reduzida a menos que a metade.

Outra técnica para encontrar de forma eficiente os valores de máximo ou de mínimo em uma operação morfológica, conhecida como *bit serial*, é apresentada em (CHEN, 1989). Muitos pesquisadores implementaram este algoritmo em VLSI para executar as operações básicas de morfologia matemática (DIAMANTARAS e KUNG, 1997; GASTERATOS et al., 1996 e LUCK e CHAKRABATY, 1995). De acordo com este algoritmo, os diferentes *bits* de n números são processados por elementos de processamento dedicados em diferentes estágios de um arranjo sistólico. Suponha que se queira encontrar os valores de máximo ou de mínimo entre n números apresentados. No primeiro estágio do processo os

bits mais significativos desses números são processados. Os números cujos *bits* mais significativos são 1 ou 0, são candidatos a serem máximo ou mínimo, assim, desprezam-se os demais números para os próximos estágios do processo. Um sinal de *flag*, f, classifica os números em duas categorias. Mais especificamente, se f=1, o número é um candidato no próximo estágio do processo, caso contrário, o número é excluído do processo. Este procedimento é então repetido no próximo estágio do processo para os *bits* de menor ordem dos números candidatos. Uma ilustração deste procedimento para encontrar o valor máximo de cinco números de 4 *bits* pode ser vista na tabela 5.1. Na tabela 5.1, x_i(i=1..5) são os cinco números, $b_{j,i}$ é o j_{th} bit da representação binária de x_i e $f_{j,i}$ é o flag do i_{th} número no j_{th} estágio.

TABELA 5.1 – Exemplo ilustrativo do algoritmo *bit serial*.

i	x_i	$f_{0,i}$	$b_{1,i}$	$f_{1,i}$	$b_{2,i}$	$f_{2,i}$	$b_{3,i}$	$f_{3,i}$	$b_{4,i}$	$f_{4,i}$
1	3	1	0	0	0	0	1	0	1	0
2	5	1	0	0	1	0	0	0	1	0
3	11	1	1	1	0	0	1	0	1	0
4	12	1	1	1	1	1	0	1	0	1
5	8	1	1	1	0	0	0	0	0	0
m=	12	m_1=	1	m_2=	1	m_3=	0	m_4=	0	

As arquiteturas mais comuns em processamento morfológico de imagens são as arquiteturas *pipeline*. Uma arquitetura *pipeline* é adequada para elementos estruturantes que obedeçam a regra da cadeia, uma vez que cada estágio da máquina *pipeline* opera por meio de um elemento estruturante pequeno. Como exemplos de arquiteturas morfológicas *pipeline*, tem-se:

- TAS (*Texture Analyser System*) (KLEIN e SERRA, 1972), que podia executar operações binárias morfológicas em um *pipeline* de três estágios. Originalmente era suportado um único elemento estruturante hexagonal, mas as outras versões podiam executar operações lógicas mais gerais.

- Cytocomputer: desenvolvido no *Environmental Research Institute of Michigan* para processamento de imagens biomédicas (STERNBERG, 1983). O Cytocomputer é constituído de um *pipeline* serial onde cada estágio executa uma transformação simples de uma imagem inteira. Oitenta e oito estágios executam operações lógicas e vinte e cinco outros executam processamento numérico. Registradores de deslocamento dentro de cada estágio armazenam duas linhas adjacentes de *n* pixels e registradores de janela mantém os dados que constituem a entrada 3x3 necessária a função de transição. Estes registradores de janela endereçam uma LUT com 512 possibilidades de valores que serve para implementar a função de transição (GERRITSEN e VERBEEK, 1984 e PRESTON, 1983).

Nas próximas subseções, alguns exemplos de arquiteturas *pipeline* serão destacados mais detalhadamente.

Implementações analógicas para as operações morfológicas podem ser muito atrativas, considerando-se que estas oferecem altas velocidades em chips mais compactos, além do mais, estas não necessitam de conversões A/D e D/A, logo, podem ser acopladas diretamente nos sensores e câmeras, assim, é possível construir instrumentos mais inteligentes (SISKOS et al., 1998).

Uma arquitetura opto-eletrônica para processamento morfológico de imagens foi proposta em (MICHAEL e ARRATHOON, 1997). Esta arquitetura usa o princípio de *pipelining*, onde seus estágios são implementados através de PLAs baseados em fibra óptica (OPLAs). Estes arranjos são caracterizados pelos altos fatores de *fan-in* e *fan-out*, assim, é possível implementar operações morfológicas com elementos estruturantes grandes sem a necessidade de decomposição, logo, a arquitetura projetada possuirá poucos estágios, entretanto, esta abordagem é cara. Os OPLAs são baseados em operações lógicas OR, onde estas são implementadas por meio de fibras e detectores ópticos, também, utilizam-se inversores ópticos. Na figura 5.6, apresenta-se a tabela de programação de um OPLA para a operação de erosão binária por um elemento estruturante 2x2.

Bits de Controle				Entradas				
C1	*C2*	*C3*	*C4*	*Ai,j*	*Ai,j+1*	*Ai+1,j*	*Ai+1,j+1*	*Ei,j*
0	0	0	1	—	—	—	1	1
0	0	1	0	—	—	1	—	1
0	0	1	1	—	—	1	1	1
0	1	0	0	—	1	—	—	1
0	1	0	1	—	1	—	1	1
0	1	1	0	—	1	1	—	1
0	1	1	1	—	1	1	1	1
1	0	0	0	1	—	—	—	1
1	0	0	1	1	—	—	1	1
1	0	1	0	1	—	1	—	1
1	0	1	1	1	—	1	1	1
1	1	0	0	1	1	—	—	1
1	1	0	1	1	1	—	1	1
1	1	1	0	1	1	1	—	1
1	1	1	1	1	1	1	1	1

FIGURA 5.6 – Tabela de programação do OPLA para operação de erosão binária por um elemento estruturante 2x2.

Sejam $a_{i,j}$ e $e_{i,j}$ pixels na imagem original e na erodida, respectivamente na linha i e na coluna j. Assim, o pixel $e_{i,j}$ será igual 1 (isto é, claro) se os pixels claros do elemento estruturante e da imagem coincidirem. Para um padrão 2x2, é possível obter 16 elementos estruturantes diferentes. Quatro bits de controle são utilizados para escolher o elemento estruturante requerido. Na figura 5.7, mostra-se o padrão de conexão em fibra-óptica para a operação de erosão.

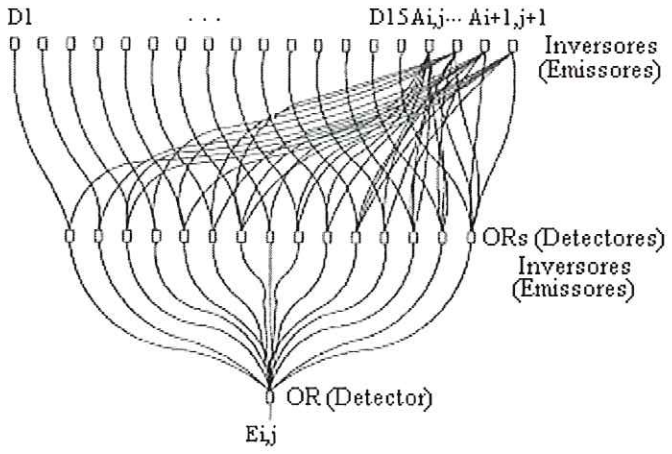


FIGURA 5.7 – Padrão de conexão em fibra-óptica para a operação de erosão.

A arquitetura *pipeline* opto-eletrônica proposta para processamento morfológico de imagens binárias pode ser vista na figura 5.8.

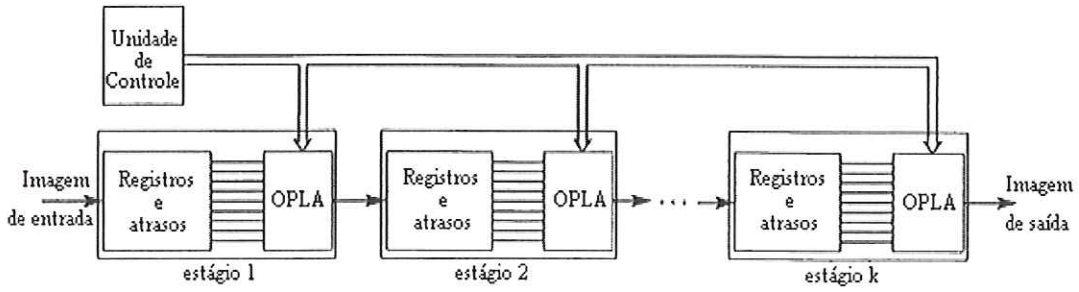


FIGURA 5.8 – Arquitetura *pipeline* utilizando OPLAs, proposta para processamento morfológico de imagens binárias.

5.2.2.1 Arquiteturas *Pipeline* Morfológicas para Processamento de Imagens Binárias

Em uma arquitetura *pipeline* para processamento de vídeo, é conveniente apresentar os dados no formato *raster*, de acordo com o padrão de varredura de uma câmera de vídeo convencional. Em (ABBOTT et al., 1988), propõe-se uma arquitetura *pipeline* para processamento morfológico de imagens binárias, onde cada estágio opera através de um elemento estruturante consistindo-se da origem e de um ponto arbitrário *(r,c)* no plano de imagem. Assim, as seguintes considerações devem ser destacadas:

- $R: E^2 \rightarrow E^1$: função que mapeia uma imagem A_{NxM}, onde N é o número de linhas e M é o número de colunas, em um arranjo $R[A]$ de pixels.
- $R[A](t)$: representação de um pixel particular.
- $R[A](rM+c)$: representação do valor do pixel da r-ésima linha e c-ésima coluna de A, onde: $0 \leq r \leq N-1$ e $0 \leq c \leq M-1$.
- $R[A](t-k)$: imagem A deslocada de k unidades de tempo.
- $R[A](u)$: duração de $R[A]$, onde: $0 \leq u \leq NM-1$.

Assim, a dilatação de uma imagem por um elemento estruturante consistindo-se da origem e de um ponto arbitrário *(r, c)* no plano de imagem pode ser realizada deslocando-se os dados de entrada por $|Mr+c|$ unidades de tempo. Se $Mr+c>0$ (caso causal), a saída pode ser obtida através de um OR lógico da versão deslocada e mascarada da entrada de dados com

a versão não deslocada da entrada e janelando-se o resultado pela duração da imagem original. Se $Mr+c<0$ (caso não causal), a imagem deslocada se tornará a imagem primária. Logo, efetuando-se um OR lógico desta versão deslocada com a versão mascarada da imagem não deslocada e janelando-se o resultado pela duração da entrada de dados deslocada, o resultado desejado será obtido. Também, os mesmos conceitos derivados anteriormente podem ser aplicados à erosão, assim, a erosão de uma imagem por um elemento estruturante consistindo-se da origem e de um ponto arbitrário (r, c) no plano de imagem pode ser efetuada deslocando-se os dados de entrada por $|Mr+c|$ unidades de tempo. Se $Mr+c>0$ (caso não causal), o resultado desejado será obtido através de um AND lógico da versão mascarada dos dados de entrada com a entrada deslocada e janelando-se o resultado pela duração da imagem deslocada. Se $Mr+c<0$ (caso causal), o resultado desejado será obtido fazendo-se um AND lógico da entrada não deslocada com a versão mascarada da entrada deslocada e janelando-se o resultado pela duração da imagem de entrada. A arquitetura *pipeline* conceitual para realizar estas operações está ilustrada na figura 5.9. Estas idéias podem ser generalizadas para elementos estruturantes de n-pontos, entretanto, a complexidade do hardware envolvido para o processamento das operações morfológicas será aumentada.

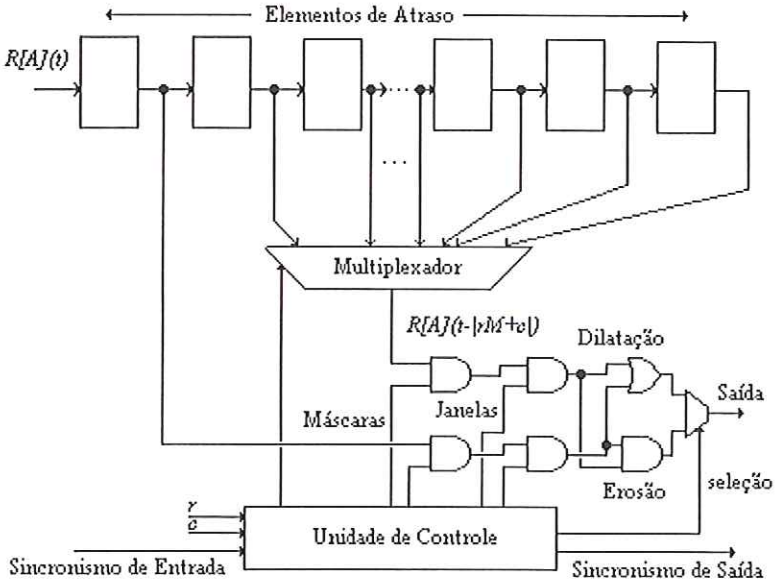


FIGURA 5.9 – Arquitetura *pipeline* conceitual para operações morfológicas.

Em (KOJIMA et al., 1994), apresenta-se uma arquitetura para processamento morfológico de imagens binárias onde a complexidade das operações básicas de morfologia pode ser reduzida através do uso de um algoritmo de decomposição dimensional do elemento estruturante 2-D. O algoritmo apresentado possui as seguintes características:

- Não há restrições quanto ao tamanho do elemento estruturante.
- Possibilidade do elemento estruturante possuir qualquer forma arbitrária.
- Alta velocidade de processamento.

Considerando-se um elemento estruturante circular como exemplo, o algoritmo supracitado se divide nos seguintes passos:

- De acordo com a figura 5.10(a), o elemento estruturante circular deve ser aproximado através de retângulos, formando-se assim o elemento estruturante B.
- Conforme mostrado na figura 5.10(b), os retângulos são definidos como b_i (i=1,2,..n), onde: $B = \bigcup_{i=1}^{n} b_i$.
- Conforme ilustrado na figura 5.10(c), cada b_i é decomposto em dois elementos estruturantes unidimensionais, assim, $b_i = b_{ix} \oplus b_{iy}$.
- Consequentemente, $B = \bigcup_{i=1}^{n} (b_{ix} \oplus b_{iy})$. Portanto, o elemento estruturante original B é decomposto em $2n$ elementos estruturantes unidimensionais.

De acordo com as discussões acima, tem-se: $A \oplus B = \bigcup_{i=1}^{n} (A \oplus b_{ix} \oplus b_{iy})$ e $A \Theta B = \bigcap_{i=1}^{n} (A \Theta b_{ix} \Theta b_{iy})$. Na figura 5.10(c), l_{ix} e l_{iy} são os comprimentos dos elementos estruturantes unidimensionais; s_{ix} e s_{iy} representam a distância entre a origem e o ponto extremo do elemento estruturante 1-D mais próximo à origem. Estes parâmetros são utilizados pelo hardware.

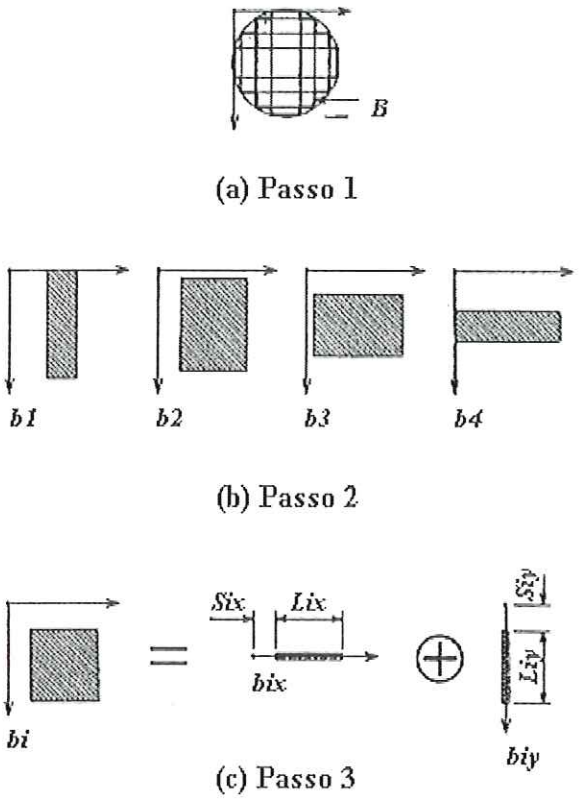


FIGURA 5.10 – Decomposição do elemento estruturante.

As figuras 5.11 e 5.12 ilustram respectivamente, o circuito de dilatação 1-D e o circuito conversor X-Y de direção de varredura. Na figura 5.13 é apresentado um exemplo contendo um diagrama de tempo de um sinal de entrada e o respectivo sinal de saída do circuito de dilatação 1-D. Assim, o sinal de saída permanecerá em nível alto quando o sinal de entrada deslocado for alto e durante a contagem decrescente do contador, determinada pelo parâmetro *l*, após a descida do sinal de entrada.

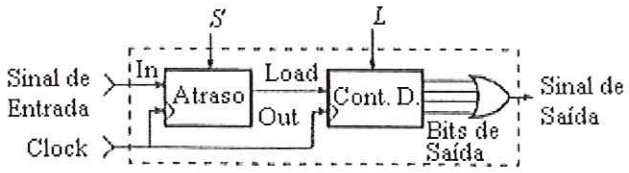


FIGURA 5.11 – Circuito de Dilatação 1-D.

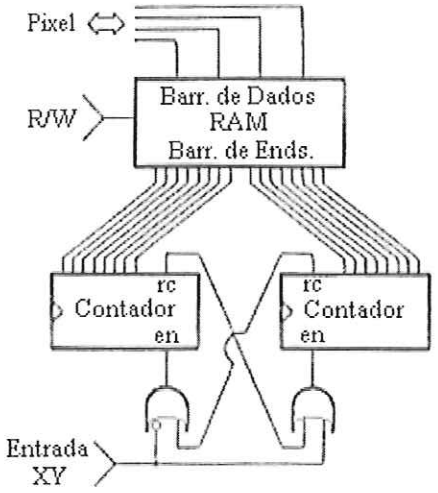


FIGURA 5.12 – Circuito conversor de direção de varredura.

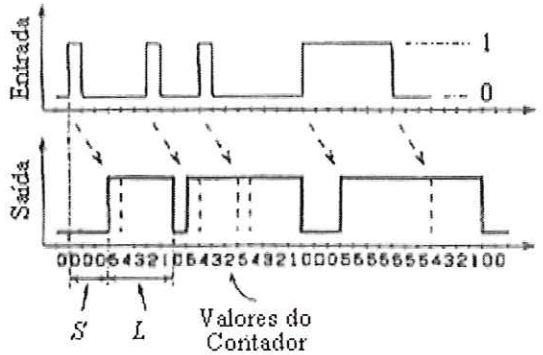


FIGURA 5.13 – Exemplo de Diagrama temporal do circuito de dilatação 1-D.

Quando o circuito conversor de varredura se encontra no modo X, tem-se: $S=s_{ix}$ e $L=l_{ix}$. Neste caso, a varredura é realizada na direção x. Idem para a direção y. No circuito de dilatação 1-D, o sinal de entrada sequencial é deslocado pela variável de atraso, S, e então é transferido à entrada *load* do contador decrescente. Quando *load* for igual a 1, o valor de L é carregado no contador decrescente, e quando este mudar para nível 0, a saída do contador será decrementada através de cada pulso de *clock* até que sua saída seja 0. Assim, a saída do circuito de dilatação 1-D permanecerá em nível lógico 1 após a carga de *load* e manterá este estado até que o valor do contador seja nulo. O circuito conversor de varredura faz a varredura da imagem nas direções x e y. Dois contadores de 8 bits são utilizados para gerar os endereços às RAMs. Assim, é possível se lidar com imagens de 256x256 pixels. A comutação de direção é realizada através da troca dos bytes mais e menos significativos do barramento de 16 bits da RAM. De acordo com a figura 5.12, esta operação pode ser implementada através do uso de contadores de 8 bits com pinos de *ripple carry* (*rc*) e *enable* (*en*). A dilatação 2-D é obtida paralelizando-se n-sistemas de dilatação 1-D e memórias conforme a figura 5.14. Nesta figura, a imagem de entrada é varrida na direção x, dilatada pelos elementos estruturantes unidimensionais e os resultados obtidos são armazenados nas RAMs. Em seguida, o processo de varredura das RAMs se dá na direção y, e novamente as imagens são dilatadas pelos circuitos de dilatação unidimensionais. Finalmente, o resultado da dilatação 2-D pode ser obtido através de uma operação binária OR entre cada saída dos circuitos de dilatação 1-D. O processo de erosão é obtido de maneira análoga, de acordo com a figura 5.15. Um dos problemas encontrados por este algoritmo é que se a forma do elemento estruturante necessitar de muitos retângulos para sua aproximação, o que não é difícil na prática, seu desempenho computacional diminui.

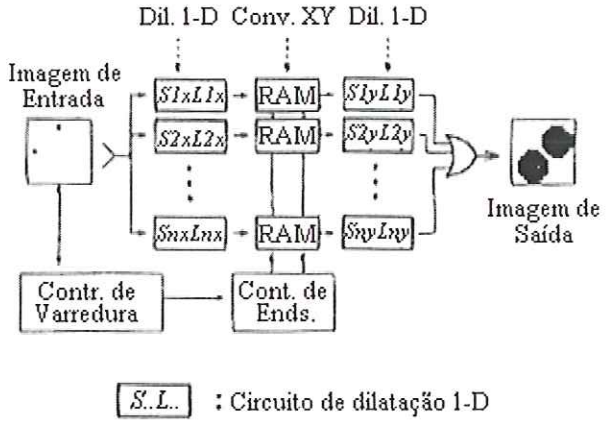


FIGURA 5.14 – Arquitetura para dilatação binária 2-D.

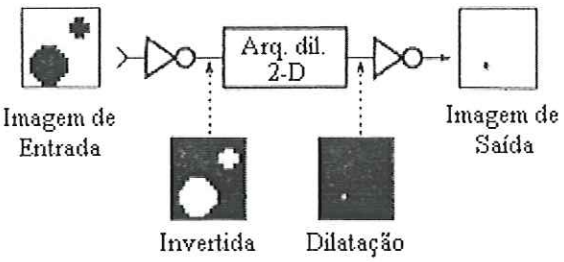


FIGURA 5.15 – Circuito para erosão binária.

5.3 Considerações Finais

Operações morfológicas aplicadas à imagens em níveis de cinza são difíceis de se implementar em tempo real. O problema encontrado em uma implementação direta das operações básicas de morfologia matemática para imagens em níveis de cinza é que a busca dos valores de máximo e de mínimo não são tão eficientes neste tipo de implementação. Assim, outras técnicas de implementação em hardware para processamento morfológico de imagens foram propostas na literatura. A técnica de decomposição por *threshold*, discutida anteriormente, é mais rápida que a técnica *bit serial*, entretanto, a complexidade do hardware envolvido é elevada. A complexidade do hardware na técnica de decomposição por *threshold* cresce exponencialmente de acordo com a resolução de pixel e o tamanho do elemento estruturante. A técnica direta cresce linearmente com ambos, a resolução de pixel e o tamanho do elemento estruturante. Para imagens com maior resolução de pixel, a técnica *bit serial* é mais atrativa que a técnica de decomposição por *threshold*. Também, foram propostas na literatura arquiteturas *pipeline* para processamento morfológico de imagens binárias em tempo real, podendo-se destacar as que utilizam a decomposição do elemento estruturante e as ópticas.

6 ARQUITETURA *PIPELINE* PARA PROCESSAMENTO MORFOLÓGICO DE IMAGENS BINÁRIAS EM TEMPO REAL UTILIZANDO CPLDS

6.1 Considerações Iniciais

Neste capítulo, será apresentado o projeto por nós desenvolvido de implementação de uma arquitetura *pipeline* dedicada para dilatação e erosão de imagens binárias em tempo real utilizando CPLDs. A arquitetura desenvolvida é capaz de processar imagens binárias de 512x512 pixels. As imagens digitais são obtidas através da amostragem de um sinal de vídeo composto padrão fornecido por uma câmera CCD. Cada estágio da arquitetura é responsável por processar um subconjunto menor de um elemento estruturante arbitrário dado. Estes estágios foram implementados através de CPLDs, permitindo-se assim, a reprogramação da forma e do tamanho dos elementos estruturantes e também do número de estágios da arquitetura. Uma unidade de controle, também implementada através de um CPLD, gera os sinais de sincronismo e de controle para cada estágio da arquitetura, assim, é possível realizar operações morfológicas mais complexas. Na figura 6.1, a arquitetura sobredita é apresentada em sua forma básica.

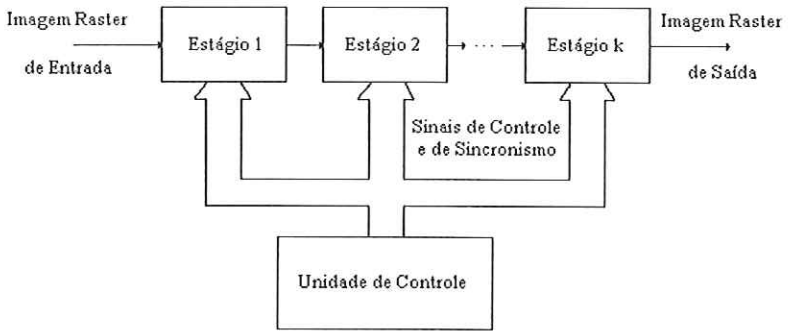


FIGURA 6.1 – Arquitetura *pipeline* desenvolvida.

6.2 Descrição do Projeto

A figura 6.2 apresenta o diagrama em blocos completo da arquitetura desenvolvida. O objetivo desta seção é apresentar de forma detalhada todas as etapas de desenvolvimento deste projeto.

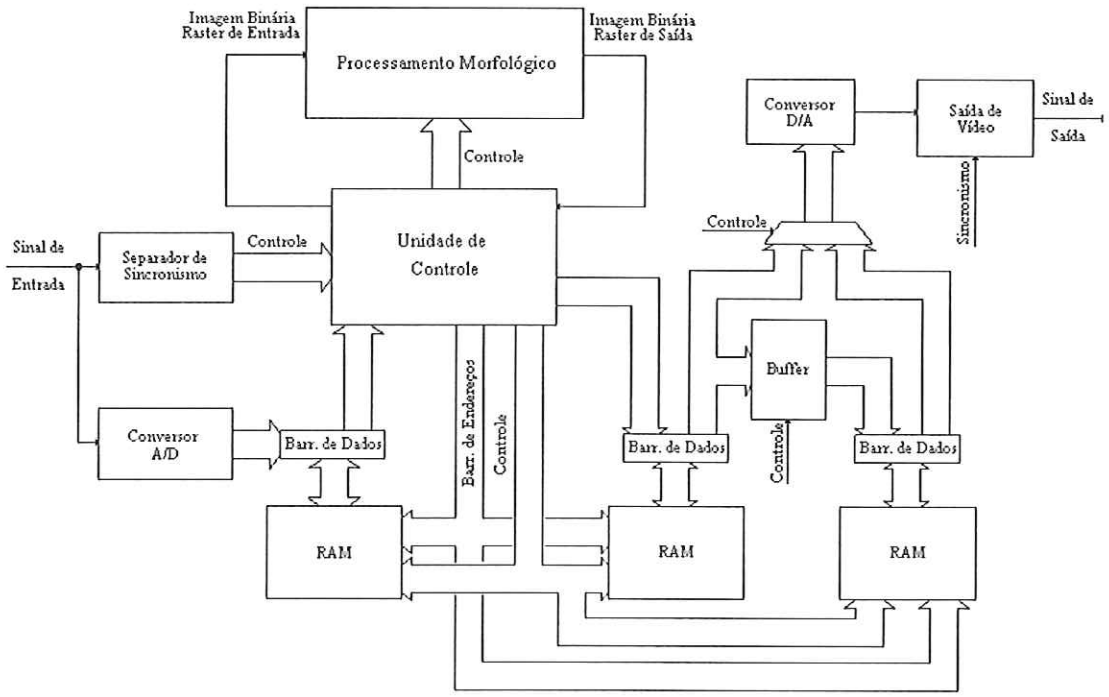


FIGURA 6.2 – Diagrama em blocos da arquitetura desenvolvida.

Para sua operação, a arquitetura realiza o desentrelaçamento do sinal de vídeo composto fornecido pela câmera CCD. Assim, as imagens em níveis de cinza são convertidas em imagens binárias através do processo de limiarização simples e inseridas sequencialmente (formato *raster*) no estágio de processamento morfológico (arquitetura *pipeline*). Após este estágio, as imagens binárias processadas são novamente entrelaçadas, gerando-se assim, um sinal de vídeo composto para apresentação em um monitor de vídeo ou em outro sistema convencional compatível com o padrão de vídeo composto. Nas próximas seções, cada estágio desta arquitetura será explanado de forma pormenorizada.

6.2.1 Separador de Sincronismo

O separador de sincronismo utilizado no projeto desta arquitetura foi o EL4581C da Elantec, cuja função é extrair as informações de sincronismo vertical, horizontal, pórtico posterior (bp) e de tipo de campo (par ou ímpar) de um sinal de vídeo composto padrão. Conforme visto no capítulo 4, estes sinais são relevantes porque através deles a unidade de controle identifica o início de uma nova linha de vídeo ou de um novo campo, gerando assim os sinais de controle e os endereços às memórias. Dessa forma, as operações de leitura ou escrita são efetuadas nos tempos adequados. Na figura 4.5, pode-se ver o formato dos sinais de saída deste chip dado um sinal de vídeo composto padrão de entrada.

6.2.2 Conversor A/D

Considerando-se que a informação de uma linha de vídeo é de aproximadamente 53μs, para se obter 512 amostras neste tempo é necessário utilizar um conversor A/D capaz de realizar a conversão em aproximadamente 100ns, o que equivale a 10 milhões de amostras por segundo. Neste projeto, foi utilizado o conversor rápido AD830 de 8 bits da Texas Instruments, adequado para aplicações de vídeo. Este conversor opera com +5V e sua frequência máxima de amostragem é de 60MHz (TEXAS INSTRUMENTS, 2001). A implementação deste conversor utilizou um amplificador operacional para acertar o nível de zero do sinal de entrada.

Na figura 6.3, apresentam-se os diagramas de tempo do referido conversor. Conforme ilustrado na figura 6.2, as informações digitalizadas por este conversor são armazenadas na RAM segundo os sinais de controle e endereços gerados pela unidade de controle.

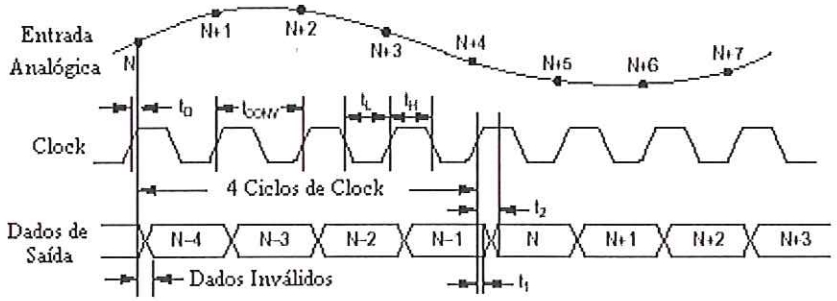


FIGURA 6.3 – Diagramas de tempo do conversor ADS830. De acordo com a figura, o dado amostrado fica disponível após quatro ciclos de clock.

6.2.3 Memórias

Considerando-se que o sinal de vídeo é entrelaçado, utilizou-se memórias RAMs estáticas para o armazenamento dos campos par e ímpar. A memória RAM adotada para este projeto foi a TC551001 de 70ns e 128k palavras de 8 bits da Toshiba. Os diagramas de tempo dessa memória para os ciclos de leitura e de escrita podem ser vistos na figura 4.7. A unidade de controle gera os sinais de controle e endereços às memórias segundo os diagramas apresentados nesta figura, controlando assim as operações de escrita e de leitura.

6.2.4 Unidade de Controle

Uma vez que a unidade de controle é considerada um dos estágios mais importantes desta arquitetura, sendo responsável pela transmissão sincronizada dos sinais de controle e endereços ao circuito, pretende-se nesta subseção abordar seu funcionamento mais detalhadamente. Para se implementar a unidade de controle foram utilizados dois CPLDs EPM7128SLC84-7 da família MAX 7000S da Altera e para sua programação utilizou-se o software MAX+PLUS II, também da Altera. Na figura 6.4, apresenta-se o diagrama esquemático do circuito implementado pelo primeiro CPLD, e na figura 6.5, apresenta-se o

outro diagrama esquemático implementado pelo segundo CPLD. A união dos dois diagramas constitui o circuito completo da unidade de controle. Nas próximas subseções, seus blocos constituintes serão explanados.

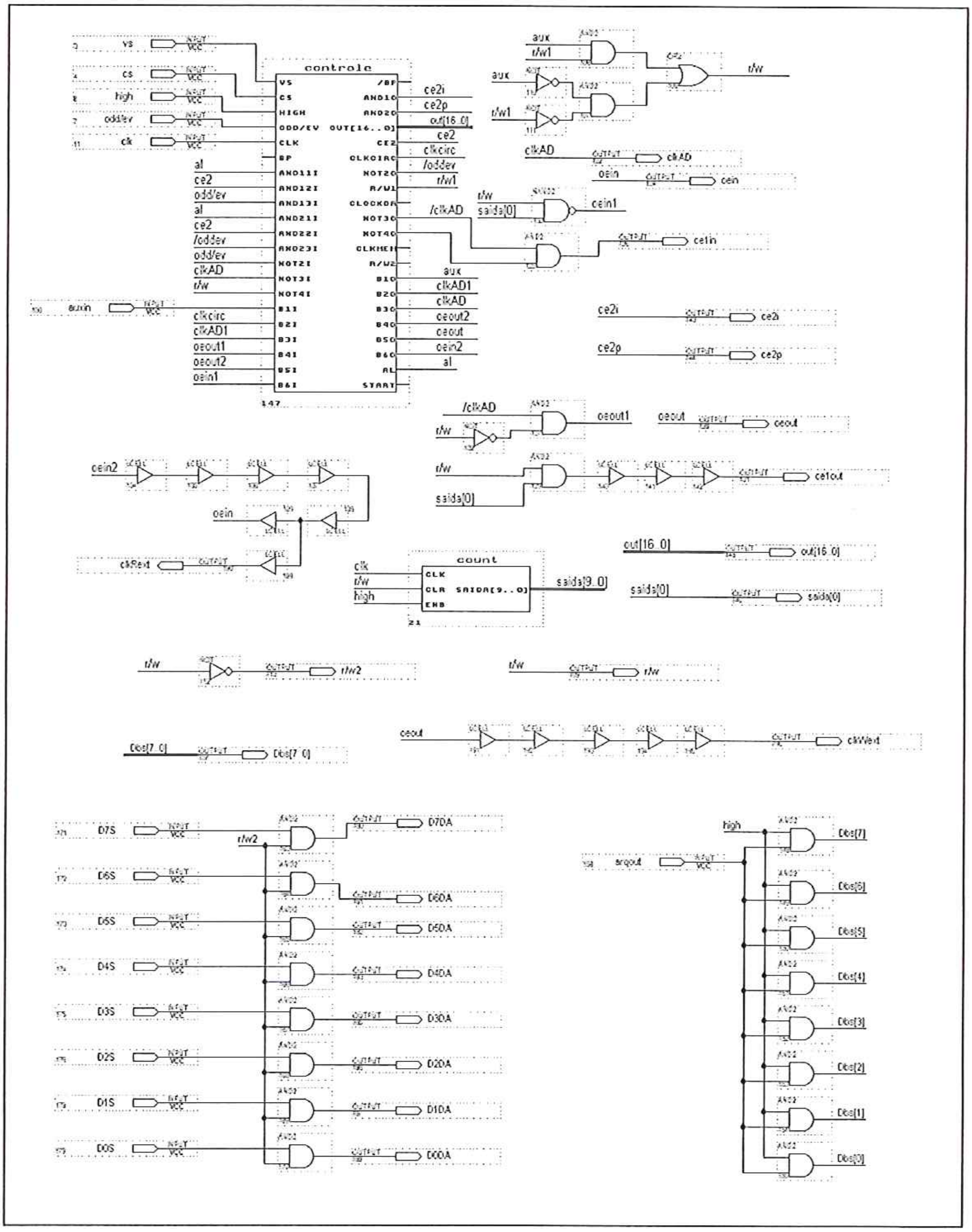


FIGURA 6.4 - Unidade de Controle – CPLD1.

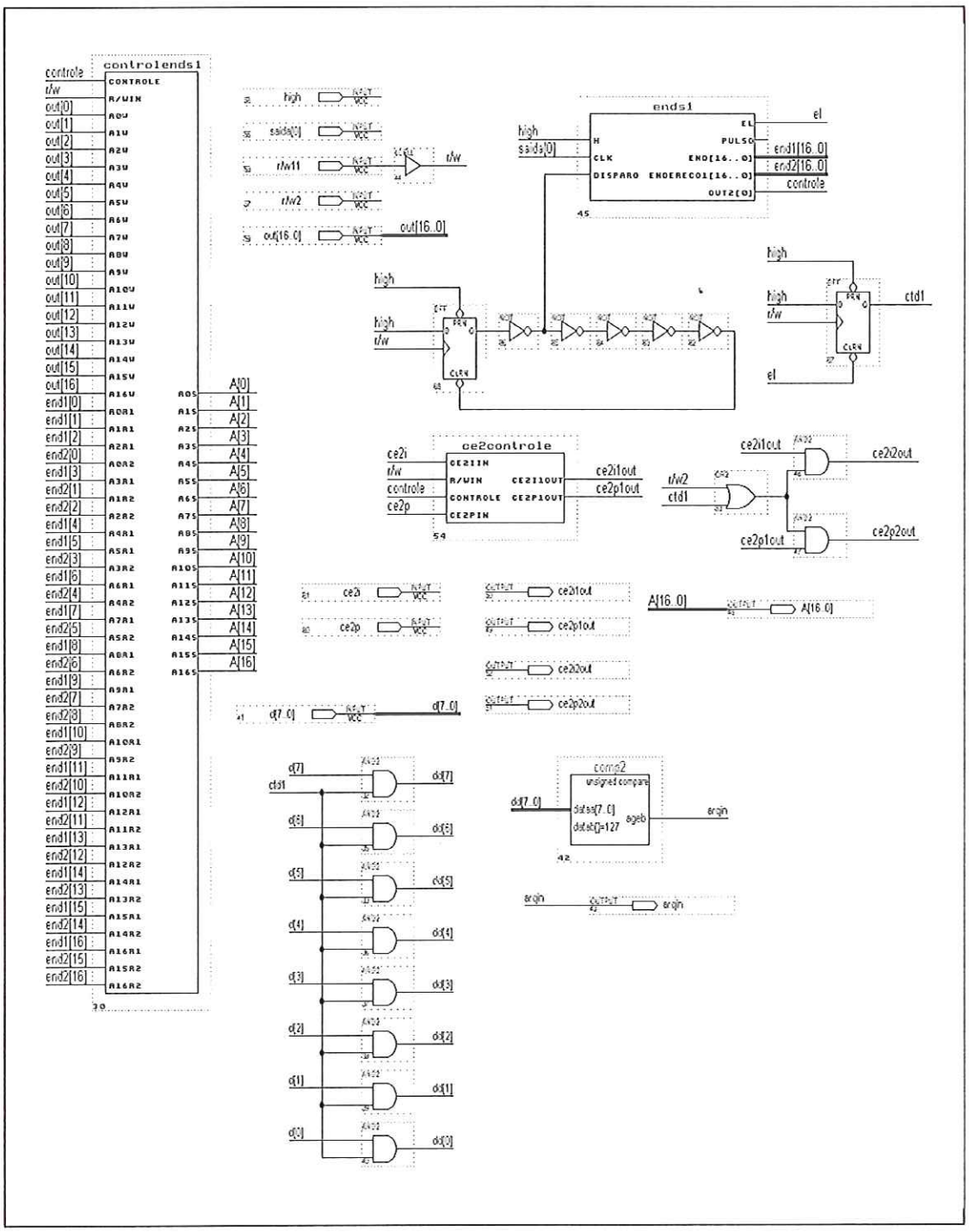


FIGURA 6.5 – Unidade de Controle – CPLD2.

6.2.4.1 Unidade de Controle – CPLD1

Os objetivos deste estágio da unidade de controle, dados os sinais de controle provenientes do separador de sincronismo, são:

- Gerar pulsos de clock para as operações de leitura e de escrita nas memórias.
- Gerar sinais de controle para as operações de leitura e de escrita nas memórias.
- Gerar sinais de controle apropriados para os tipos de campos.
- Gerar sinais de endereçamento às memórias.

Na figura 4.8, apresenta-se o diagrama esquemático do estágio denominado *controle* da unidade de controle corrente. Seu funcionamento é similar ao já apresentado no capítulo 4. Conforme já visto, este estágio possui as seguintes unidades: unidade de exclusão de pulsos de sincronismo, unidade de controle interna e unidade de controle das memórias. A unidade de exclusão de pulsos de sincronismo exclui os pulsos equalizadores e serrilhados fornecidos pelo sinal *cs*. Esta unidade está representada na figura 4.9 e é composta por um contador desenvolvido em HDL (Apêndice B) que é responsável pela contagem dos pulsos de sincronismo após o início de cada campo e por uma máquina de estados, também desenvolvida em HDL (Apêndice C), cuja função é sinalizar à unidade de controle interna *unidadecontr* (Apêndice D) que os pulsos de sincronismo foram excluídos através de um sinal interno denominado *start*. O funcionamento lógico desta unidade é descrito na subseção 4.2.4.1. A função da unidade de controle interna (máquina de estados *unidadecontr*) é gerar um sinal de disparo a um circuito de controle responsável por enviar às memórias um sinal denominado *al* que habilitará as mesmas para as operações de escrita ou de leitura durante os períodos de linha ativa do sinal de vídeo. Seu funcionamento lógico é descrito na subseção 4.2.4.2. A unidade de controle das memórias está representa na figura 4.10 e é responsável pela geração de endereços às memórias e de sinais de controle apropriados para desabilitar as mesmas no final de cada linha de vídeo e também no final da varredura de quadros. Basicamente é formada por um contador de linhas de vídeo desenvolvido em HDL (Apêndice E) e por um contador de pixels (Apêndice F), também desenvolvido em HDL. Seu funcionamento lógico é abordado na subseção 4.2.4.3.

6.2.4.2 Unidade de Controle – CPLD2

Os objetivos mais relevantes deste estágio da unidade de controle são:

- Gerar endereços e sinais de controle apropriados para as memórias.
- Converter as imagens de níveis de cinza para imagens binárias através de um circuito comparador utilizando a técnica de limiarização simples.

Basicamente, este estágio é composto de quatro unidades, de acordo com a figura 6.5. A unidade *ends1* é considerada a mais importante. A idéia desta unidade é gerar os endereços de forma sequencial às memórias durante as operações de leitura, ou seja, lê-se uma linha da memória de campo ímpar e em seguida se lê uma linha da memória de campo par, repetindo-se o processo para as demais linhas de cada memória. Assim, os pixels são enfileirados para serem inseridos no estágio de processamento morfológico. Na figura 6.6, apresenta-se o circuito desta unidade.

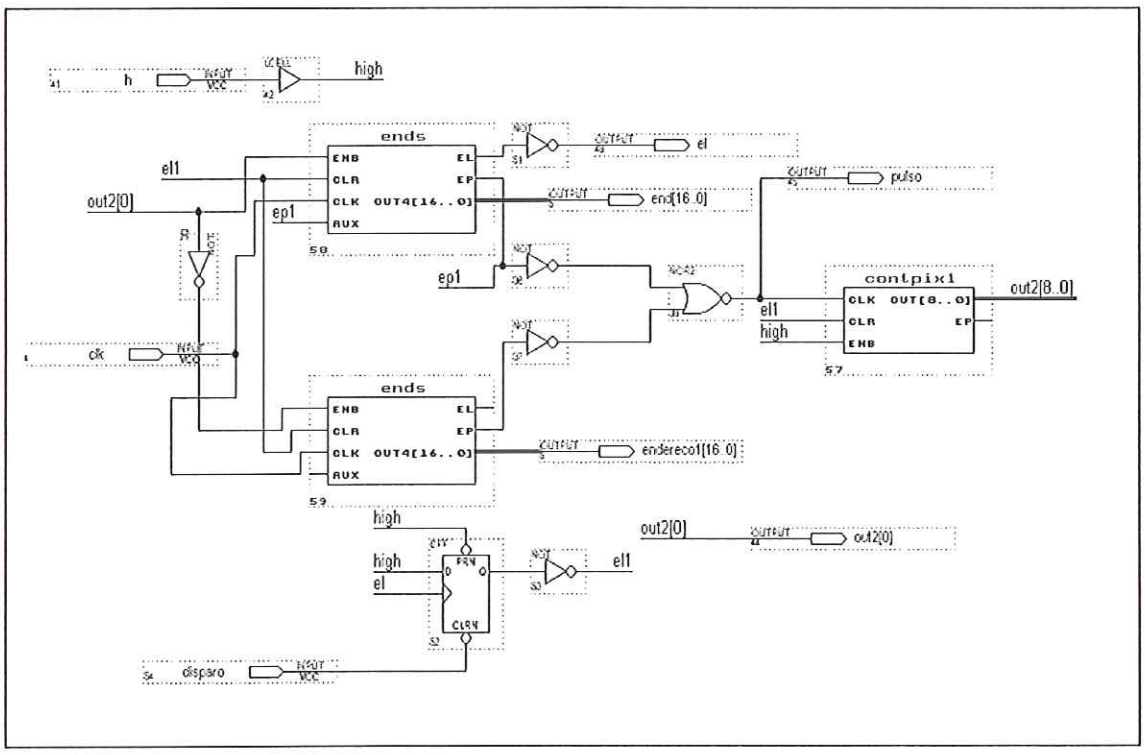


FIGURA 6.6 – Unidade *ends1* – Unidade de Controle - CPLD2.

Quando o sinal *r/w* muda de 0 para 1, durante as operações de leitura das memórias, é gerado um sinal de disparo à unidade *ends1* através de um *flip-flop* tipo *D*, assim, o contador *contpix1* da figura 6.6 será habilitado e através do sinal *out2[0]*, habilitará ou desabilitará dois contadores geradores de endereços às memórias denominados *ends*. O diagrama esquemático destes contadores pode ser visto na figura 6.7. Quando um dos contadores atinge o valor máximo de contagem para cada linha de vídeo, gera-se um pulso ao contador *contpix1*, habilitando-se assim o outro contador responsável pela leitura da próxima linha de vídeo subjacente. No final de um quadro, um dos contadores produzirá um sinal denominado *el*, que indicará o final da operação de leitura nas memórias.

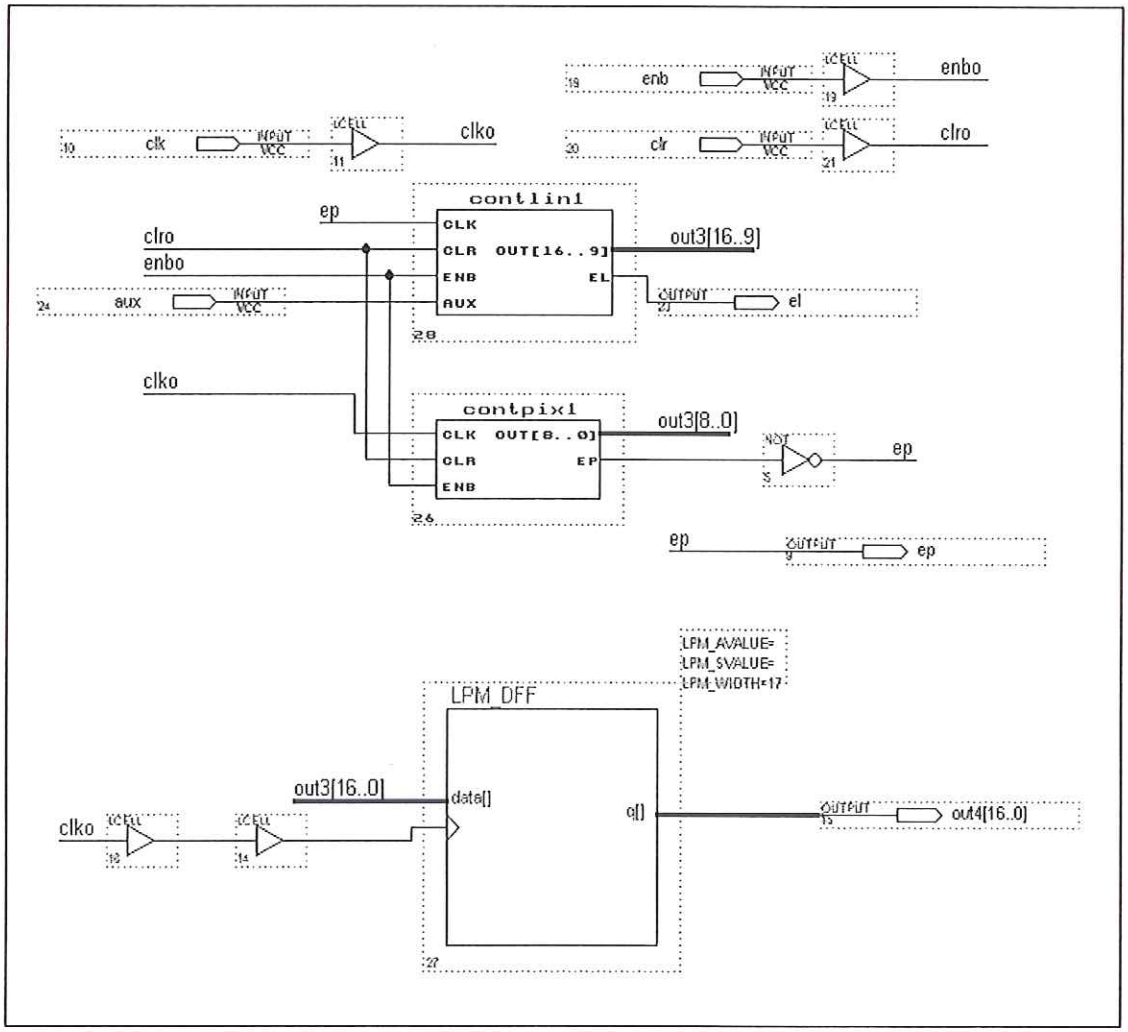


FIGURA 6.7 – Circuito gerador de endereços *ends* – Unidade *ends1* – Unidade de Controle – CPLD2.

Na figura 6.8, apresenta-se o diagrama esquemático da unidade *ce2controle*. Seu papel é gerar os sinais de controle para as memórias de campo ímpar e de campo par durante as operações de escrita e de leitura. Durante as operações de leitura, esta unidade é controlada pelo sinal *controle* enviado pela unidade *ends1*.

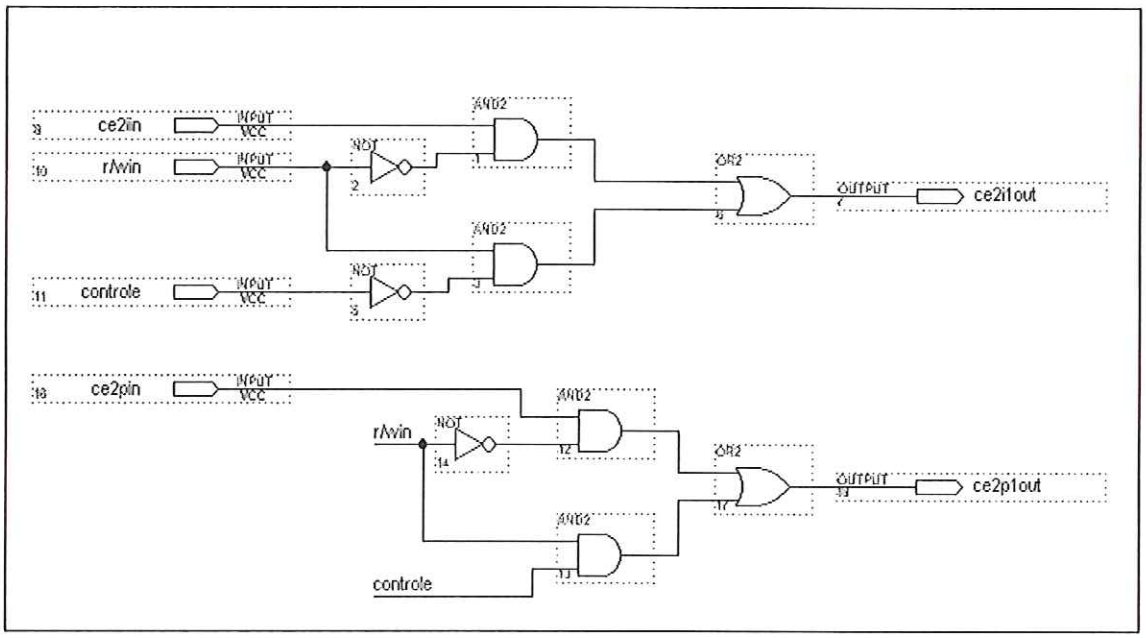


FIGURA 6.8- Unidade *ce2controle* – Unidade de Controle – CPLD2.

Na figura 6.9, apresenta-se a unidade *controlends1* e na figura 6.10 uma de suas unidades básicas. A unidade *controlends1* gera os endereços para as memórias de acordo com o nível lógico do sinal *r/w*. Assim, se o sinal *r/w* estiver em nível lógico 0, os endereços gerados pelo circuito do CPLD1 passam diretamente à arquitetura, caso contrário, passam à arquitetura, os endereços gerados pelo circuito do CPLD2, conforme o nível do sinal *controle*. Por fim, a unidade *comp2* produz um valor de nível lógico 1 para todo pixel de entrada maior ou igual a 127. No Apêndice K é possível ver uma simulação da unidade de controle criada através do programa Max+Plus II da Altera.

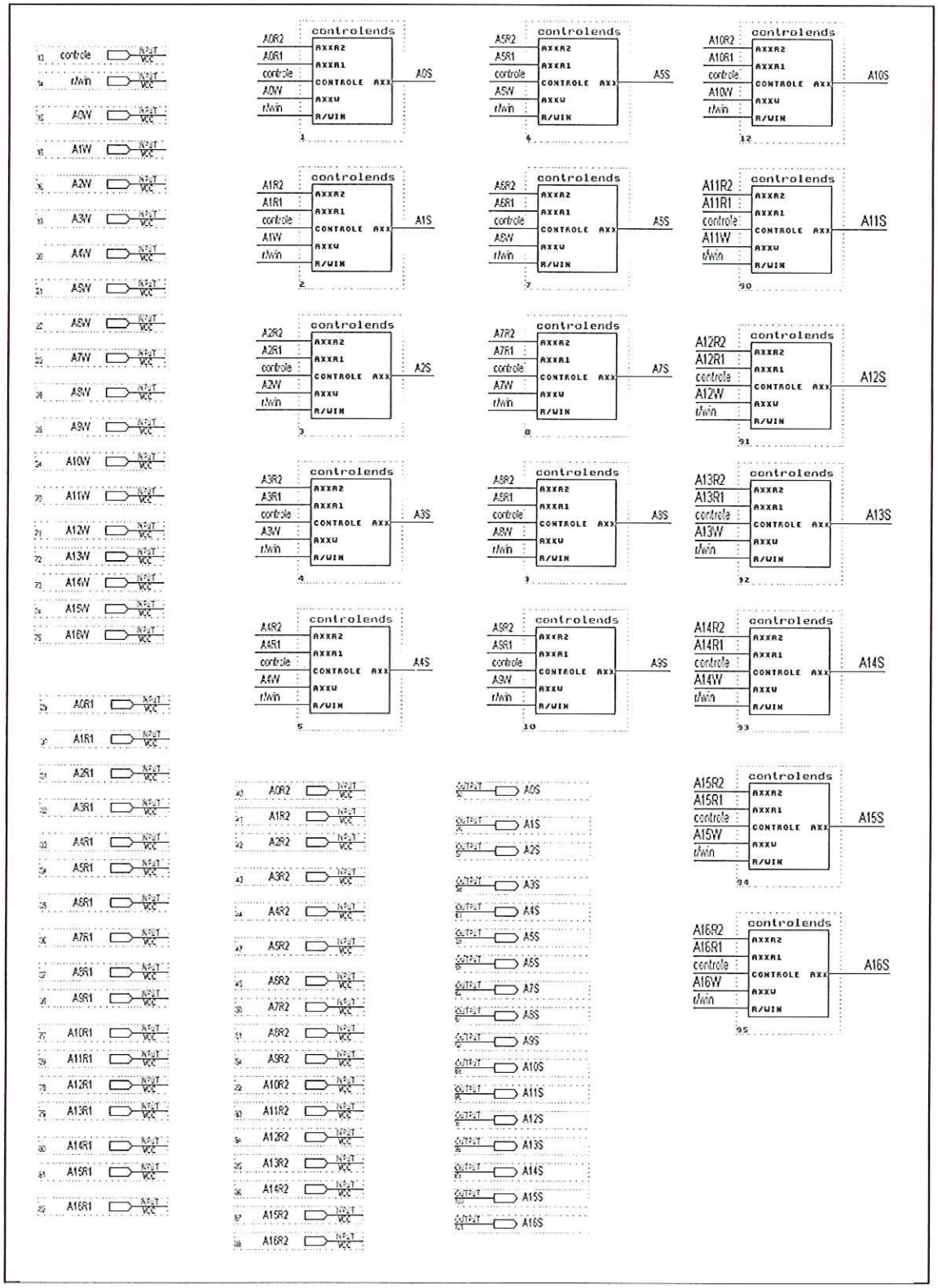


FIGURA 6.9 – Unidade *controlends1* – Unidade de Controle – CPLD1.

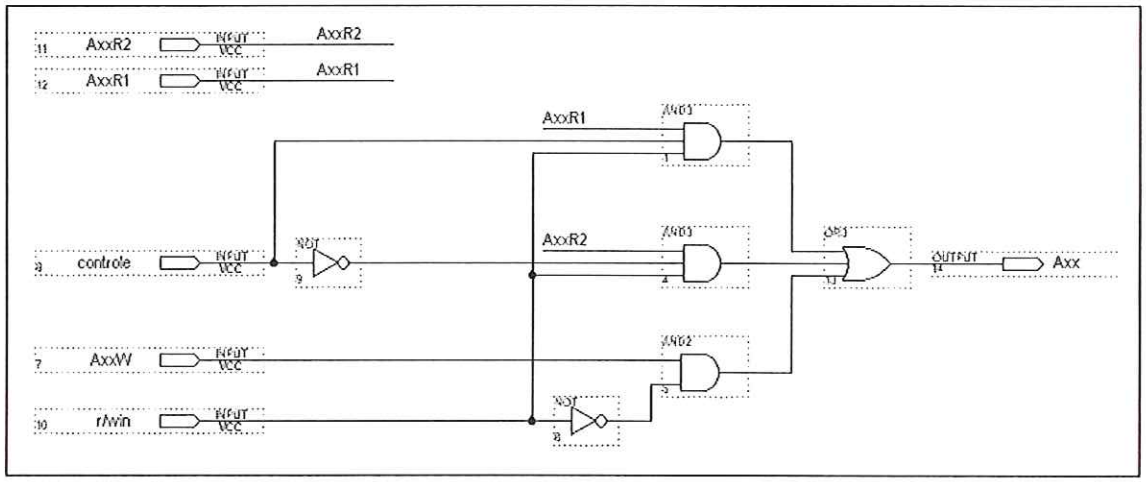


FIGURA 6.10 – Célula básica da unidade *controlends1*.

6.2.5 Unidade de Processamento Morfológico

O objetivo desta unidade é realizar um processamento morfológico em tempo real sobre imagens binárias de entrada no formato *raster* através do uso de dispositivos lógicos programáveis de alta capacidade. Assim, implementa-se através de múltiplos estágios em uma arquitetura *pipeline* serial as operações básicas de morfologia matemática (dilatação e erosão). Dessa forma, é possível se reduzir a complexidade das operações morfológicas envolvidas. Dependendo da capacidade lógica do dispositivo programável utilizado é possível se trabalhar com elementos estruturantes de forma e tamanho arbitrários. Considerando-se que esta unidade pode ser reprogramada, inclusive no próprio circuito, há diversas possibilidades de implementação para as operações supracitadas. Uma das possibilidades para a implementação de um estágio básico de processamento nesta unidade pode ser vista na figura 6.11. Nesta implementação, registradores de deslocamento, criados através de *megafunctions* customizadas pelo software Quartus II da Altera, armazenam duas linhas adjacentes de 512 pixels e registradores de janela mantém os dados que constituem a entrada 3x3, neste exemplo, necessários às operações morfológicas desejadas, criadas através de portas lógicas. Outra possibilidade para se implementar as operações sobreditas seria através de uma tabela (*Look Up Table* – LUT), que neste caso específico, seria endereçada pelos registradores de janela e haveria 512 possibilidades de valores para

o elemento estruturante em questão, aumentando-se portanto a flexibilidade de escolha quanto à sua forma.

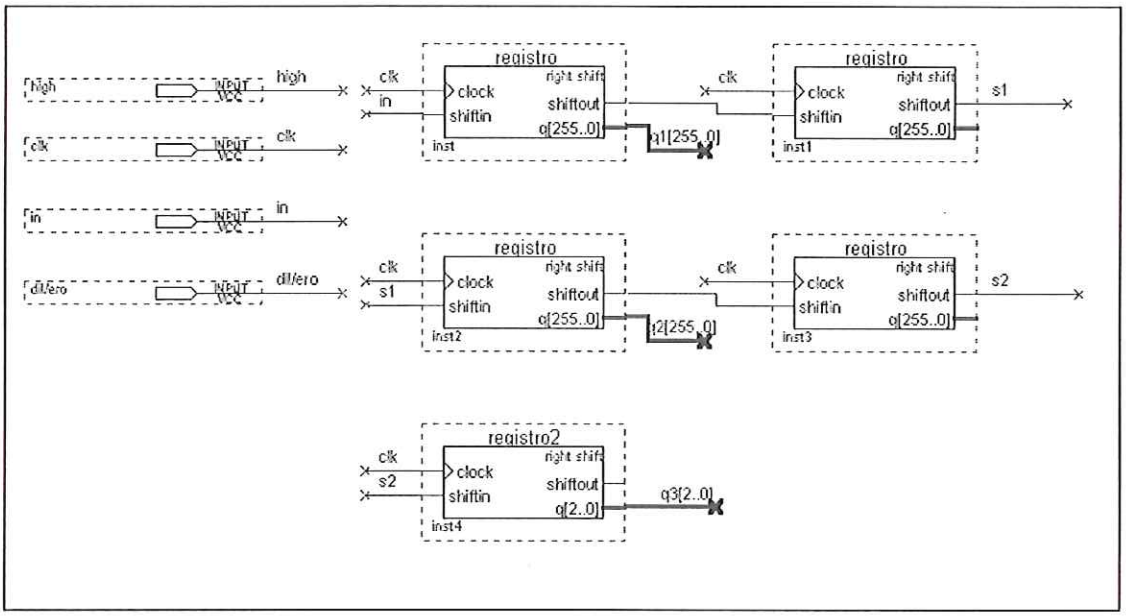


FIGURA 6.11(a) – Registradores de deslocamento utilizados para a armazenagem de duas linhas adjacentes de 512 pixels.

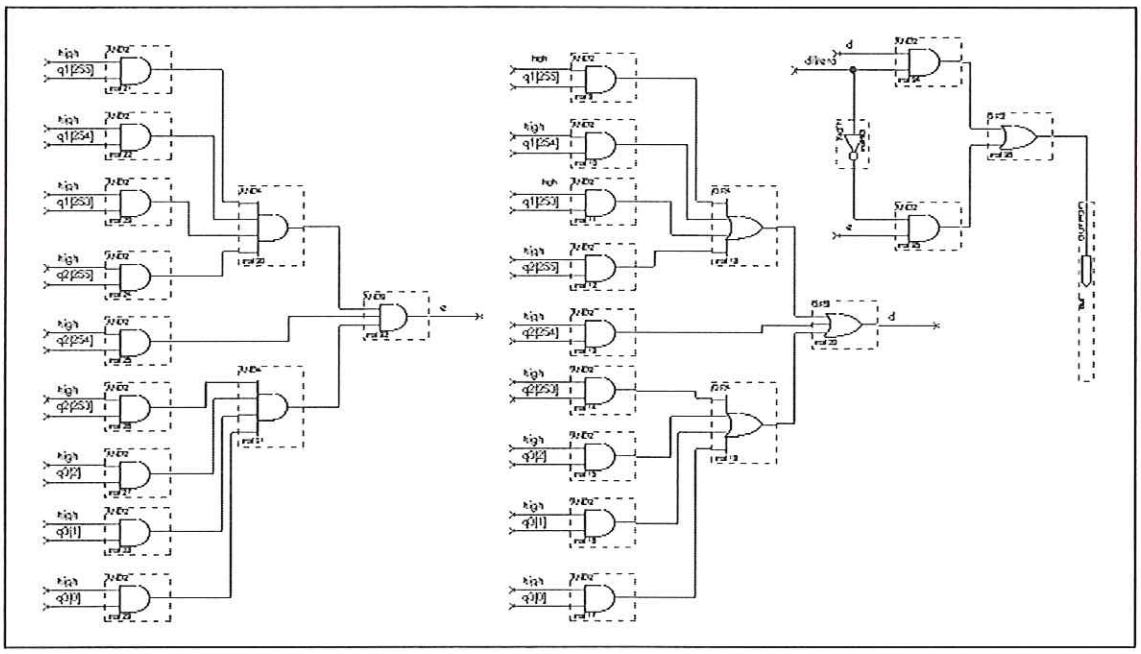


FIGURA 6.11(b) – Implementação das operações morfológicas básicas utilizando portas lógicas, para um elemento estruturante quadrado 3x3.

Nas figuras 6.12 e 6.13, apresentam-se alguns resultados de simulações, dados sinais arbitrários de entrada, realizadas no software Quartus II da Altera para a implementação esquematizada na figura 6.11 e também para implementações suportando elementos estruturantes maiores possuindo outras formas.

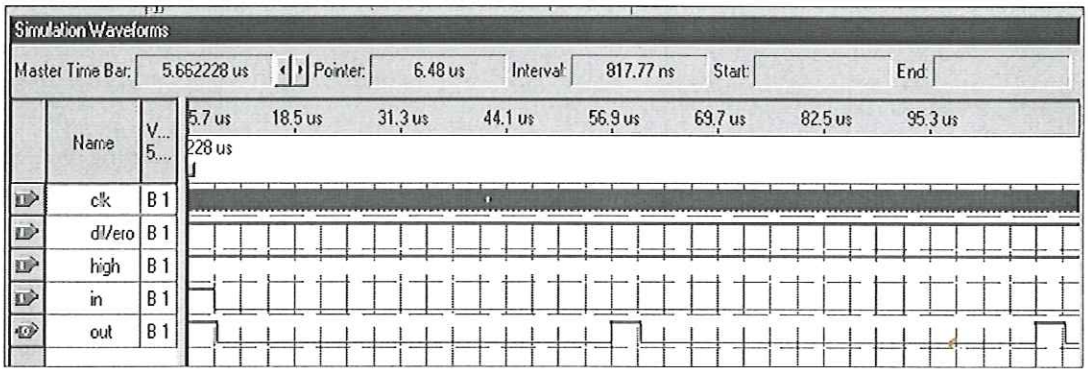


FIGURA 6.12(a) – Dilatação de um sinal arbitrário por um elemento estruturante quadrado 3x3.

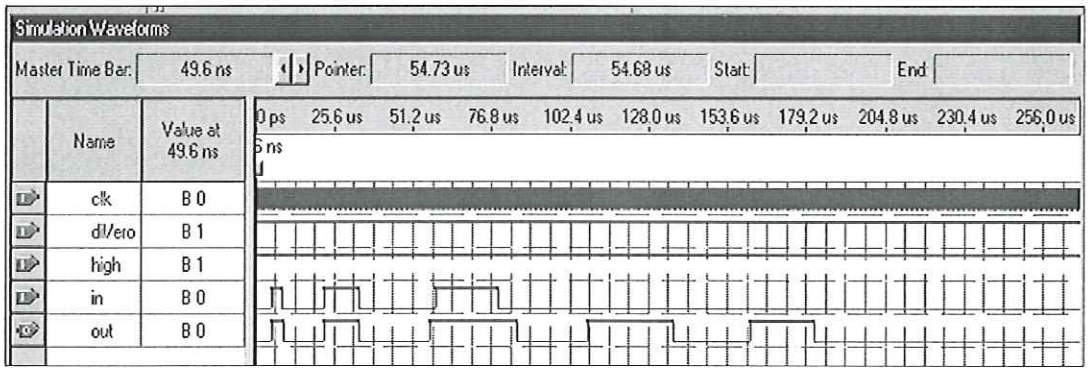


FIGURA 6.12(b) – Dilatação de uma imagem arbitrária constituída de aproximadamente cinco linhas de vídeo por um elemento estruturante quadrado 3x3.

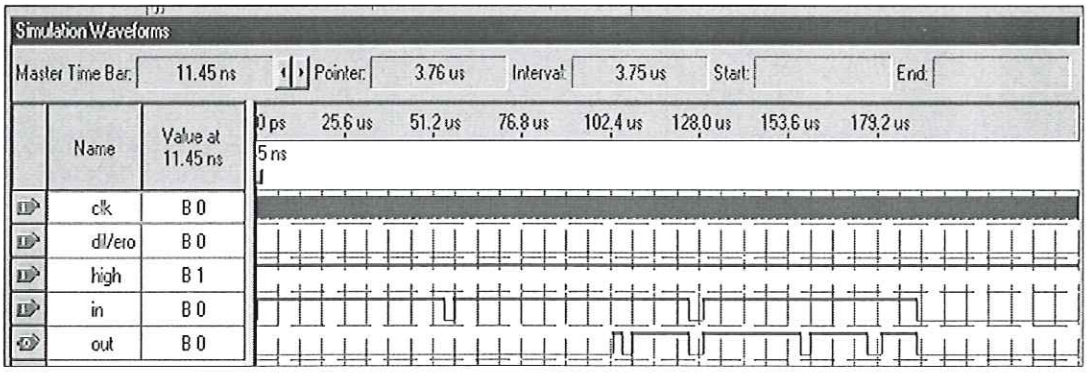


FIGURA 6.12(c) – Erosão de um sinal arbitrário por um elemento estruturante quadrado 3x3.

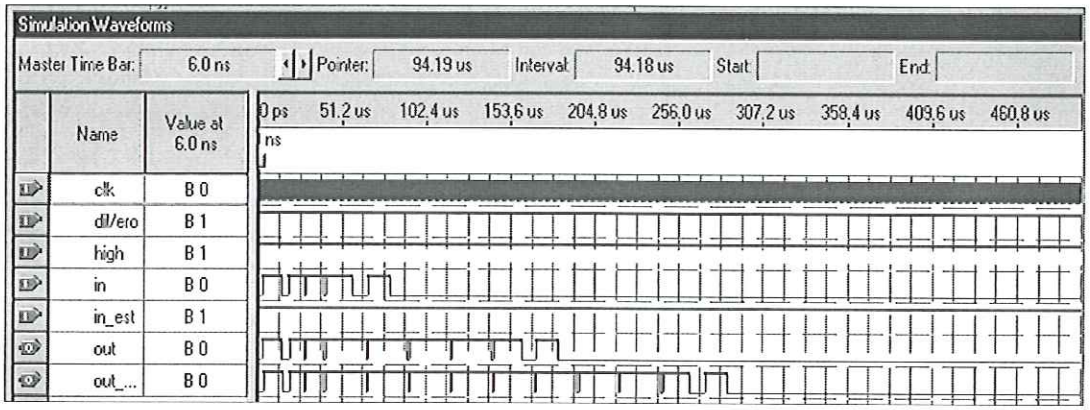


FIGURA 6.12 (d) – Dilatação de um sinal arbitrário por um elemento estruturante circular 5x5.

Na figura 6.13(a), apresenta-se uma possível configuração *pipeline* utilizando os estágios supracitados, realizada através do software Quartus II da Altera, para dilatação de imagens por um elemento estruturante quadrado 13x13. O resultado desta simulação pode ser visto na figura 6.13(b). O dispositivo escolhido para esta simulação foi o EP200EQC208 da Altera.

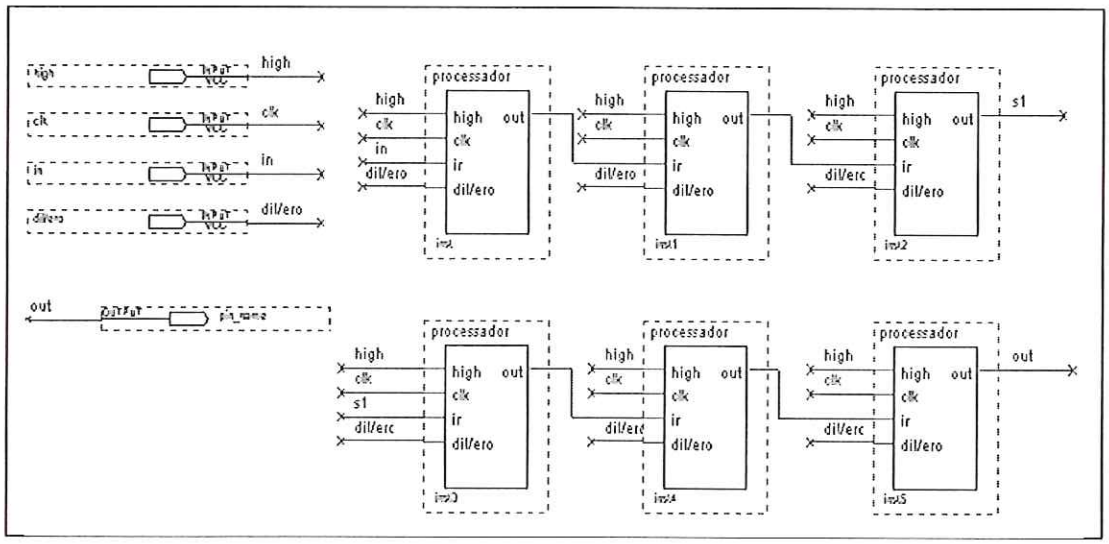


FIGURA 6.13(a) – Arquitetura *pipeline* de 6 estágios para dilatação de imagens binárias utilizando um elemento estruturante quadrado 13x13.

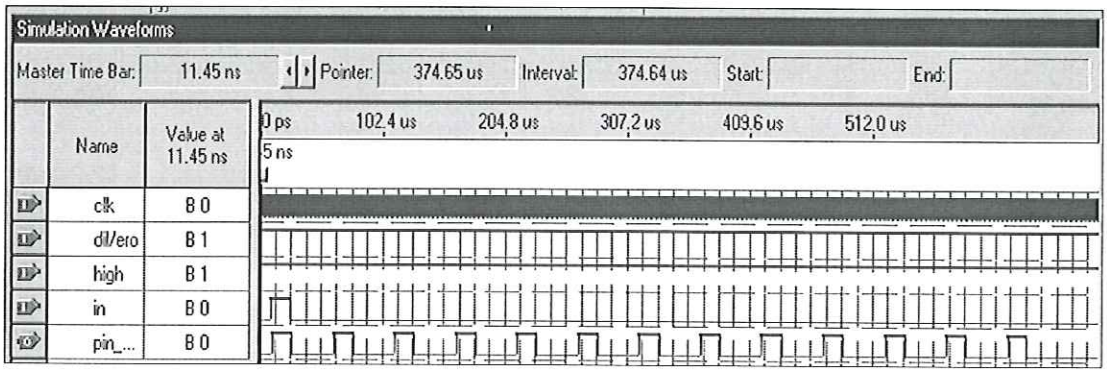


FIGURA 6.13(b) – Dilatação de um sinal arbitrário por um elemento estruturante quadrado 13x13.

Neste projeto, foram implementadas algumas operações morfológicas básicas utilizando estágios suportando elementos estruturantes convexos 3x3. Utilizou-se para as implementações, o dispositivo EPF10K20RC240 da família FLEX 10K da Altera. Os resultados obtidos são apresentados na seção 6.3.

6.2.6 Conversor D/A

O Conversor D/A utilizado neste projeto também foi o CA3338 de 8 bits da Intersil, discutido na subseção 4.2.5. Este conversor utiliza uma rede R-2R, sendo específico para aplicações que envolvam sinais de vídeo e pode operar numa taxa de até 50 MSPS. Seu diagrama de tempos pode ser visto na figura 4.11.

6.2.7 Saída de Vídeo

O objetivo desta unidade é reconstituir o sinal de vídeo composto na saída, dados o sinal de saída do conversor D/A e o sinal de sincronismo composto proveniente do separador de sincronismo. Este circuito foi implementado através de transístores SMDs e pode ser visto no diagrama esquemático do Apêndice A.

6.3 Resultados Obtidos

Nesta subseção serão apresentados alguns resultados obtidos através da arquitetura implementada em hardware. Nas figuras 6.14a a 6.14h são apresentados resultados de algumas operações morfológicas obtidos com a arquitetura desenvolvida.

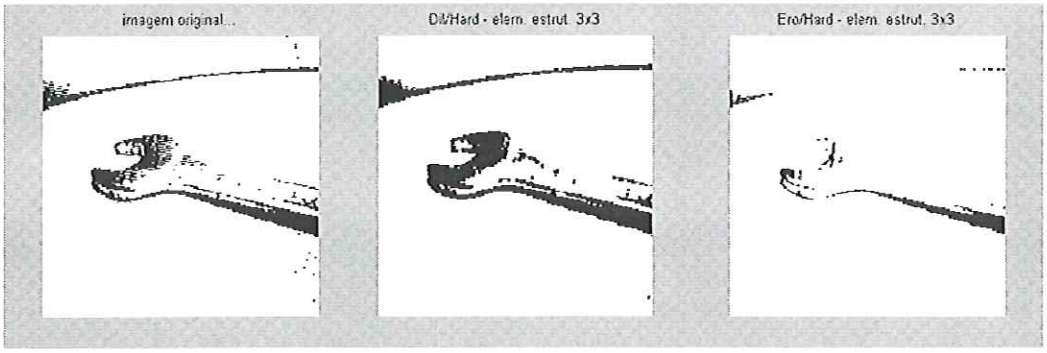


Figura 6.14a – Resultados obtidos pelo hardware para as operações de dilatação e de erosão da imagem original por um elemento estruturante quadrado de dimensão 3x3.

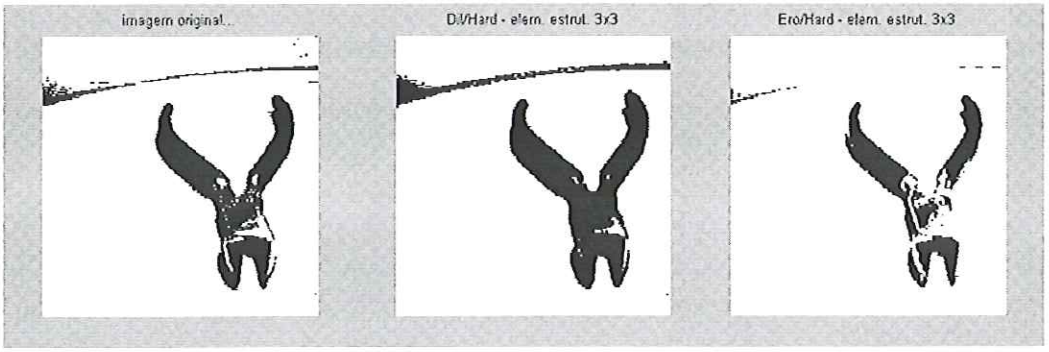


Figura 6.14b – Resultados obtidos pelo hardware para as operações de dilatação e de erosão da imagem original por um elemento estruturante quadrado de dimensão 3x3.

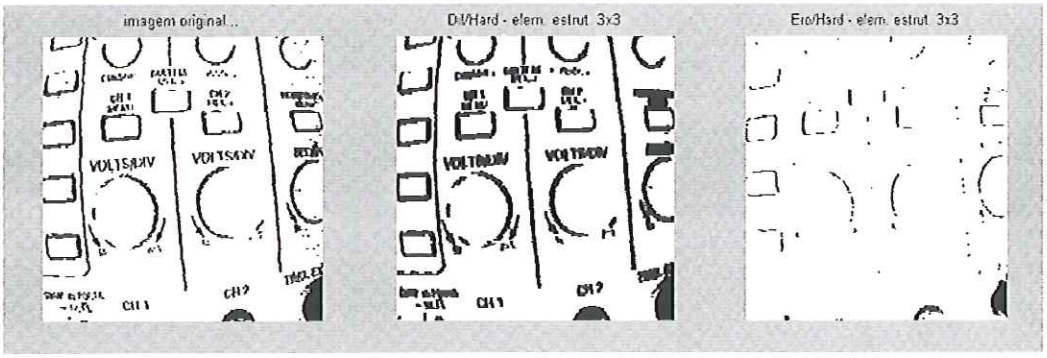


Figura 6.14c – Resultados obtidos pelo hardware para as operações de dilatação e de erosão da imagem original por um elemento estruturante quadrado de dimensão 3x3.

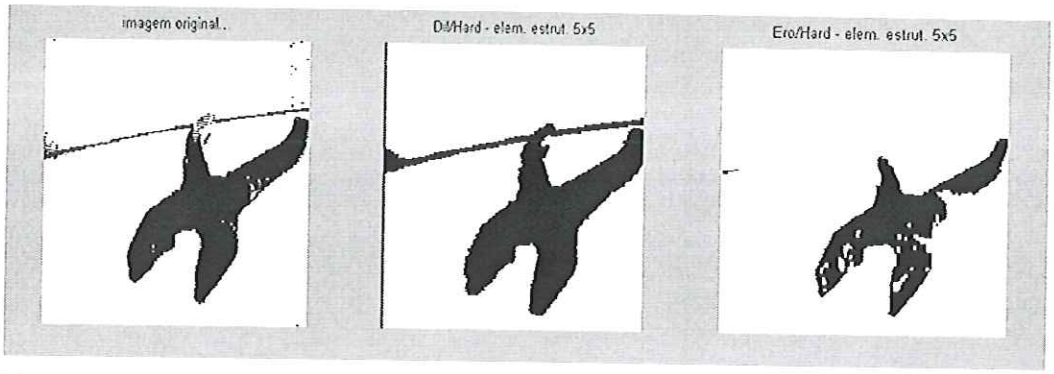


Figura 6.14d – Resultados obtidos pelo hardware para as operações de dilatação e de erosão da imagem original por um elemento estruturante quadrado de dimensão 5x5.

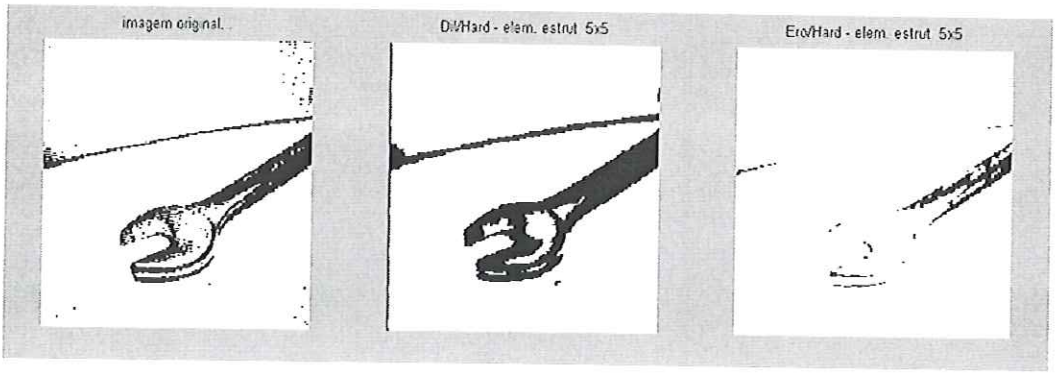


Figura 6.14e – Resultados obtidos pelo hardware para as operações de dilatação e de erosão da imagem original por um elemento estruturante quadrado de dimensão 5x5.

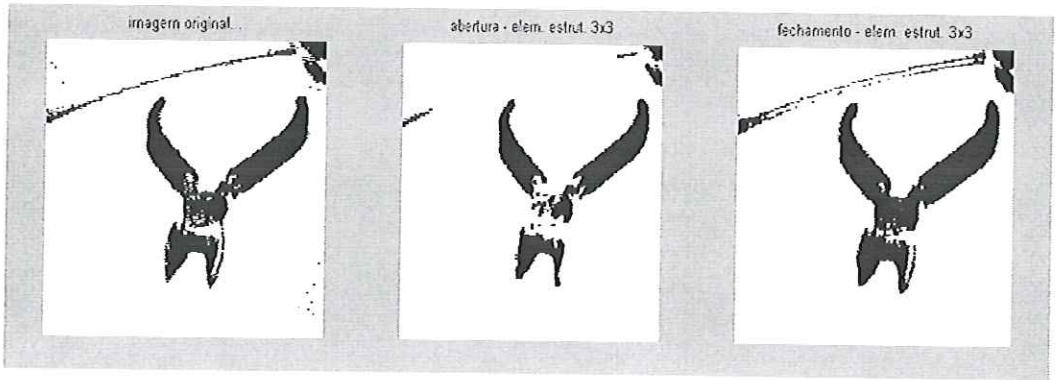


Figura 6.14f – Resultados obtidos pelo hardware para as operações de abertura e de fechamento da imagem original por um elemento estruturante quadrado de dimensão 3x3.

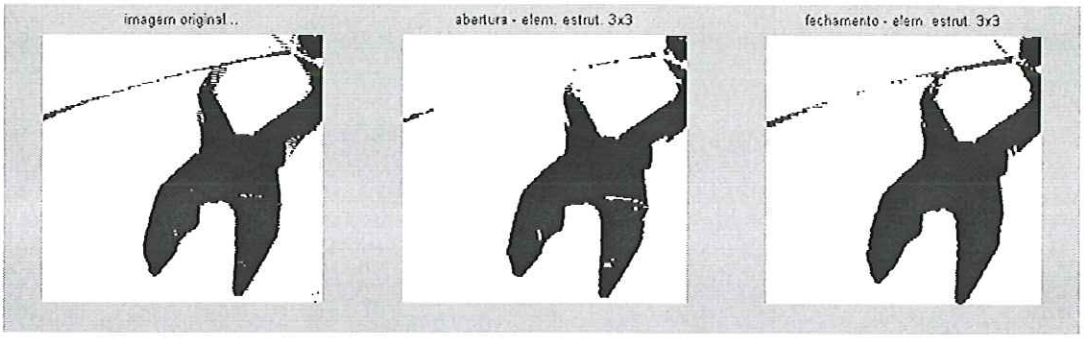


Figura 6.14g – Resultados obtidos pelo hardware para as operações de abertura e de fechamento da imagem original por um elemento estruturante quadrado de dimensão 3x3.

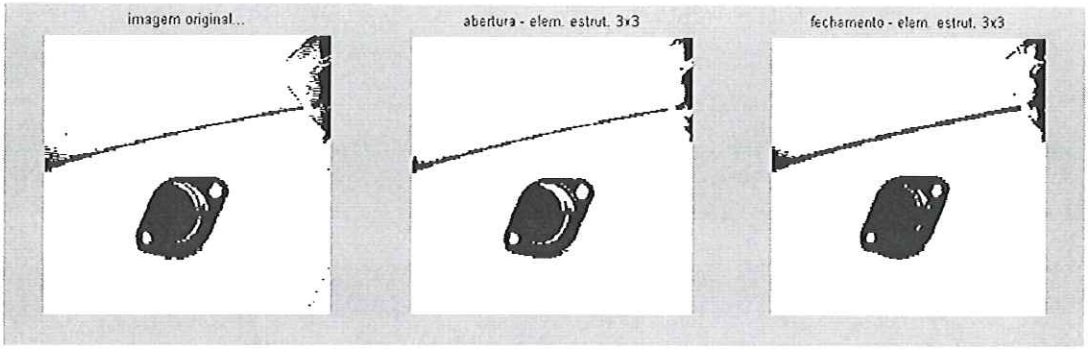


Figura 6.14h – Resultados obtidos pelo hardware para as operações de abertura e de fechamento da imagem original por um elemento estruturante quadrado de dimensão 3x3.

Na figura 6.15, apresentam-se os resultados obtidos de uma simulação (tempo x tamanho do elemento estruturante) por software da arquitetura *pipeline* desenvolvida em comparação com os resultados obtidos de uma implementação direta correspondente. Esta simulação foi realizada através do software Matlab 6.0 rodando em um Pentium 166 MHz com 64 MB de RAM.

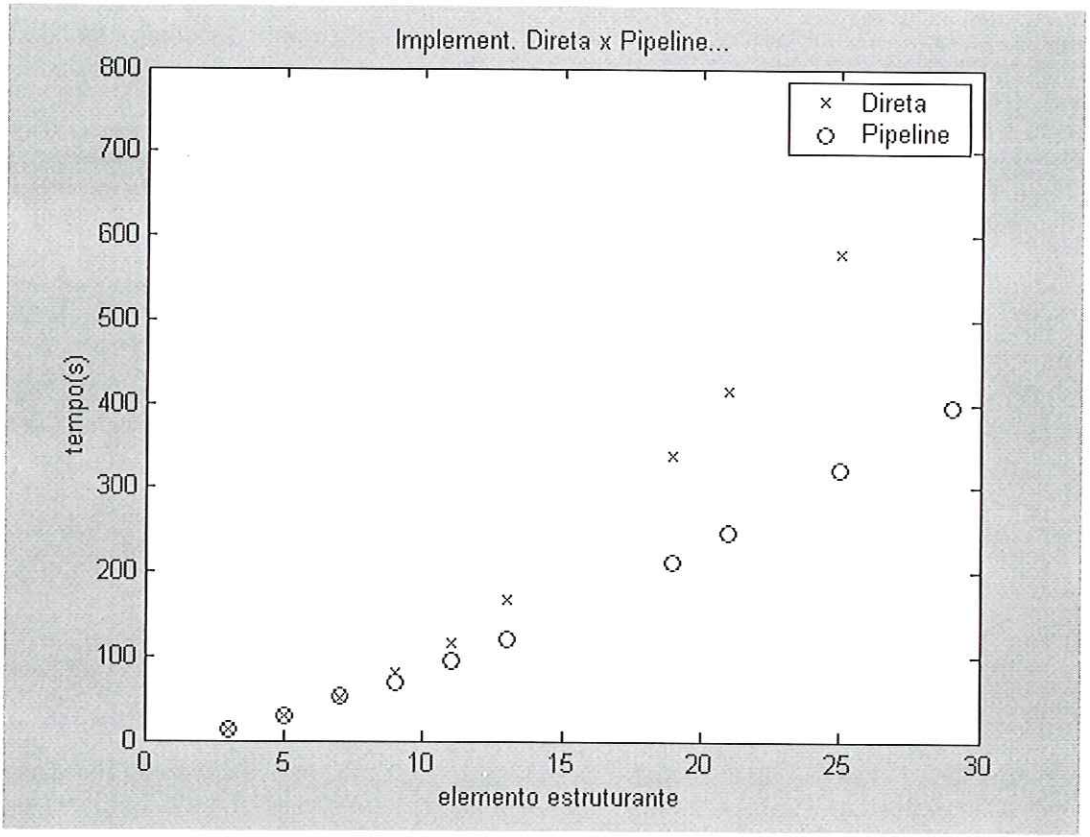


FIGURA 6.15 – Comparação temporal: implementação direta x implementação *pipeline* para diversos tamanhos de elementos estruturantes.

6.4 Considerações Finais

Neste projeto, os níveis de tensão utilizados foram compatíveis com o padrão TTL. As impedâncias de entrada e de saída do sinal de vídeo composto são de 75 ohms. O clock de 10 MHz do circuito é obtido pela divisão por 2 da frequência fornecida por um módulo oscilador de cristal de 20 MHz. A unidade de controle, bem como os estágios de processamento *pipeline*, podem ser reprogramados no próprio circuito através de interfaces JTAG. A implementação da unidade de processamento morfológica é flexível, considerando-se o uso de dispositivos lógicos programáveis de alta capacidade. Para se obter os resultados ilustrados nas figuras 6.14 foi necessário inverter os sinais de entrada e de saída em cada estágio da arquitetura *pipeline*.

7 CONCLUSÕES

7.1 Considerações Iniciais

A morfologia matemática é uma área de processamento de imagens e visão de máquina que está crescendo muito rapidamente nos últimos anos e possui aplicações em diversas áreas, tais como: visão de máquina, medicina, inspeção visual, análises de textura e de cena, entre outras. Muitas destas aplicações requerem processamento em tempo real, que pode somente ser obtido através de arquiteturas de propósitos especiais e estruturas de hardware especializadas. Também, de acordo com (ABBOTT et al., 1988; CHAUDHURI e CHEUNG, 1997 e SVOLOS et al., 2000), muitas arquiteturas dedicadas foram propostas para executar operações morfológicas em hardware com o objetivo de se obter um desempenho em tempo real, entretanto, impondo diversas restrições em relação ao tamanho e forma do elemento estruturante. As principais classes de arquiteturas em uso são as de arranjos de processadores paralelos e *pipeline*, sendo esta última mais adequada para tais aplicações. (BROGGI, 1994) mostra para uma arquitetura serial de propósito geral que a complexidade das operações morfológicas seriais depende do número de elementos que formam os operandos morfológicos, assim, fazendo-se a decomposição do elemento estruturante, o número de operações matemáticas envolvidas neste tipo de arquitetura pode ser reduzido. Portanto, levando-se em consideração tais princípios, foram desenvolvidos neste trabalho de pesquisa o projeto e a implementação de uma arquitetura *pipeline* dedicada para processamento morfológico de imagens binárias em tempo real utilizando dispositivos de lógica programável de alta capacidade. Assim, através da utilização destes dispositivos, o desenvolvimento, a análise e a implementação do projeto em hardware podem ser acelerados e os custos podem ser reduzidos. Também, a flexibilidade de implementações quanto à forma e tamanho do elemento estruturante é aumentada através desta abordagem.

7.2 Análise dos Resultados

Os resultados obtidos foram apresentados no capítulo anterior. A arquitetura desenvolvida apresentou um bom desempenho. Logo, o projeto da arquitetura em hardware dedicado para processar as operações básicas de morfologia matemática demonstrou ser uma alternativa viável e eficiente. Também, o gráfico apresentado na figura 6.15 mostra que para a arquitetura *pipeline* desenvolvida o tempo varia linearmente em relação ao tamanho do elemento estruturante em uma implementação por software, que não é o caso para uma implementação direta das operações básicas de morfologia matemática. Para se economizar hardware, a taxa de processamento utilizada neste projeto foi de 15 quadros/segundo, logo, foi necessário realizar um processo de leitura dupla na saída para se obter os 30 quadros/segundo necessários para se eliminar problemas de ofuscamento. Isto poderia ser evitado com o adicionamento de mais memória ao projeto. Na figura 7.1 é possível ver o diagrama de tempos dos módulos de memória da arquitetura desenvolvida. Antes das informações serem apresentadas à taxa de 30 quadros por segundo na saída, há um atraso inicial de 2 quadros (1/15 segundo). Também, através da figura é possível notar a concorrência existente no processamento dos módulos. A latência de cada módulo é de 1/30 segundo.

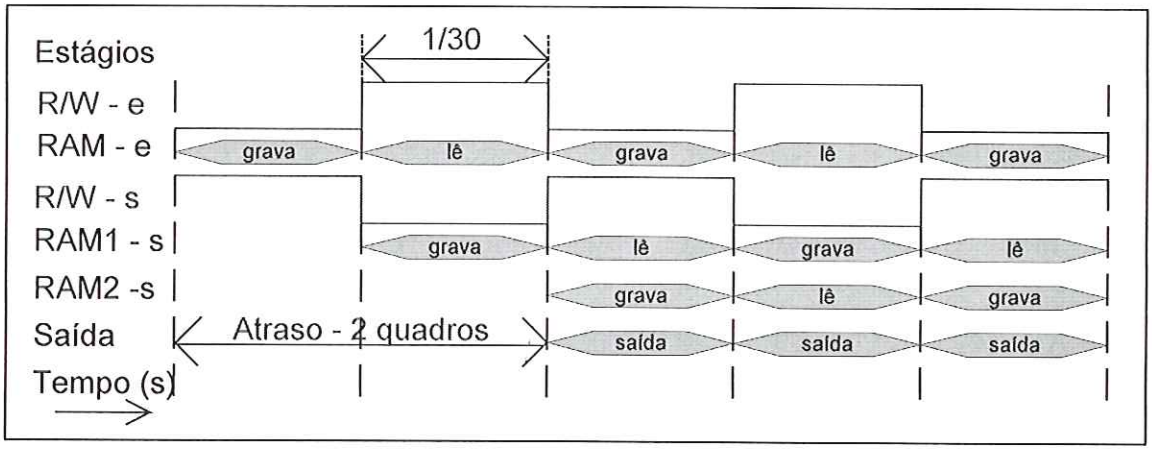


Figura 7.1 – Diagrama de tempos dos módulos de memória da arquitetura implementada.

7.3 Contribuições

Com a arquitetura desenvolvida neste trabalho é possível realizar em tempo real operações mais complexas e outros algoritmos em processamento de imagens e visão de máquina, considerando-se que os estágios podem ser reprogramados e também que o número de estágios pode ser alterado. Além desta arquitetura, desenvolveu-se um sistema de aquisição e armazenamento de imagens monocromáticas que poderá servir de modelo a outras aplicações e implementações de arquiteturas em processamento de imagens utilizando dispositivos lógicos programáveis de alta capacidade. Este projeto foi desenvolvido com ferramentas modernas utilizando descrições em esquemático e linguagens de descrição de hardware, possibilitando-se assim o reaproveitamento de algumas partes em outras aplicações. A linguagem HDL utilizada neste projeto foi a AHDL da Altera.

7.4 Sugestões para Trabalhos Futuros

Em relação ao projeto desenvolvido, poderiam ser feitas algumas melhorias. Entre elas, poderia se substituir o processo de limiarização simples adotado por um algoritmo de segmentação adaptativa para converter as imagens de níveis de cinza em imagens binárias. Este algoritmo foi implementado no software Matlab 6.0 utilizando o método de Bernsen, demonstrando-se assim, ser uma abordagem mais eficiente em relação às imagens binárias obtidas. Entretanto, a complexidade do hardware utilizado seria aumentada. Também, os vários dispositivos lógicos programáveis utilizados neste trabalho poderiam ser substituídos por um único dispositivo de alta densidade. Dessa forma, a implementação seria mais compacta, e além disso, possibilitaria a implementação de elementos estruturantes diversos em relação ao tamanho e a forma.

Referências

ABBOTT, A. et al. (1988). *Pipeline Architectures for Morphologic Image Analysis.* Machine Vision and Applications, v.1, n.1, p. 23-40.

ALI, K. S. (1996). *Digital Circuit Design Using FPGAs.* Computers ind. Engng, v. 31, n.1, p. 127-129, October.

ALTERA, Corp. (1998). *Advantages of ISP-Based PLDs over Tradicional PLDs.* Product Information Bulletin, n. 24.

ALTERA, Corp. (2002). *CPLDs x FPGAs, Comparing High-Capacity Programmable Logic.* Product Information Bulletin, n.18.

ALTERA, Corp. (2003). *Stratix Device Questions & Answers.* http://www.altera.com/products/devices/stratix/utilities/stx-q_a.html (jul. 2003).

ANDREADIS, I. et al. (1996). *An ASIC for fast grey-scale dilation.* Microprocessors and Microsystems v.20, p. 89-95.

BARROS, M. A. (1996). *Uma Metodologia de Projeto e Implementação de Operadores para Processamento Digital de Imagens em Tempo Real usando Field Programmable Gate Arrays (FPGA).* Anais do IX SIBGRAPI, p. 297-304.

BLOCH, I.; MAITRE, H. (1995). *Fuzzy Mathematical Morphologies: A comparative Study.* Pattern Recognition, v.28, p. 1341-1387.

BOURIDANE, A. et al. (1999). *A high level FPGA-based abstract machine for image processing.* Journal of System Architecture, v.45, p. 809-824.

BROGGI, A. (1994). *Speeding-up Mathematical Morphology Computations with Special-Purpose Array Processors*. Proccedings of the Twenty-Seventh Annual Hawaii International Conference on Systems Sciences.

BROWN, S. (1996). *FPGA Architectural Research: A Survey*. IEEE Design & Test Of Computers, v.13, n.4, p. 9-15.

BROWN, S.; ROSE, J. (1996). *FPGA and CPLD Architectures: A Tutorial*. IEEE Design & Test Of Computers, v.13, n.2, p. 42-57.

CHAN, P. K.; MOURAD, S. (1994). *Digital Design Using Field Programmable Gate Arrays*. Prentice Hall.

CHAUDHURI, A. S.; CHEUNG, P. Y. K. (1997). *A Reconfigurable Data-Localised Array for Morphological Algorithms*. Field-Programmable Logic and Applications (LNCS 1304, Springer), p. 344-353.

CHEN, K. (1989). *Bit-Serial Realizations of a Class of Nonlinear Filters Based on Positive Boolean Functions*. IEEE Transactions On Circuits And Systems, v.36, n.6, p. 785-794, June.

CHOW, P. et al. (1999a). *The Design of an SRAM-Based Field-Programmable Gate Array – Part I: Architecture*. IEEE Transactions On Very Large Scale Integration (VLSI) Systems, v.7, n.2, June.

CONEXANT (1999). *20 MSPS Monolithic CMOS 8-bit Flash Video A/D Converter (Datasheet)*.

COSTA, L. F.; CESAR Jr., R. M. (2001). *Shape Analysis and Classification: Theory and Practice*. CRC Press.

DIAMANTARAS, I.; KUNG, S. Y. (1997). *A Linear Systolic Array for Real-Time Morphological Image Processing.* Journal of VLSI Signal Processing, v.17, p. 43-55.

DICK, C.; HARRIS, F. (1998). *Virtual signal processors*. Microprocessors and Microsystems, v.22, p. 135-148.

DOUGHERTY, E. R. (1992). *An Introduction to Morphological Image Processing*. Washington, SPIE.

ELANTEC (1993). *EL4581C Sync Separator, 50% Slice, S-H, Filter (Datasheet)*.

FACON, J. (1996). *Morfologia Matemática: Teoria e Exemplos*. Curitiba, Editora Universitária Champagnat da Pontífica Universidade Católica do Paraná.

FITCH, J. P. et al. (1985). *Threshold Decomposition of Multidimensional Ranked Order Operations.* IEEE Transactions On Circuits And Systems, v.32, p. 445-450.

GASTERATOS, A. (2002). *Specialized Hardware Structures for Morphological Image Processing.* http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/GASTERATOS (Dez. 2002).

GASTERATOS, A. et al. (1996). *Improvement of the Majority Gate Algorithm for Grey-Scale Dilation/Erosion.* Electronics Letters, v.32, p. 806-807.

GERRITSEN, F. A.; VERBEEK, P. W. (1984). *Implementation of Cellular-Logic Operators Using 3*3 Convolution and Table Lookup Hardware*. Computer Vision, Graphics, and Image Processing, v. 27, p. 115-123, August.

GONZALEZ, R. C.; WOODS, R. E. (1992). *Digital Image Processing*. 3.ed. Addison-Wesley.

GONZÁLEZ, F. et al. (2002). *Morphological processor for real-time image applications.* Microelectronics Journal, v.33, p. 1115-1122, July.

GREENE, J. et al. (1993). *Antifuse Field Programmable Gate Arrays*. Proceedings Of The IEEE, v.81, n.7, July.

GUŠTIN, V. (1998). *An FPGA extension to ALU functions*. Microprocessors and Microsystems, v.22, p. 501-508, December.

HARALICK, R. M. et al. (1987). *Image Analysis Using Mathematical Morphology*. IEEE Transactions On Pattern Analysis And Machine Intelligence, vol.9, n.4, p.532-550, July.

HAUCK, S. (1998). *The Roles of FPGA's in Reprogrammable Systems*. Proceedings Of The IEEE, v.86, n.4, p. 615-638, April.

HOROWITZ, P.; HILL, W. (1989). *The Art Of Electronics*. 2.ed., Cambridge Univerity Press.

IMAGING RESEARCH INC. (1998). *Evaluating Image Analysis Systems.*

INTERSIL (1997). *CA3338, CA3338A, CMOS Video Speed, 8-Bit,50 MSPS, R2R D/A Converters (Datasheet).*

KATZ, R. H. (1993). *Contemporary Logic Design*. University of California, Benjamin Cummings/Addison Wesley Publishing Company.

KLEIN, J. C.; SERRA, J. (1972). *The texture analyser*. Journal of Microscopy, v. 95, p. 349-356, April.

KNAPP, S. (2000). *Frequently-Asked Questions (FAQ) About Programmable Logic*. http://www.optimagic.com/faq.html (28 Maio 2002).

KOJIMA, S. et al. (1994). *Fast Morphology Hardware Using Large-Sized Structuring Element.* Systems and Computers in Japan, v.25, n.6, p. 41-49.

KOSKINEN, L. et al. (1991). *Soft Morphological Filters.* Proceedings of SPIE Symposium on Image Algebra and Morphological Image Processing, p. 262-270.

KUHN, K. J. (1996). *Conventional Analog Television – An Introduction.* http://www.ee.washington.edu/conselec/CE/Kuhn/ntsc/95X4.htm (24 Jul. 2002).

LANDIS, D. (1996). *Handbook of Components for Electronics*. 2.ed., H. Jones and C. Harper editors, McGraw-Hill, Inc.

LUCK, L.; CHAKRABATY, C. (1995). *A Digit-Serial Architecture For Gray-Scale Morphological Filtering.* IEEE Transactions On Image Processing, v.4, p. 387-391.

MARQUES FILHO, O.; VIEIRA NETO, H. (1999). *Processamento Digital de Imagens.* Rio de Janeiro, Brasport.

MICHAEL, N.; ARRATHOON, R. (1997). *Optoelectronic pipeline architecture for morphological image processing.* Applied Optics, v.36, n.8, p. 1718-1725, March.

NINCE, U. S. (1991). *Sistemas de Televisão e Vídeo*. 2. ed. Rio de Janeiro, LTC.

PRESTON Jr., K. (1983). *Cellular Logic Computers for Pattern Recognition.* Computer, v.16, p. 36-47, January.

ROSE, J. et al. (1993). *Architecture of Field-Programmable Gate Arrays.* Proceedings Of The IEEE, v.81, n.7, p. 1013-1029, July.

SALCIC, Z.; SMAILAGIC, A. (1998). *Digital Systems Design and Prototyping Using Field Programmable Logic.* Boston, Kluwer Academic Publishers.

SCHURE, A. (1964). *Televisão Básica*. Rio de Janeiro, Livraria Freitas Bastos.

SENATORI, N. O. B.; SUKYS, F. (1987). *Introdução à Televisão e ao Sistema PAL-M*. Rio de Janeiro, Guanabara.

SERRA, J. (1982). *Image Analysis and Mathematical Morphology*. Califórnia, Academic Press Inc.

SERRA, J. (1986). *Introduction to Mathematical Morphology*. Computer Vision, Graphics, and Image Processing, 35, p.283-305, March.

SHIH, F. Y. C.; MITCHELL, O. R. (1989). *Threshold Decomposition of Gray-Scale Morphology into Binary Morphology*. IEEE Transactions On Pattern Analysis And Machine Intelligence, v.11, n.1, p. 31-42, January.

SISKOS, S. et al. (1998). *Analog Implementation of Fast Min/Max Filtering*. IEEE Transactions On Circuits And Systems, v.45, n.7, p. 913-918, July.

SOLDEK, J.; MANTIUK, R. (1999). *A reconfigurable processor based on FPGAs for pattern recognition, processing, analysis and synthesis of images*. Pattern Recognition Letters, v.20, p. 667-674.

SONKA, M.; HLAVAC, V.; BOYLE, R. (1993). *Image Processing, Analysis and Machine Vision*. Chapman & Hall Computing.

STERNBERG, S. R. (1983). *Biomedical Image Processing*. Computer, vol.16, p. 22-34.

STOREY, N.; STAUTON, R. C. (1989). *An Adaptive Pipeline Processor For Real-Time Image Processing*. Proc. SPIE, v.1197, p. 238-246.

SVOLOS, A. I. et al. (2000). *Efficient Binary Morphological Algorithms on a Massively Parallel Processor*. Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00).

TEKALP, A. M. (1995). *Digital Video Processing*. Prentice-Hall.

TEXAS INSTRUMENTS, (2001). *ADS830 SpeedPLUS 8 bit, 60MHz Sampling ANALOG-TO-DIGITAL CONVERTER (Datasheet).*

THOMAS, F. et al. (1999). *A hardware/software codesign for improved data acquisition in a processor based embedded system*. Microprocessors and Microsystems, v.24, p. 129-134, September.

TOSHIBA (1997). *TC551001 CP/CF/CFT/CTR/CSR –55, -70, -85, -55L, -70L, -85L (Datasheet).*

TURLEY, J. (1997). *Soft computing reconfigures designer options*. Computer Design, p. 76-82, April.

VASSÁNYI, I. (1997). *FPGAs and cellular algorithms: Two implementation examples.* Journal of System Architecture, v.43, p. 23-26.

VILLASENOR, J.; SMITH, W. H. M. (1997). *Configurable Computing*. Scientific American. http://www.sciam.com/0697issue/0697villasenor.html (28 Maio 2002).

WEEKS Jr., A. R. (1996). *Fundamentals of Electronic Image Processing*. Washington, SPIE.

ZHANG, X.; NG, K. W. (2000). *A review of high-level synthesis for dynamically reconfigurable FPGAs*. Microprocessors and Microsystems, v.24, p. 199-211, April.

Bibliografia Complementar

A Closer Look At Eproms. http://www.phreaker.org/text/newtext/005.txt (28 Maio 2002).

BRAIN, M. (2002). *How Television Works*. http://www.howstuffworks.com/tv.htm (09 Jan. 2002).

CHEUNG, M. (1995). *Field Programmable Gate Arrays*. http://www.ele.auckland.ac.nz/students/chengms/fpga.htm (28 Maio 2002).

CHOW, P. et al. (1999b). *The Design of a SRAM-Based Field-Programmable Gate Array – Part II: Circuit Design and Layout*. IEEE Transactions On Very Large Scale Integration (VLSI) Systems, v.7, n.3, September.

DATAPRO (2000). *Video Signal Standards*. http://www.datapro.net/videodoc.html (13 Ago. 2001).

ENGDAHL, T. *RS-170 Video Signal*. http://www.epanorama.net/documents/video/rs170.html (13 Ago. 2001).

FAIRCHILD SEMICONDUCTOR CORPORATION (1998). *Using Existing Programmers to Program Low Voltage EPROMs*. Fairchild Application Note, n.825.

GALBIATI Jr., L. J. (1990). *Machine Vision and Digital Image Processing Fundamentals*. Prentice-Hall.

HODGES, D. A.; JACKSON, H. G. (1988). *Analysis and Design of Digital Integrated Circuits*. 2.ed. Berkeley. McGraw-Hill.

JENKINS, J. H. (1994). *Designing With FPGAs and CPLDs*. Prentice Hall.

KAPUSTA, R. (1997). *Navigating the Programmable Logic Landscape*. Microelectronics Journal, v.28, n.5, p. viii-xviii, June.

LUPPE, M. (1997). *Desenvolvimento de um Processador Dedicado para Extração de Bordas em Tempo Real.* 90p. Dissertação (Mestrado) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos. 1997.

ROCKWELL (1998). Application Note. *Comparison of NTSC, PAL, and SECAM Video Formats.*

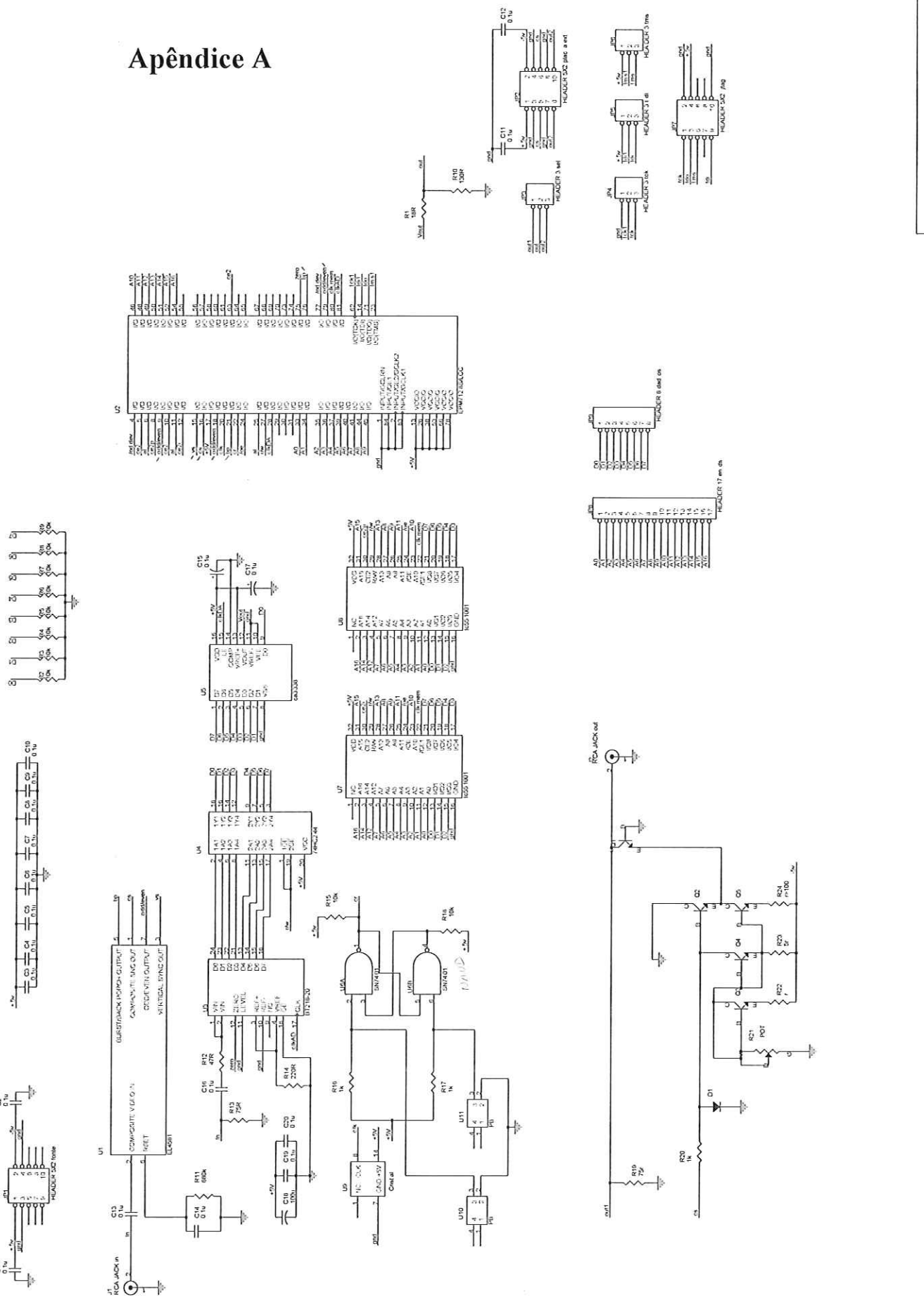
VINCENTELLI, A. S. et al. (1993). *Synthesis Methods for Field Programmable Gate Arrays*. Proceedings Of The IEEE, vol.81, n.7, p. 1057-1083, July.

VIRTUAL COMPUTER CORPORATION. *An Introduction for the Unfamiliar – Simply Speaking – The difference between Fixed & Reconfigurable Hardware.* http://www.vcc.com/intro1.html (28 Maio 2002).

VIRTUAL COMPUTER CORPORATION. *Field Programmable Gate Arrays (FPGAs). An Enabling Technology.* http://www.vcc.com/fpga.html (28 Maio 2002).

ZENG, S. (2000). *The Architecture of a Moletronics Computer.* http://www.egr.msu.edu/~zengshuq/cse820_paper4.html (28 Maio 2002).

Apêndice A



Apêndice B

```
% Contador de pulsos serrilhados e equalizadores %
% Autor: Emerson Carlos Pedrino %
% Fev / 2002 %

SUBDESIGN contpulsos
(
        clk, clr, enb, odev: INPUT;
        out[4..0], max: OUTPUT;
)
VARIABLE
        count[4..0]: DFF;
BEGIN
        count[].clk=clk;
        count[].clrn=clr;
        IF enb THEN
                count[].d=count[].q+1;
                IF (count[].q==H"10") & (odev==1) THEN
                        max=B"1";
                ELSIF
                        (count[].q==H"f") & (odev==0)  THEN
                        max=B"1";
                ELSE
                        max=B"0";
                END IF;
        ELSE
                count[].d=count[].q;
        END IF;
        out[]=count[];
END;
```

Apêndice C

```
% Controle de exclusao de pulsos serrilhados e equalizadores %
% Autor: Emerson Carlos Pedrino %
% Fev. 2002 %

SUBDESIGN controlexclusao
(
        maxin, clk, reset: INPUT;
        start: OUTPUT;
)
VARIABLE
        estado: MACHINE OF BITS (q1,q0)
                        WITH STATES (
                           ie,  % inicio exclusao %
                           ex, % exclusao %
                           fe  % fim exclusao %
                                   );
BEGIN
        estado.clk=clk;
        estado.reset=reset;
        TABLE
              estado, maxin => estado, start;
                 ie, 0 => ex, 0;
                 ex, 0 => ex, 1;
                 ex, 1 => fe, 0;
         END TABLE;
END;
```

Apêndice D

```
% Unidade de controle  -  Autor: Emerson Carlos Pedrino - Fev / 2002 %
SUBDESIGN unidadecontr
(
   % start -> sinal de controle da unidade de controle interna %
   % cs -> composite sync %
   % alinit -> inicio do periodo de linha ativa %
   % cr -> controle de leitura/escrita %
   % rw -> sinal de leitura/escrita %
   start, cs, clk, reset, cr: INPUT;
   alinit, rw: OUTPUT;
)
VARIABLE
        estado: MACHINE OF BITS (q1, q0) WITH STATES (
                                                vbinit, % inicio vertical %
vb, % vertical blanking %
                                                li % line init %
                                                        );
BEGIN
        estado.clk=clk;
        estado.reset=reset;
        IF cr THEN
                rw=B"1";
        ELSE
                rw=B"0";
        END IF;
        TABLE
                estado, start, cs => estado, alinit;
                vbinit, 0, X  => vbinit, 0;
                vbinit, 1, X  => vb, 0;
                vb, 0, X => li, 0;
                li, 0, 0 => li, 1;
                li, 0, 1 => li, 0;
        END TABLE;
END;
```

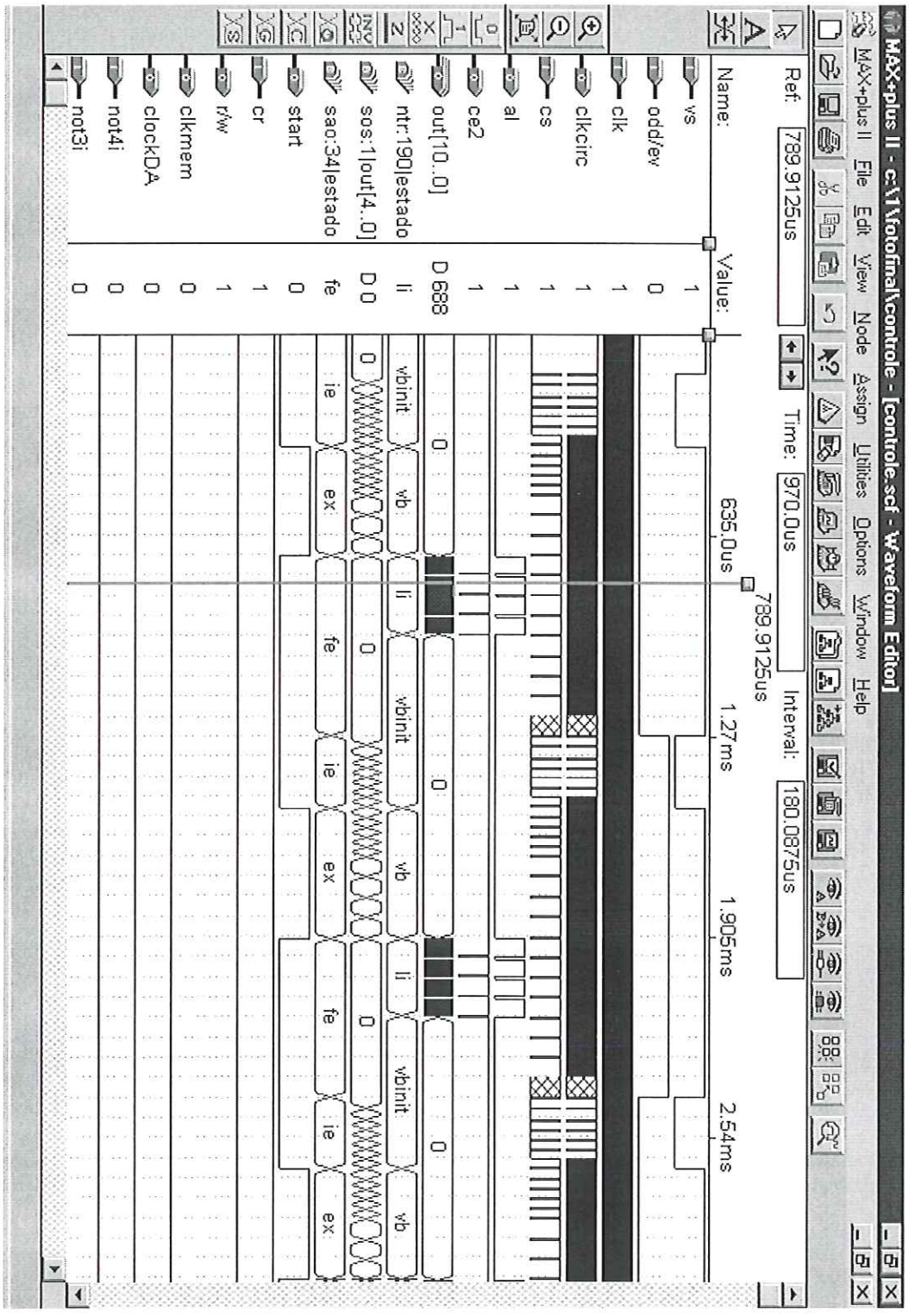
Apêndice E

```
% Contador de linhas de video %
% Autor: Emerson Carlos Pedrino %
% Fev / 2002 %

SUBDESIGN contlin
(
        clk, clr, enb, ep: INPUT;
        out[16..9], el: OUTPUT;
)
VARIABLE
        count[16..9]: DFF;
BEGIN
        count[].clk=clk;
        count[].clrn=clr;
        IF enb THEN
                count[].d=count[].q+1;
        ELSE
                count[].d=count[].q;
        END IF;
        out[]=count[];
        el=(out[]==H"ff")&(ep==1);
END;
```

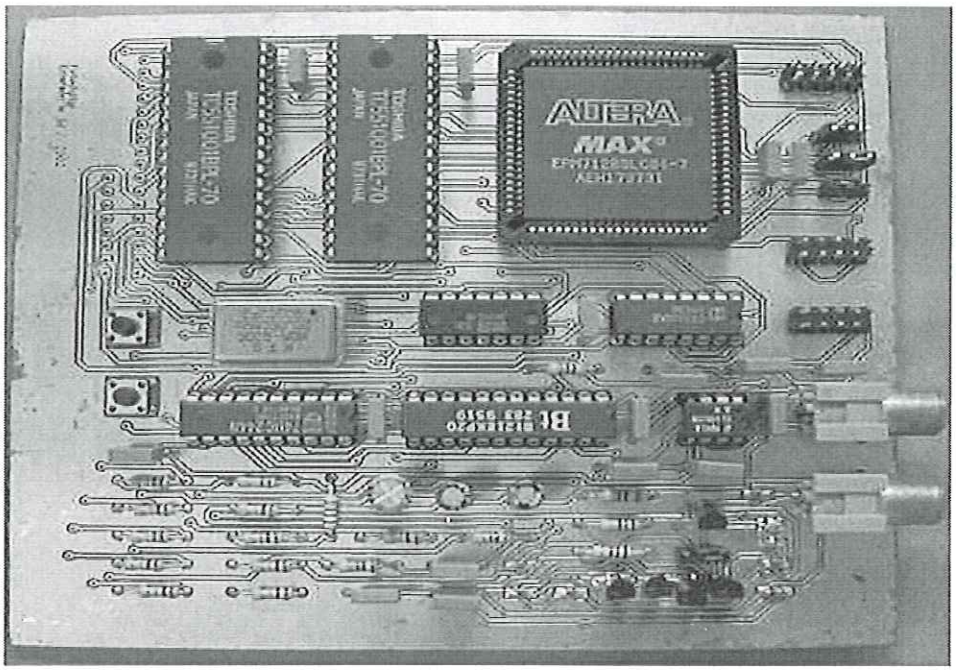
Apêndice F

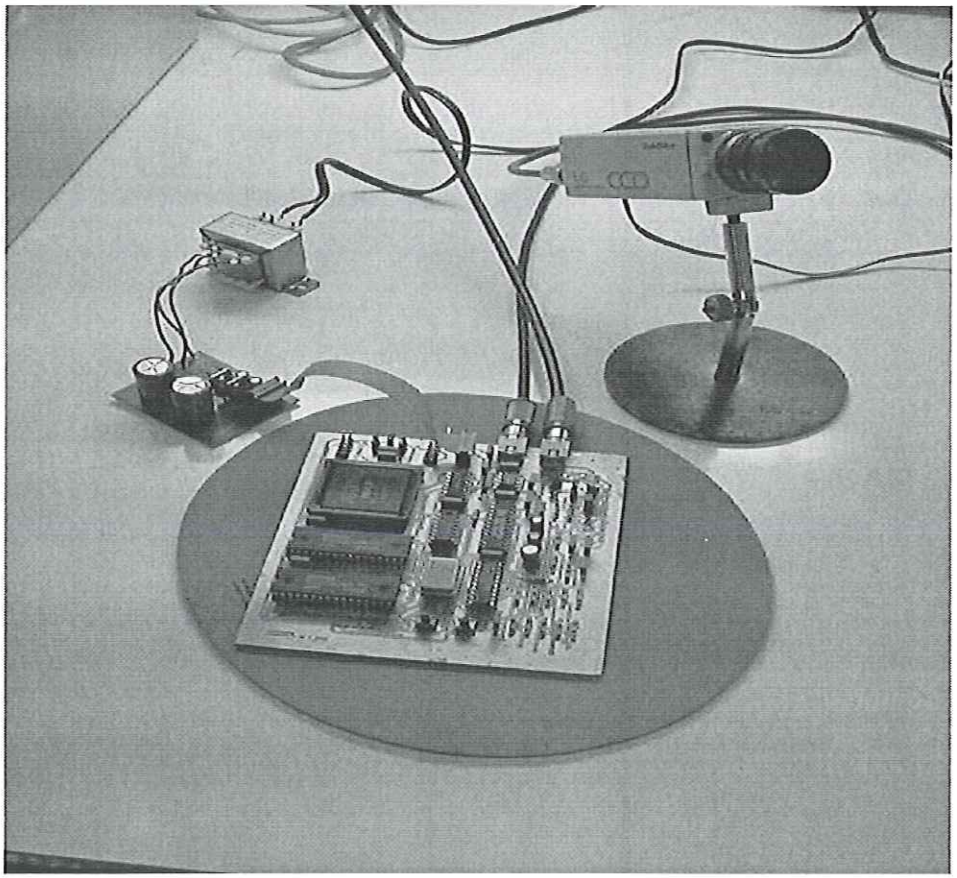
```
% Contador de pixels %
% Autor: Emerson Carlos Pedrino %
% Fev / 2002 %

SUBDESIGN contpix
(
        clk, clr, enb: INPUT;
        out[8..0], ep: OUTPUT;
)
VARIABLE
        count[8..0]: DFF;
BEGIN
        count[].clk=clk;
        count[].clrn=clr;
        IF enb THEN
                count[].d=count[].q+1;
        ELSE
                count[].d=count[].q;
        END IF;
        out[]=count[];
        ep=(out[]==H"1ff")%&(!clk)%;
END;
```

Apêndice G: Simulação da Unidade de Controle do Cap. 4.

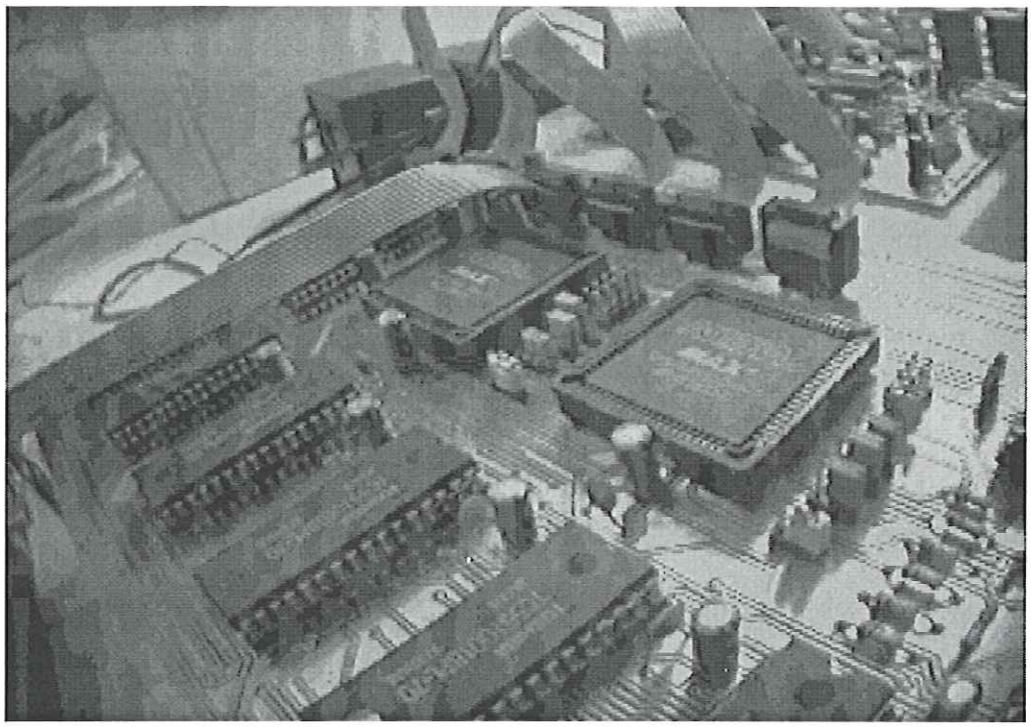


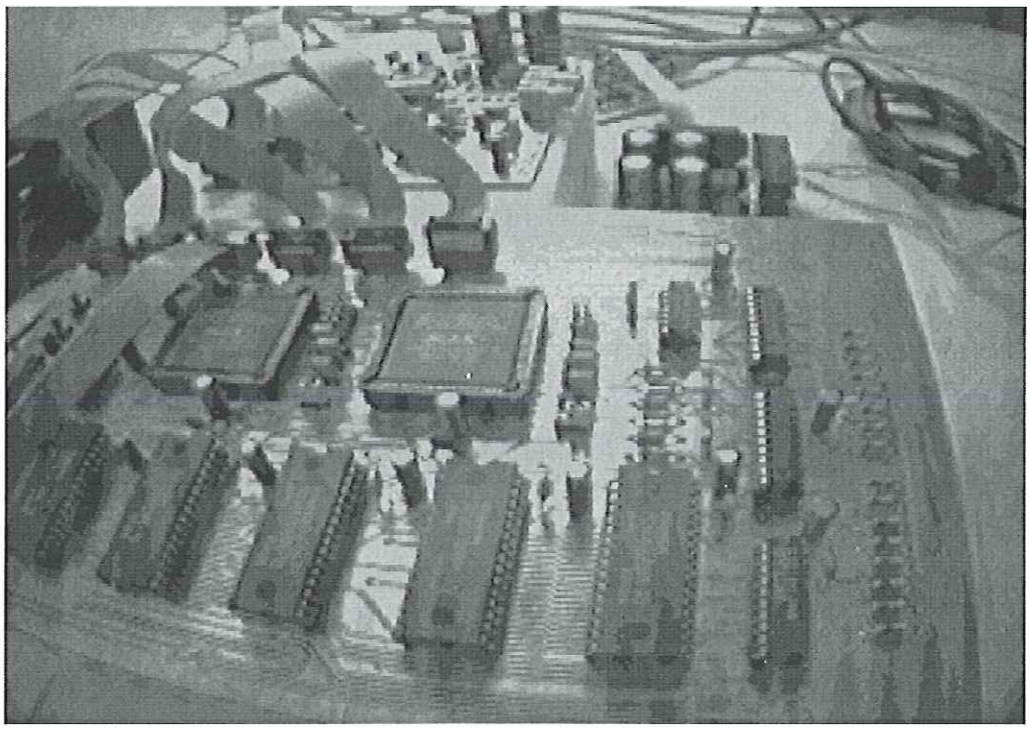
Apêndice H: Fotos do Sistema Desenvolvido no Capítulo 4.



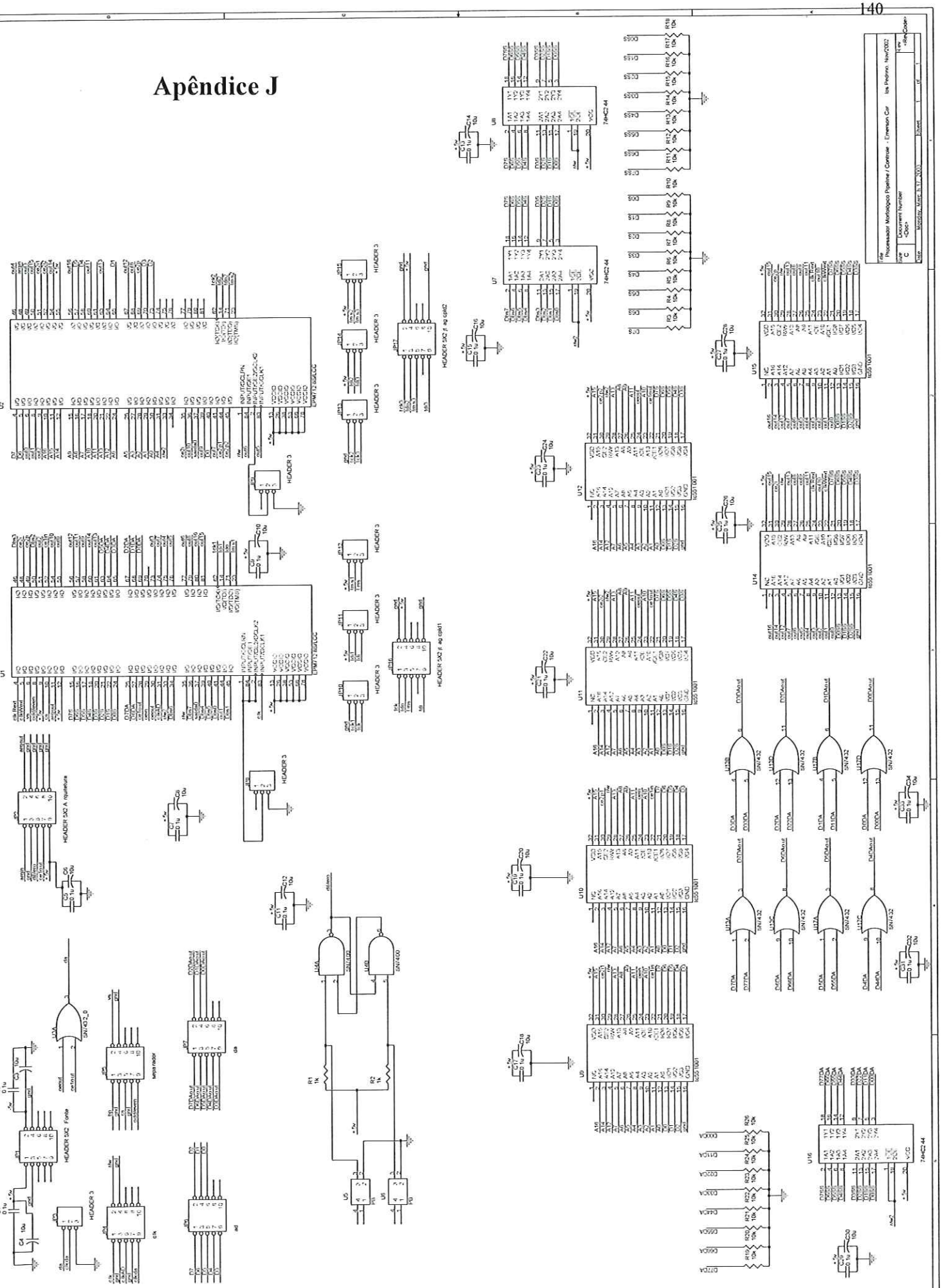


Apêndice I : Imagens Obtidas pelo Sistema do Cap. 4.

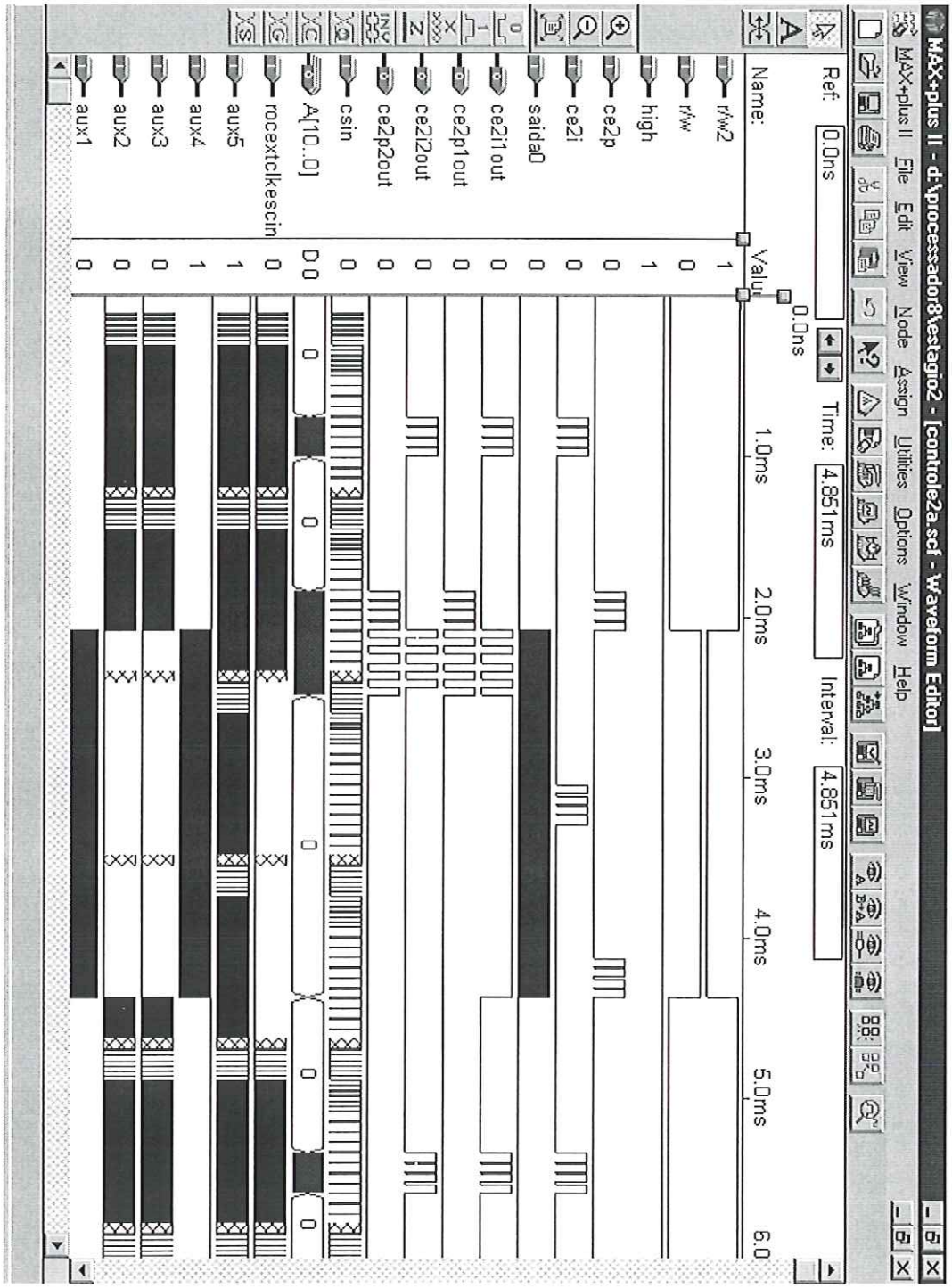




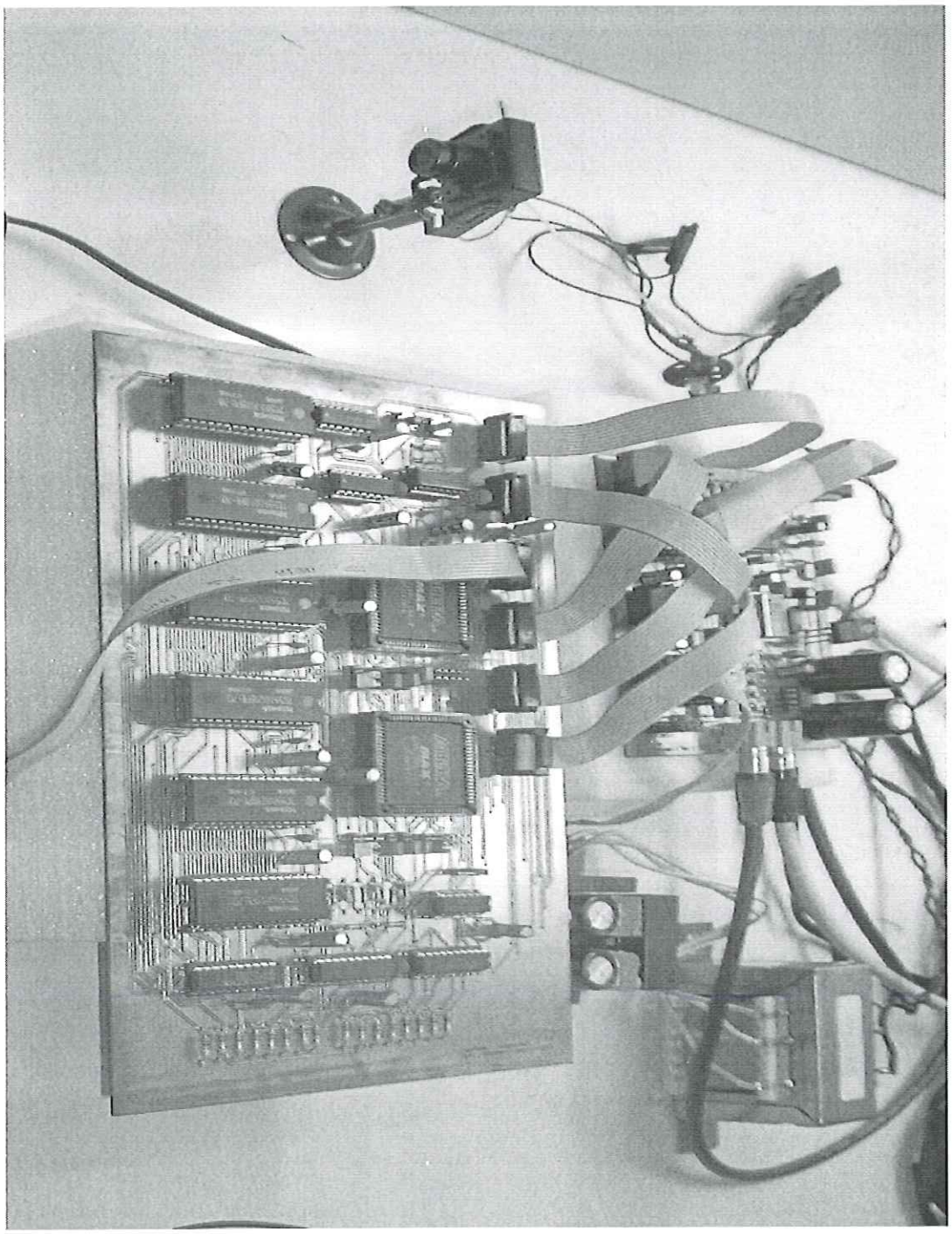
Apêndice J

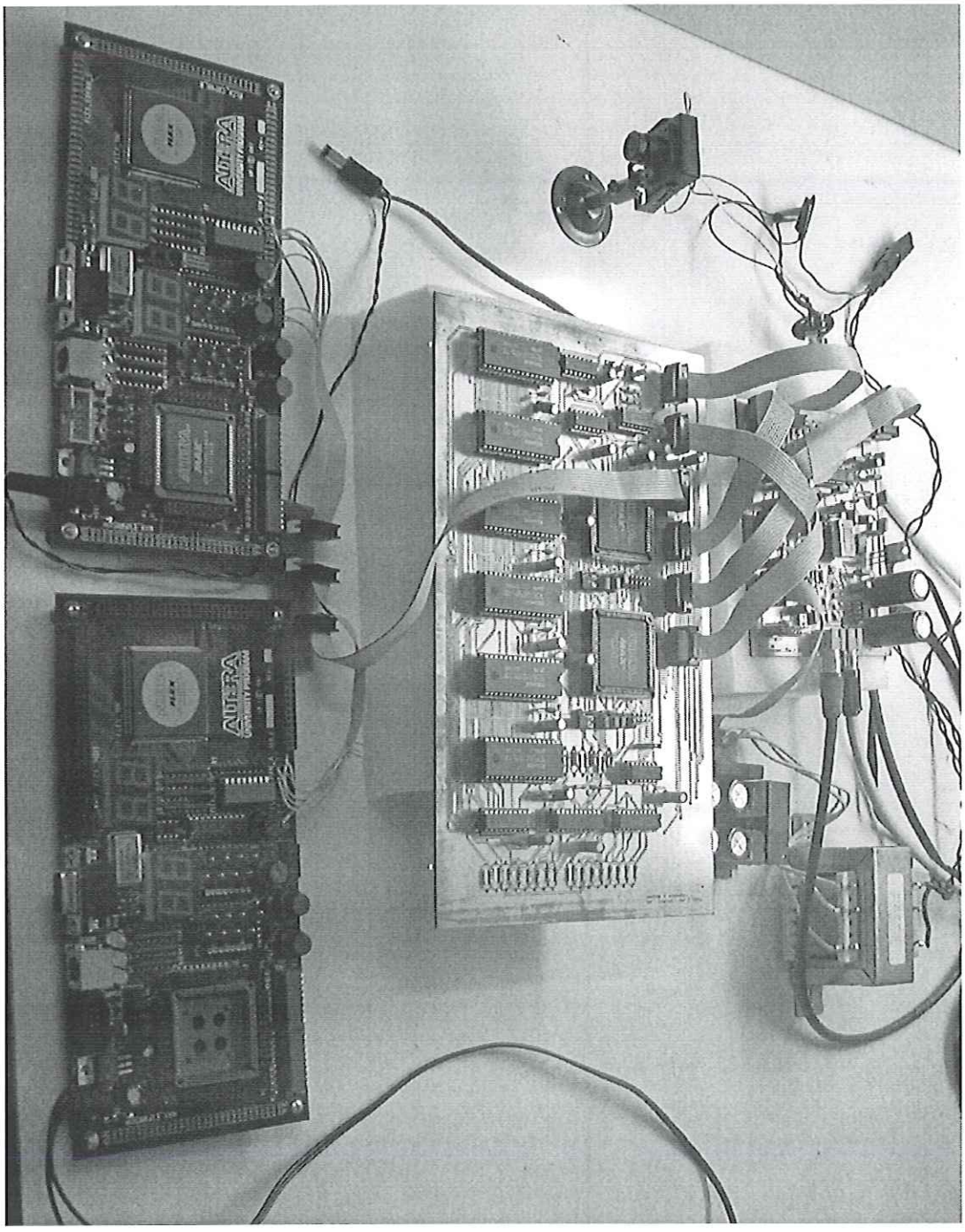


Apêndice K : Simulação da Arquitetura Desenvolvida.



Apêndice L : Fotos da Arquitetura Desenvolvida.




ALTERA
ALTERA