IMPLEMENTAÇÃO DE CONTROLADOR LÓGICO BASEADO EM LÓGICA PROGRAMÁVEL ESTRUTURADA (FPGA)

Cesar da Costa
Cost036@attglobal.net
UNITAU – Universidade de Taubaté
Departamento de Engenharia Mecânica
Pós-Graduação em Automação Industrial

Resumo

A proposta deste trabalho é desenvolver um controlador programável para uso geral que utilize na sua arquitetura lógica programável estruturada(FPGA) e, macro instruções para programação desse controlador, que será utilizado por uma empresa nacional, no ramo de instrumentação, como elemento de auxílio ao treinamento em CLP. O objetivo da utilização de lógica programável nessa proposta é integrar em um único dispositivo as funções do microcontrolador e dos circuitos integrados de aplicações específicas (ASICs) e, estudar a possibilidade de minimizar a realização dos ciclos de busca das instruções, visando a melhoria das dimensões do sistema, o rendimento de operação do controlador e a redução do consumo de energia.

Palavras Chaves: ASIC, FPGA, HDL, EDA, PLD, SPLD, CPLD, EPLD, EPROM, PLA, CLP, VHDL, Verilog, SRAM, ISP, LE, ASIC, LUT, LAB, síntese lógica, configuração, roteamento, ciclo de busca.

1 INTRODUÇÃO

Os controladores lógicos programáveis (CLPs) disponíveis no mercado brasileiro, tipicamente, utilizam na sua arquitetura microcontroladores e circuitos integrados de aplicações específicas (ASICs - *Application Specific Integrated Circuits).* Esses microcontroladores para executarem os seus programas de controle necessitam realizar ciclos de busca e execução da instrução. O ciclo de busca da instrução não está diretamente relacionado com o processo no qual o controlador lógico programável está inserido, mas é condição determinante para o microcontrolador executar o programa que está carregado na memória. Essa necessidade de busca da instrução demanda tempo do microprocessador o qual poderia estar sendo utilizado na execução das tarefas pertinentes ao processo.

Os microcontroladores são componentes extremamente flexíveis devido a sua programabilidade. A sua programação permite a sua aplicação em diversos tipos de controles industriais. A execução de um algoritmo depende de um software (memória), que será executado numa arquitetura tipo Von Neumann, por exemplo, com ciclos de busca e execução das instruções. Numa arquitetura baseada em lógica programável estruturada, por exemplo, FPGA(*Field Programmable Gate Array*) um algoritmo é implementado por hardware, sem a necessidade de ciclos de busca e execução de instruções. O problema básico a ser resolvido é a implementação de uma arquitetura eficiente, baseada em lógica programável estruturada, para execução desse algoritmo ao invés de compilá-lo para sua execução em uma CPU.

A tarefa que faz a tradução de um algoritmo para uma arquitetura de hardware eficiente é denominada Síntese. A Síntese cria uma arquitetura com células lógicas que executam as operações de algoritmos, sem a necessidade de se gerar e decodificar instruções.

Uma das grandes vantagens da utilização de FPGA's nessa nova arquitetura proposta é a possibilidade de se definir vários blocos de hardware, que operam em paralelo, aumentando muito a capacidade computacional do sistema [3]. Já o ambiente de desenvolvimento além de ter o tempo e o custo reduzido em relação aos ambientes tradicionais de projeto, permite simular e testar rapidamente em campo o protótipo ou a versão final deste, empregando-se novas metodologias de projeto de hardware apoiadas em poderosas ferramentas de software EDA *(Eletronic Design Automation)*.

A partir desse novo conceito surge a motivação para o projeto de um controlador lógico programável CLP baseado em lógica programável estruturada (FPGA), que seja dinamicamente reconfigurável e que possua estrutura adequada á implementação de algoritmos via hardware. Além disso, a nova poderá ser utilizada em outras aplicações que não sejam na área de automação industrial como aeronáutica, metroviária, etc [4].

Este trabalho está organizado da seguinte forma. A seção 2 apresenta o conceito e a estrutura básica dos dispositivos lógicos programáveis simples SPLDs e EPLDs abordados neste projeto. A seção 3 mostra a evolução dos dispositivos lógicos programáveis complexos HCPLDs, dando ênfase aos dispositivos CPLDs e FPGAs. Na seção 4 é apresentada a metodologia de projeto EDA bem como as técnicas de configuração e programação dos dispositivos lógicos programáveis, abordando as principais linguagens de descrição de hardware como HDL, VHDL, edição gráfica. A seção 5 trata da descrição dos CLPs tradicionais, arquitetura, ciclo de varredura e tempo de execução. A seção 6 apresenta a nova arquitetura proposta baseada em dispositivo lógico programável FPGA, definição do bloco lógico, flexibilidade de interconexão, ferramentas de auxílio à programação, simulação e testes. A seção 7 apresenta alguns resultados, conclusões finais quanto ao desempenho em relação a arquitetura tradicional de CLP e indicações de trabalhos futuros.

2 CONCEITO DE LÓGICA PROGRAMÁVEL ESTRUTURADA

2.1 Circuitos Integrados Digitais

Os circuitos integrados digitais implementados em pastilha de silício podem ser classificados como circuitos digitais padrões ou circuitos digitais de aplicações específicas *ASICs (Application specific integrated circuits)* [1] . Os circuitos padrões são constituídos por funções lógicas como *AND, OR, NAND, NOR, EXOR, Flip-Flops*, etc e, necessitam de vários componentes externos para a realização de uma função específica.Os circuitos *ASICs* também conhecidos como *customizados,* como por exemplo os microcontroladores, caracterizam-se por sua programabilidade e, não necessitam de muitos componentes externos para a realização de uma função específica, pois o seu conjunto de instruções torna-o flexível para implementação de vários tipos de aplicações.Porém, em ambos os casos os circuitos integrados digitais possuem as suas funções lógicas definitivas, implementadas na sua construção no processo de fabricação[5].

2.2 Dispositivos de Lógica Programável

O desenvolvimento de projetos de circuitos digitais tem evoluído rapidamente nas últimas décadas. A utilização de ferramentas EDA (*Eletronic Design Automation*) e o aperfeiçoamento dos dispositivos de lógica programável *PLD (programmable logic device)* tem simplificado e acelerado todo o ciclo de projeto [6].Os dispositivos de lógica programável *PLDs* são circuitos integrados que podem ser configurados pelo próprio usuário, não apresentam uma função lógica definida, até que sejam configurados. Possuem como principal característica a capacidade de programação das funções lógicas pelo usuário, eliminando-a do processo de fabricação do chip, facilitando assim as prováveis mudanças de projeto. Em comparação com outras tecnologias de circuitos integrados digitais os dispositivos de lógica programável apresentam um ciclo de projeto muito curto e custos muito baixo [5].

Os Dispositivos lógicos programáveis *PLDs* foram os dispositivos eletrônicos que possibilitaram a implementação da lógica flexível. Esses dispositivos lógicos possuem como sua principal característica a capacidade de programação pelo usuário, eliminando partes do processo de fabricação do *chip* tradicional e facilitando desta forma as mudanças de projeto. Para um bom entendimento do conceito de lógica flexível vamos considerar como tal, todo o circuito de lógica digital configurado pelo usuário final, incluindo os circuitos simples de baixa capacidade denominados

SPLDs (*Simple Programmable Logic Devices*) e os circuitos complexos de alta capacidade conhecidos como *EPLDs* (*Erasable Programmable Logic Devices*) ou HCPLDs (*High Complex Programmable Logic Devices*)[7].

2.2.1. Evolução dos Dispositivos de Lógica Programável

No início da década de 80, a industria eletrônica começou a utilizar os dispositivos programáveis *PLDs* de baixa densidade, na implementação de circuitos lógicos discretos. Hoje em dia, os *PLDs* integram em um único chip uma grande quantidade de blocos com muitas funções lógicas [7]. Esse mercado encontra-se em plena expansão, de forma que existem diversos fabricantes de equipamentos de comunicação de dados, filtros eletrônicos, etc optando por *PLDs,* em vez dos tradicionais chips customizados *ASICs.* A alta demanda de mercado por dispositivos programáveis reduziu drasticamente o seu custo por unidade. Outro fator que contribuiu com a redução do custo de fabricação dos dispositivos *PLDs* foi a utilização de novas tecnologias em seu projeto e na sua fabricação. Hoje em dia é possível encontrar no mercado dispositivos *PLDs* com integração, densidade, performance e, custo equivalentes a um circuito integrado *ASIC*. Sendo que, os dispositivos lógicos programáveis de alta capacidade são confeccionados com tecnologia *CMOS,* com densidades que podem atingir a 250.000 portas lógicas[7].

2.2.2. Estrutura Interna

A estrutura interna de um dispositivo programável PLD é classificada em arranjos de blocos (*gate arrays*), em arranjos de células (*cell based*) e em arranjos gerais de células (*macro cells*) [7]. Em todos os casos, a fabricação do dispositivo é baseada em uma matriz que conecta os blocos lógicos, barramentos, pinos de entrada e saída, e blocos de memória.

Os primeiros dispositivos *PLDs* possuíam apenas um único bloco lógico que realizava globalmente as operações, conectado aos pinos de entrada e saída do dispositivo. Com o aperfeiçoamento da tecnologia de fabricação, esses dispositivos receberam uma estrutura de interconexão programável, que viria flexibilizar mais ainda a programação. A evolução trouxe uma otimização do aproveitamento de espaço, aumentando ainda mais a interconexão da matriz de blocos lógicos [5].

2.2.3. Arquitetura Básica

A arquitetura básica de um dispositivo *PLD* é uma matriz programável de portas *ANDs* e portas *ORs*, cujas variáveis de entrada são ligadas em circuitos lógicos inversores ou diretamente aplicadas as entradas das portas *AND.* Em ambos os casos, as variáveis de entrada passam por fusíveis ou portas lógicas programáveis antes de alimentarem as portas *AND*, produzindo um único produto das variáveis de entrada. As saídas das portas *AND*, denominadas linhas de produtos, são também ligadas através de fusíveis às entradas das portas *OR.* Esse arranjo de portas *AND* e *OR* garante que cada saída das portas *OR* serão iguais ao nível lógico “1” (alto). Se o usuário final queimar ou programar um ou mais fusíveis das entradas *OR*, estas entradas terão nível lógico “0” (baixo), podendo-se, então, programar qualquer função que se queira na saída[9]. Na figura 2.1 observa-se um dispositivo PLA *(Programmable Logic Array*) [5]. Os primeiros dispositivos com esta configuração foram introduzidos pela Philips no início dos anos 70 .

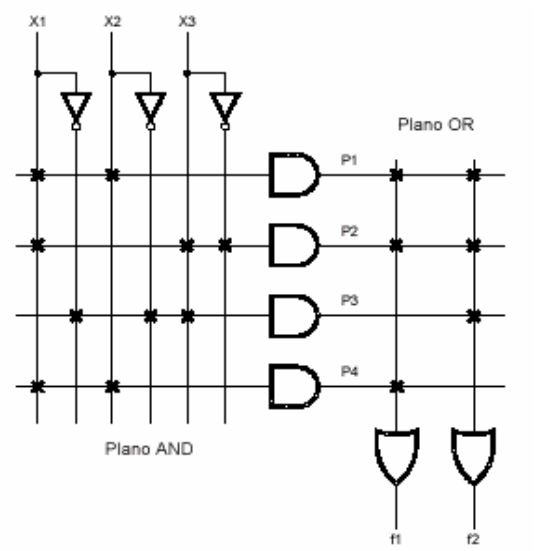

Figura 2.1 - Dispositivo Programável *PLA*

A tecnologia *PLA* no início de sua implantação apresentou alguns problemas. Esses problemas ocorreram devido a existência de dois planos de lógica programável que introduziam atrasos significativos de propagação de sinais elétricos e tinham um custo de fabricação elevado [5]. A tecnologia *PAL (Programmable Array Logic)* foi então desenvolvida para superar as deficiências dos circuitos *PLAs .* Os circuitos integrados *PALs* apresentavam um único nível de programação, um custo de fabricação menor e um melhor desempenho.Nesses dispositivos apenas o plano AND era configurável [9]. A figura 2.2 ilustra um esquema simplificado de um dispositivo programável *PAL.* Os dispositivos *PLAs* e *PALs* são agrupados numa categoria denominada *SPLDs* (*Simple Programmable logic devices*), cujas principais características são o baixo custo, alto desempenho e a baixa integração [7].

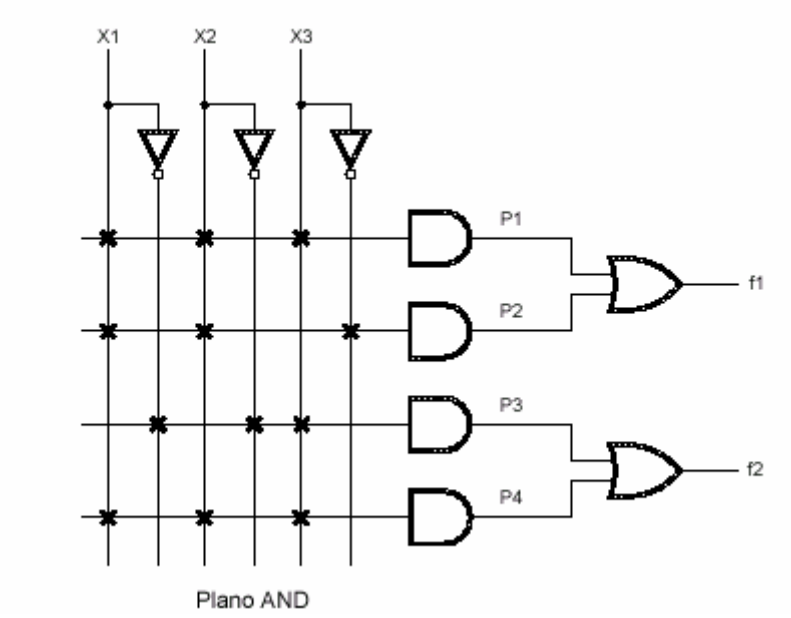

Figura 2.2 - Dispositivo Programável *PAL*.

2.2.3 Dispositivos Lógicos Programáveis e Apagáveis - EPLDs

Os dispositivos *EPLDs* (*Erasable Programmable Logic Devices*) foram introduzidos no mercado americano pela empresa Altera Corp. em 1983. Eram dispositivos programáveis e reprogramáveis pelo usuário, com alto desempenho, baixo custo por função e alta capacidade de integração. Um dispositivo *EPLD* pode ser aplicado, por exemplo, como uma máquina de estado ou decodificador de sinais, substituindo centenas de circuitos discretos que implementariam a mesma função[10]. As suas principais vantagens em relação aos circuitos discretos e *ASICs* tradicionais são:

- Programabilidade e reprogramabilidade : Permite que funções lógicas possam ser alteradas, simplificando o desenvolvimento de protótipos;
- Tecnologia CMOS : Menor consumo de energia elétrica;
- Integração em larga escala: Redução de tamanho da placa de circuito impresso, pois possibilita a eliminação de diversos componentes discretos;
- Simplificação e redução do tempo de desenvolvimento: Simplifica e reduz o tempo de desenvolvimento da placa de circuito impresso, pois permite que o projetista movimente os sinais elétricos, conforme desejado, determinando a distribuição da pinagem do dispositivo.
- Teste e depuração : As linguagens utilizadas na programação do dispositivo permitem a simulação, teste e depuração rápida do protótipo;

O princípio de funcionamento dos dispositivos *EPLDs* está baseado nos dispositivos PALs . Entretanto, os dispositivos *EPLDs* são constituídos de vários blocos PAL, com estruturas de somas de produto, interligadas por arranjos de conexões também programáveis [11]. A figura 2.3 ilustra a arquitetura interna de um dispositivo EPLD.

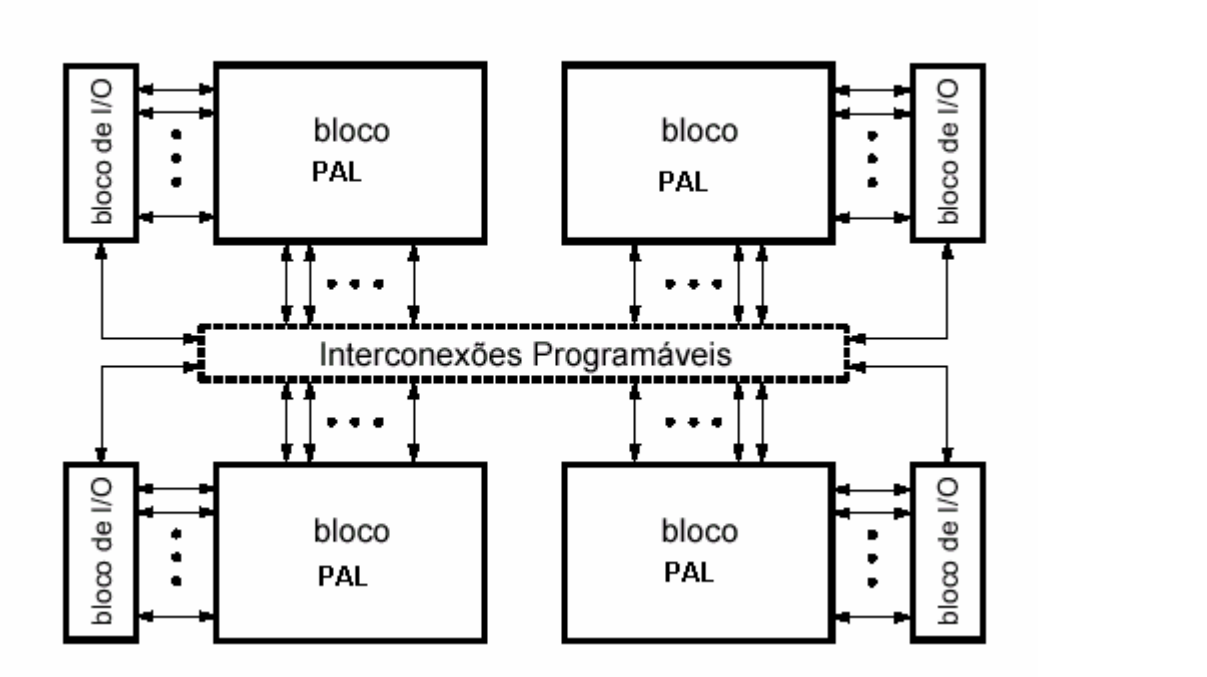

Figura 2.3 - Arquitetura interna de Dispositivo Programável *EPLD*

3 DISPOSITIVOS LÓGICOS PROGRAMÁVEIS DE ALTA CAPACIDADE

3.1 Estruturas Básicas

Os dispositivos lógicos programáveis simples *SPLDs* apresentados no seção 2 deste trabalho, permitem a implementação de uma pequena quantidade de portas lógicas. Entretanto, os dispositivos programáveis de alta capacidade *HCPLD* (*High Complex Programmable Logic Device*) que serão analisados nesta seção, possuem capacidade para mais de 600 portas lógicas [7]. Quanto maior o número de portas de um dispositivo *PLD*, maior será sua complexidade. Os dispositivos *HCPLD* dividem-se, basicamente em dois grupos : *CPLD* (*Complex Programmable Logic Devices*) e *FPGA* (*Field Programmable Gate Arrays*). A diferença básica entre os dois dispositivos está na estrutura interna de suas células lógicas e na metodologia de interligação dessas células.

3.1.2 Células Lógicas

Um dispositivo lógico programável de alta complexidade é formado por várias estruturas internas chamadas Células Lógicas [5]. No interior de cada célula existem dois modos possíveis para implementação de funções lógicas: matriz programável de portas *ANDs* e portas *ORs* e bloco de memória LUT (*Look-Up Table)*. A matriz programável de portas *ANDs* e portas *ORs* é uma estrutura utilizada em *PLDs* mais simples como *PLA, PAL e GAL* . Na implementação de funções mais complexas quando se utiliza o cascateamento de matrizes programáveis, tem-se como resultado vários níveis lógicos e um atraso (*delay*) maior na propagação do resultado de saída [5]. Por essa razão, em funções mais complexas deve-se utilizar o bloco de memória *LUT* para evitar que ocorram atrasos maiores de propagação, provocados pelo cascateamento das matrizes programáveis [5].

3.1.3 Bloco de Memória LUT (Look-Up Table)

O bloco de memória *LUT (Look-up table*) foi a solução encontrada para substituição da matriz de portas *ANDs* e portas *ORs* na programação de funções mais complexas num dispositivo de lógica programável de alta complexidade [7]. O *LUT* é um tipo de bloco lógico que contém células de armazenamento que são utilizadas para implementar pequenas funções lógicas. Cada célula é capaz de armazenar um único valor lógico 0 ou 1, funcionando basicamente como uma memória pré-programada de um bit de largura, onde suas linhas de endereço funcionam como entradas do bloco lógico e sua saída fornece o valor da função lógica. A figura 3.1 ilustra a estrutura de um bloco de memória *LUT* com 4 entradas e uma saída. As variáveis de entrada são utilizadas como chaves seletoras para selecionar um dos 16 *Latches* de leitura ou escrita. Para programação, deve-se colocar o endereço de entrada no decodificador de escrita e o valor correspondente na entrada de dados Din. Para leitura deve-se colocar o endereço de leitura no MUX e o valor a ser lido deve ser obtido na saída G. Como o bloco *LUT* não realiza combinação lógica das variáveis de entrada, não há níveis lógicos, logo o atraso (*delay*) da função lógica é dependente apenas do tempo de leitura da memória que o implementa [5] .

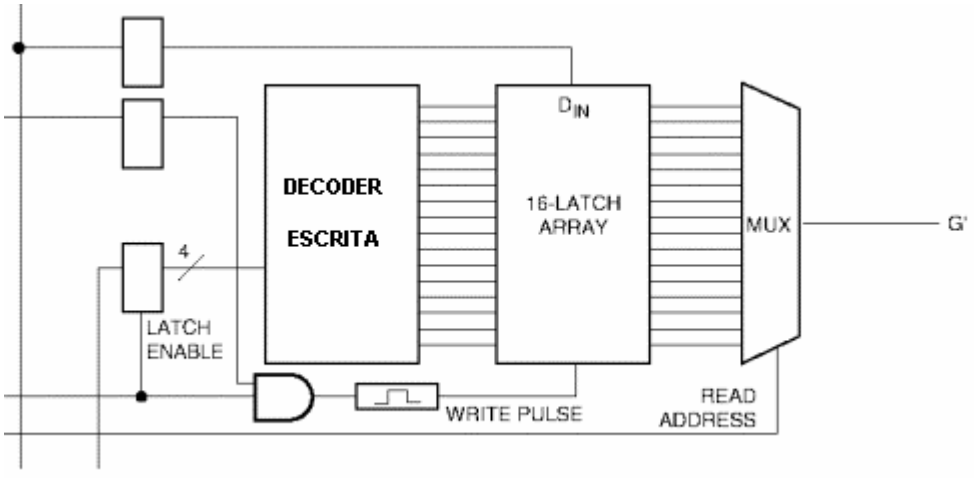

Figura 3.1 – Estrutura para um LUT com 4 entradas

3.2 Estrutura Interna

De uma forma geral, internamente os dispositivos de alta complexidade podem ser vistos como dispositivos que integram na sua estrutura centenas de macrocélulas programáveis, que são interligados por conexões também programáveis. Os CPLDs não diferem muito dos FPGAs, sendo que essas diferenças serão abordadas nas próximas seções.

3.2.1 Granularidade

Granularidade é uma característica dos dispositivos lógicos programáveis relacionada com o grão, sendo que entenda-se por grão a menor unidade configurável da qual é composta um dispositivo CPLD ou FPGA [12]. Os blocos lógicos dos CPLDs e FPGAs variam muito de tamanho e capacidade de implementação lógica. Uma primeira diferença entre eles está na granularidade de suas células lógicas. Um FPGA é mais granular que um CPLD [7]. Em um CPLD tem-se 10 elementos lógicos LEs em cada célula lógica, enquanto um FPGA possui, equivalentemente, um elemento lógico LE por célula lógica. A figura 3.2a ilustra a granularidade de um CPLD e a figura 3.2 b a granularidade de um FPGA.

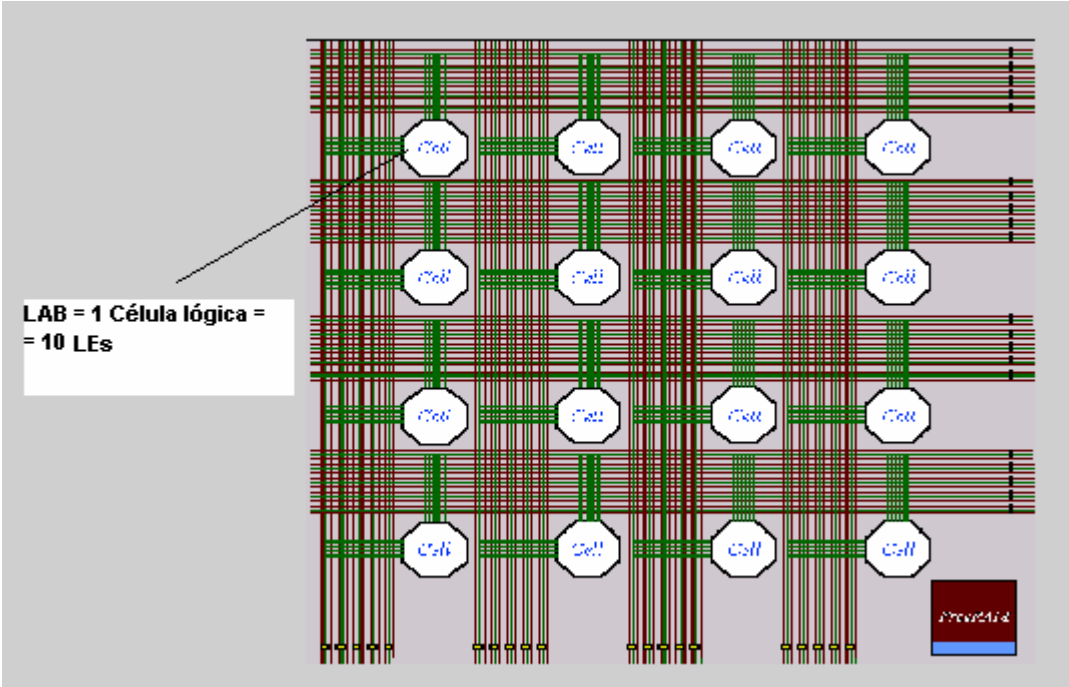

Figura 3.2a – Granularidade de um dispositivo CPLD.

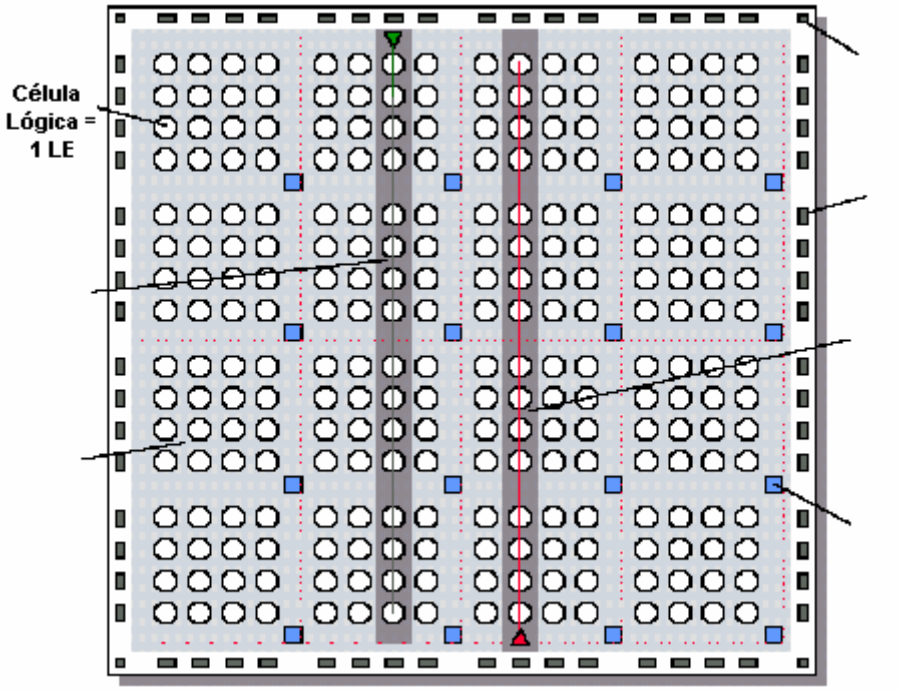

Figura 3.2b – Granularidade de um dispositivo FPGA

Algumas categorias de FPGAs foram criadas afim de classificá-los quanto ao bloco lógico que empregam. Essas categorias foram chamadas de granularidade fina (blocos simples e pequenos) e de granularidade Grossa (blocos mais complexos e maiores [13]. Um bloco de granularidade fina contem alguns transistores interconectáveis ou portas lógicas básicas. A principal vantagem desta granularidade é que os blocos lógicos são quase sempre totalmente utilizados. A desvantagem reside no fato de requererem uma grande quantidade de trilhas de conexão e chaves programáveis. Um roteamento desse tipo se torna lento e ocupa uma grande área do chip [13].

Um bloco de granularidade grossa geralmente é baseado em multiplexadores e blocos de memória LUTs. Os blocos lógicos baseados em multiplexadores tem a vantagem de fornecer um alto grau de funcionalidade com um número relativamente pequeno de transistores. No entanto eles possuem muitas entradas necessitando de muitas chaves comutadoras, o que sobrecarrega o roteamento. Logo, a tecnologia *antifuse* é a mais adequada para a fabricação desse tipo de FPGA, devido ao tamanho reduzido das chaves comutadoras *antifuse* [5]. Devido a maior granularidade de um FPGA existe uma forte dependência do roteamento, que pode ser percebida na figura 3.3b, pela presença de um número elevado de células lógicas

3.3 Estrutura de Interconexão

Os *FPGAs* e os *CPLDs* não possuem em sua arquitetura interna matrizes de portas *ORs* ou *ANDs* [5]. Suas arquiteturas internas consistem de arranjos de células lógicas formadas por blocos de memória LUT, *flip-flops* e multiplexadores, que podem ser utilizadas na implementação de funções lógicas. Basicamente, um *FPGA* e um *CPLD* apresentam na sua arquitetura blocos lógicos, blocos de E/S (entrada/saída) e chaves de interconexão. Os blocos lógicos formam arranjos bi-dimensionais e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas de células lógicas. Estes canais de roteamento possuem chaves programáveis que permitem conectar os blocos lógicos de maneira conveniente, em função das necessidades de cada projeto.

Para conexão entre as células e os blocos de saída, os dispositivos lógicos programáveis de alta complexidade possuem estruturas conhecidas como barramentos. Para os CPLDs mais complexos e para os FPGAs verificam-se barramentos horizontais e verticais em cada célula. A

segunda diferença básica entre um CPLD e um FPGA está na estrutura de constituição dos barramentos. Um CPLD é constituído por barramentos contínuos enquanto que um FPGA por barramentos segmentados [7].

.3.3.1 Barramento Contínuo

Um barramento contínuo é constituído por linhas de metal de comprimento e largura uniforme que atravessam toda a área do circuito integrado. Assim, a resistência e a capacitância de todas as interconexões são fixas e previsíveis, fazendo com que os atrasos de propagação (*delay*) entre quaisquer duas células lógicas do dispositivo sejam constante [7]. A figura 3.3 ilustra um dispositivo CPLD com barramentos contínuos.

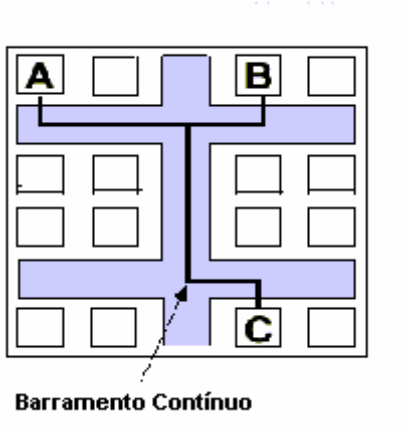

Figura 3.3 – CPLD com barramento interno contínuo.

3.3.2 Barramento Segmentado

Um barramento segmentado é constituído por vários segmentos de metal que atravessam toda a área do circuito integrado nas direções horizontal e vertical. Estes segmentos podem ser conectados de diversas formas, através de chaves lógicas programáveis. Assim, o número de segmentos necessários para a conexão entre duas células não é constante, depende muito da disposição das células e das múltiplas possibilidades de interconexão entre elas. Como não se pode definir o número de segmentos necessários para a conexão entre as células, antes do roteamento, não se pode dimensionar os atrasos de propagação (*delay*). O atraso total de propagação do dispositivo é função de um atraso devido ao barramento que varia de caso a caso, mais o atraso da estrutura [7]. A figura 3.4 ilustra um dispositivo FPGA com barramentos segmentados.

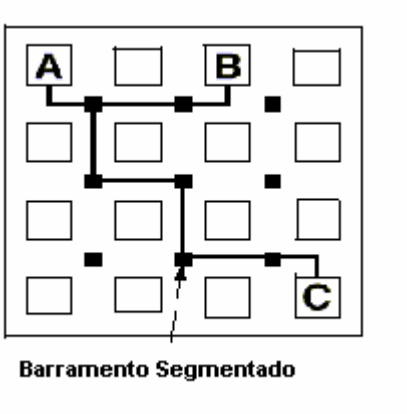

Figura 3.4 – FPGA com barramento interno segmentado.

3.3.3 Arquitetura de Roteamento

A arquitetura de roteamento de um dispositivo de alta complexidade é a forma pela qual os seus barramentos e as chaves de interconexão são posicionadas para permitir a interconexão entre as células lógicas [13]. Esta arquitetura deve permitir que se obtenha um roteamento completo e, ao mesmo tempo, uma alta densidade lógica.

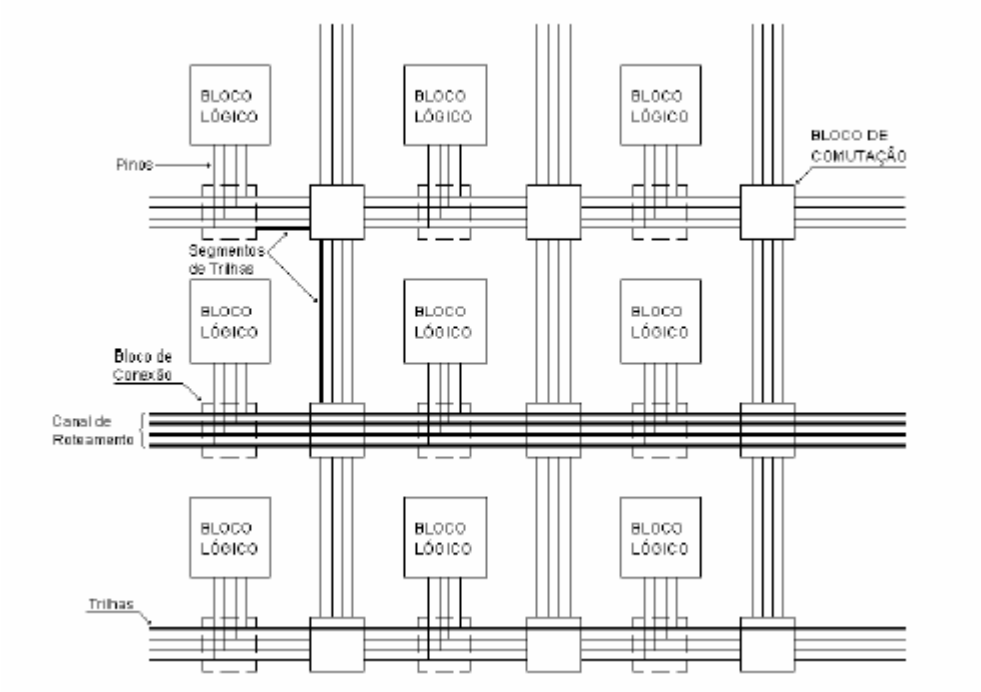

Figura 3.5 – Arquitetura de roteamento de um FPGA .

A figura 3.5 ilustra uma arquitetura de roteamento de um FPGA. A definição de alguns conceitos é fundamental para o perfeito entendimento desta arquitetura.

- Pinos: são entradas e saídas dos blocos lógicos.
- Conexão: ligação elétrica de um par de pinos.
- Rede: é um conjunto de pinos que estão conectados.
- Chave ou comutador de roteamento: utilizada para conectar dois segmentos de trilha.
- Segmento de trilha: segmento não interrompido por chaves programáveis.
- Canal de roteamento: grupo de duas ou mais trilhas paralelas.
- Bloco de conexão: permite a conectividade das entradas e saídas de um bloco lógico com os segmentos de trilhas nos canais.
- Bloco de comutação: permite a conexão entre os segmentos de trilhas horizontais e verticais.

3.3.4 Tecnologias de Programação das Chaves de Roteamento

As chaves ou comutadores de roteamento apresentam algumas propriedades como tamanho, resistência, capacitância e tecnologia de fabricação, que afetam diretamente o desempenho de um dispositivo FPGA [5]. Basicamente existem três tipos de tecnologia de programação das chaves de roteamento:

- *SRAM* (*Static Random Acess Memory*)
- *Antifuse*
- *Gate* flutuante

3.3.4.1 Tecnologia de Programação SRAM

Nessa tecnologia a chave de roteamento é um transistor de passagem controlado pelo estado de um bit de *SRAM*. A figura 3.6 ilustra uma célula de *RAM* estática controlando transistores de passagens ou multiplexadores. Como as memórias *SRAM* são voláteis, os *FPGAs* que utilizam essa tecnologia de programação em suas chaves de roteamento precisam de uma memória externa do tipo *EPROM, EEPROM* ou *FLASH*. A grande vantagem dessa tecnologia é que permite a reprogramação rápida do *FPGA* e a sua desvantagem é que essa tecnologia ocupa muito espaço no *chip* [6].

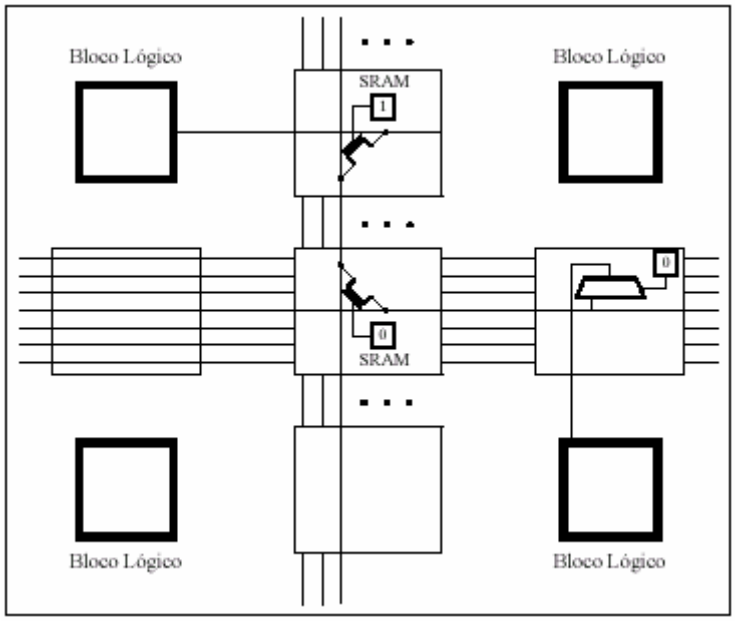

Figura 3.6 – Tecnologia de Programação SRAM .

3.3.4.2 Tecnologia de Programação Antifuse

A tecnologia *Antifuse* baseia-se num dispositivo de dois terminais, que no estado não programado apresenta uma alta impedância (circuito aberto). Aplicando-se uma tensão entre 11 e 20 volts o dispositivo "*queima*", fechando um contato de baixa impedância entre seus terminais [14]. O *Antifuse* não permite reprogramação, porém apresenta algumas vantagens como: tamanho reduzido, baixa capacitância quando não programado e baixa resistência quando programado.

3.3.4.3 Tecnologia de Programação Gate Flutuante ou EPROM

A tecnologia *Gate Flutuante* baseia-se em transistores *MOS (Metal Oxide Semiconductor)* , especialmente construído com dois *gates* flutuantes iguais aos usados nas memórias *EPROM* (*Erasable Programmable Read Only Memory*) e *EEPROM (Electrical Erasable Programmable Read Only Memory*). A figura 3.7 ilustra uma chave programável baseada em *gate* flutuante. A aplicação de uma tensão alta entre o *gate* 1 e o *source* do transistor faz com que o transistor não conduza, chave fechada. Ao expor o dispositivo aos raios ultravioletas (UV), o transistor volta a conduzir, chave aberta [5].

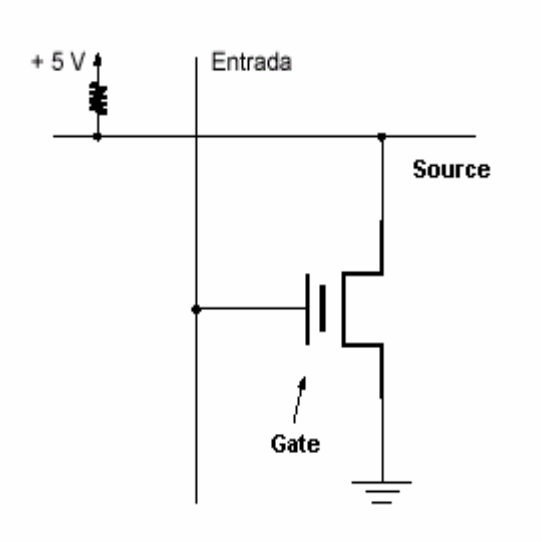

Figura 3.7 – Chave Programável baseada em *gate flutuante*

A maior vantagem dessa tecnologia é a sua capacidade de reprogramação e retenção de dados. Além disto é possível programar e reter as informações no dispositivo com o *chip* já instalado na placa, característica chamada programação em sistema (ISP - *In System Programmability*). Como desvantagens apresentam alto consumo de energia e ocupam muito espaço físico da célula [2].

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO DE PROJETO UTILIZANDO LÓGICA PROGRAMÁVEL ESTRUTURADA

4.1 Ferramenta de Projeto EDA

O processo de projeto com lógica programável estruturada envolve várias etapas de projeto que geralmente são automatizadas.A utilização de ferramenta *EDA* (*Eletronic Design Automation*) tem simplificado e acelerado todo o ciclo de projeto [5]. Um ambiente típico para desenvolvimento e implementação de projetos utilizando lógica programável estruturada, baseado em ferramenta *EDA*, consiste de vários programas interconectados, conforme ilustrado na figura 4.1.

A entrada do projeto pode ser feita criando-se um diagrama esquemático com uma ferramenta gráfica ou utilizando-se uma linguagem de descrição de hardware *HDL* (*Hardware Description Language*). Como a lógica inicial não está otimizada, algoritmos são utilizados para uma síntese algébrica das equações booleanas, manipulando-as para obtenção de uma redução da área a ser ocupada no chip, como também uma redução no atraso de propagação (*delay*) dos sinais envolvidos. A ferramenta *EDA* possibilita desde a entrada em linguagem de descrição de hardware (*HDL*) ou esquemático, até a configuração final do dispositivo lógico programável passando por síntese, arquivo *netlist*, mapeamento, roteamento e posicionamento [2].

Para transformar as equações booleanas em células lógicas, por exemplo, a rede otimizada alimenta um programa de mapeamento na tecnologia adotada (ALTERA, XILINS, etc). Esse passo mapeia as equações em células lógicas, que também é uma oportunidade para otimizar, ou minimizar o número total de células lógicas requeridas (otimização de área), ou o número de células lógicas em caminhos críticos (otimização de atraso). O conjunto de células lógicas é então passado para um programa de posicionamento, que seleciona uma localização específica no dispositivo lógico programável para cada célula. Algoritmos típicos de posiciomento geralmente tentam minimizar o comprimento total das interconexões necessárias [15].

O passo final da ferramenta EDA é realizado pelo software de roteamento, que aloca os recursos de roteamento do dispositivo para interconectar as células lógicas posicionadas. As ferramentas de

roteamento devem assegurar que 100% das conexões necessárias são realizadas, e devem procurar maximizar a velocidade das conexões críticas, porém essa meta nem sempre é utilizada para configurar o dispositivo [2].Por último a saída do sistema EDA e utilizada para configurar (programar) o FPGA.

Os Softwares MAX+PLUS II (***M**ultiple **A**rray Matri**X** **P**rogrammable **L**ogic **U**ser **S**ystem*) e QUARTUS II são as ferramentas EDA de desenvolvimento e programação das diversas famílias de dispositivos lógicos programáveis da empresa norte americana ALTERA, utilizadas nesse trabalho. A entrada de projetos pode ser efetuada nos seguintes modos:

- Desenho esquemático;
- *AHDL*;
- Formas de onda;
- Importação de arquivos padrão *edif* (usados pelo *CAD*, *TANGO*,etc.);
- Importação de arquivos padrão *Xilins*;
- *VHDL*;
- *Verilog*;

Para facilitar o ambiente de projeto, estas ferramentas EDAs, permitem a síntese lógica, arquivo *Netlist*, mapeamento, posicionamento e roteamento, simulação, análise de tempo e potência, e programação do dispositivo.

4.2 Etapas de Projeto

O processo de desenvolvimento e implementação de projeto pode ser dividido nas seguintes etapas[16]:

- Especificação e entrada do projeto
- Síntese lógica e mapeamento
- Posicionamento e roteamento
- Verificação e testes
- Configuração /programação do dispositivo lógico programável

4.2.1 Especificação e Entrada do Projeto

A especificação do projeto é apresentada em termos abstratos ou em métodos formais, seguida pela análise da viabilidade da implementação, obtida através de simulação de alto nível [2]. Nessa etapa a linguagem de programação a ser utilizada deve ser o mais próximo possível da linguagem humana. Assim a implementação do projeto será expressa por uma descrição de hardware validada pelo ambiente EDA. Geralmente os ambientes EDA aceitam como entrada: os editores esquemáticos (portas lógicas, macros,etc) e as linguagens de descrição de hardware HDL [2].

4.2.1.1 Editor Esquemático

As ferramentas de captura de esquemático ou editores de esquemáticos permitem que o projetista especifique o circuito como um diagrama lógico em 2D, conectando componentes lógicos programáveis com recursos de roteamento. A figura 4.2 ilustra uma tela do editor de esquemático do Software MAX+PLUS II. Os componentes lógicos estão contidos em uma bibliotecas de macros fornecidas pelo software ou podem ser definidas pelo próprio usuário. Geralmente, as bibliotecas contem portas lógicas, pinos de *I/O*, *buffers*, multiplexadores*, flip-flops*, *latches*, decodificadores, registradores, contadores, comparadores, memórias, funções aritméticas, e outras funções especiais. Estão disponíveis também símbolos especiais para controle do mapeamento, posicionamento e roteamento durante a fase de implementação do projeto. As macros podem ser *soft,* que significa que o posicionamento e roteamento ficarão a cargo das ferramentas ; ou *hard*, que significa que já estão pré-mapeadas, pré-posicionadas e pré-roteadas [2].

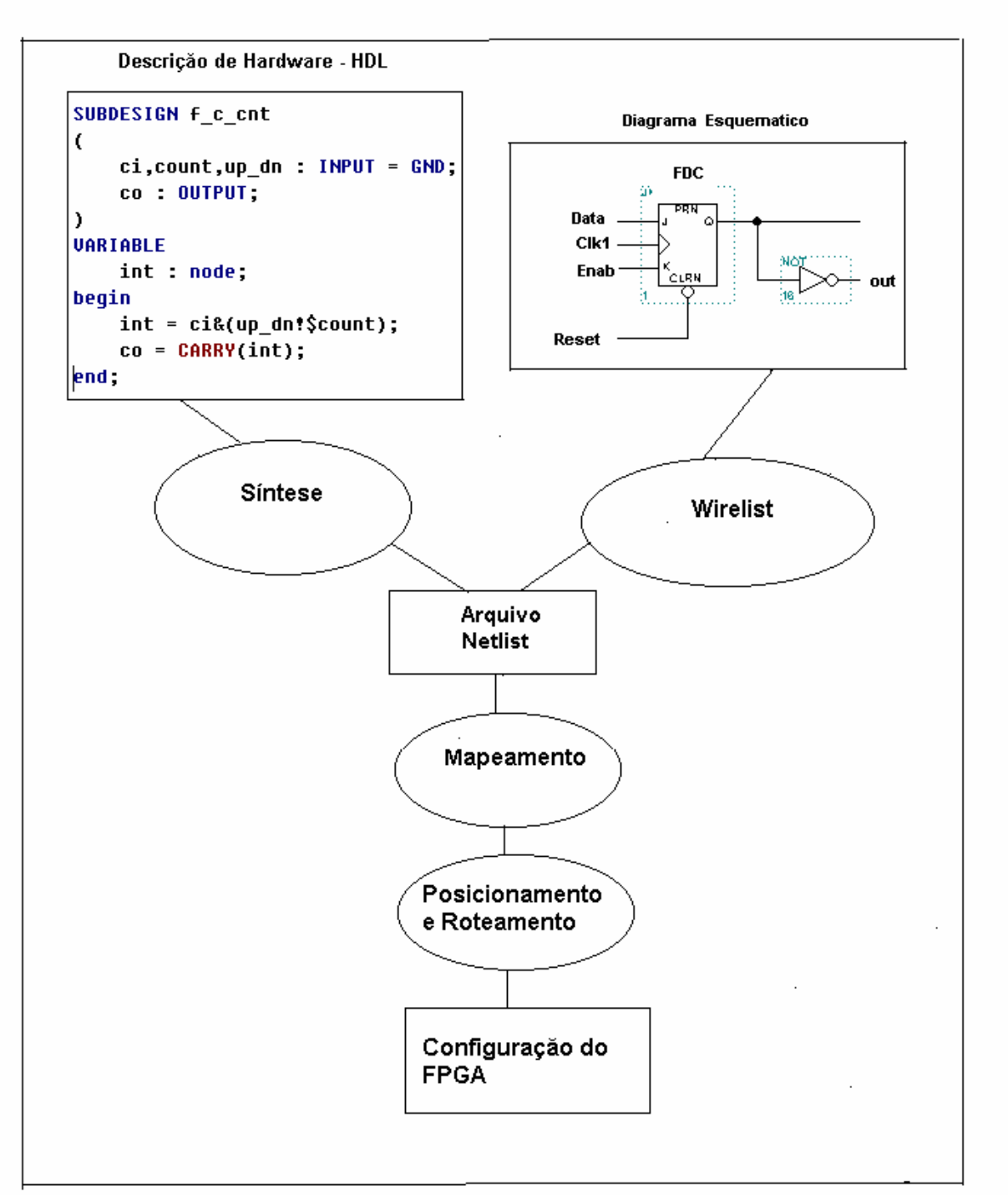

Figura 4.1 – Ambiente de Desenvolvimento de Projetos com Sistema EDA

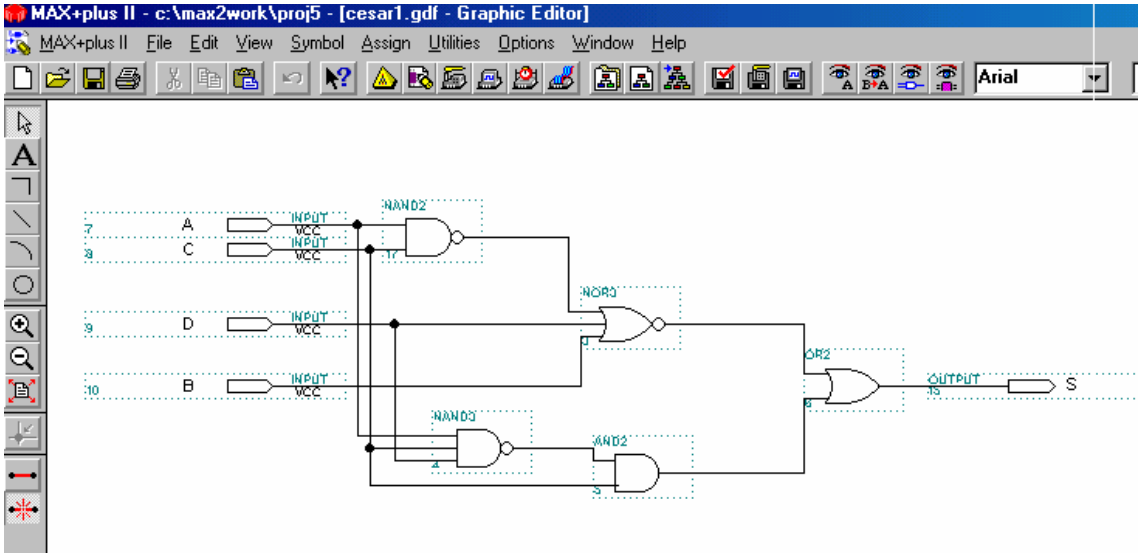

Figura 4.2 – Editor de esquemático do Software MAX+PLUS II.

4.2.1.2 Linguagens de Descrição de Hardware

A introdução de esquemáticos não é o modo ideal para projeto de circuitos complexos, torna-se uma tarefa tediosa quando se projetam grandes blocos repetidamente. A medida que os projetos ficam mais complexos, as descrições em nível de portas lógicas tornam-se inviáveis, fazendo com que seja necessário descrever os projetos em modos mais abstratos. As linguagens de descrição de hardware também conhecidas como *HDL* (*Hardware Description Language*) foram desenvolvidas para auxiliar os projetistas a documentarem projetos e simularem grandes sistemas, principalmente em projetos de dispositivos ASICs [2].

A programação de um dispositivo PLD consiste em se carregar no *chip* as funções lógicas que serão implementadas nele. Em um FPGA, por exemplo, para que cada bloco lógico e suas interconexões sejam configuradas são necessários algumas centenas de bits. Cada bit de configuração define o estado de uma célula de memória, que controla uma função *" look-up table "* ou seleciona uma entrada de um multiplexador, ou define o estado de uma chave de interconexão [5].

Existem diversas linguagens de descrição de hardware disponíveis, sendo as mais comumente utilizadas: ***ABEL*** (*Advanced Boolean Equation Language*) e ***VHDL*** (*Very High Speed Integrated Circuit Hardware Description Language*). A linguagem *ABEL* foi a primeira linguagem *HDL* a ser desenvolvida. Foi criada pela empresa americana Data I/O Corporation para programar dispositivos PLD, sendo uma linguagem mais simples que a linguagem *VHDL*. Já a linguagem *VHLD* é capaz de programar sistemas de maior complexidade como dispositivos *HCPLDs (FPGA e CPLD*) [1].

As linguagens de descrição de hardware HDL são utilizadas para descrever o comportamento de um sistema digital de variadas formas, inclusive equações lógicas, tabelas verdades e diagramas de estado que utilizam declarações como a linguagem C [17]. O compilador dessas linguagens permite simular, projetar e implementar funções lógicas em dispositivos PLDs como PALs, CPLDs e FPGAs.

4.2.1.3 Linguagens Abel

A linguagem ABEL (Advanced Boolean Equation Language) é uma linguagem descritiva de hardware (HDL), que permite programar descrições de um circuito lógico. Foi criada pela empresa Data I/O Corporation para programar funções lógicas em dispositivos simples de lógica programável SPLD [1]. Um programa em linguagem ABEL é um arquivo texto que contem vários elementos:

- Cabeçalho incluindo nome do módulo, opções e título ;

- Declarações que identificam pinos, constantes, nodos, configurações, estados lógicos, bibliotecas, etc;
- Descrições da lógica incluindo equações, tabela-verdade, diagrama de estados;
- Vetores de teste, que especificam a saída esperada, em função de certas entradas;

A linguagem ABEL é suportada por uma linguagem de baixo nível, chamada de Compilador ABEL.O trabalho deste compilador é traduzir o arquivo texto ABEL em um padrão que pode ser carregado diretamente num chip PLD. A linguagem ABEL permite programar PLDs com tabelas verdades, expressões algébricas,etc. O compilador manipula as expressões e minimiza o resultado das equações de modo que caibam na estrutura física do dispositivo PLD [1].

4.2.1.3.1 Estrutura Básica de um Programa em Abel

A Tabela 4.1 apresenta uma estrutura básica de um programa em linguagem ABEL.

Module	Nome do módulo	-
Title	*String* de título	-
Device	Identificação do dispositivo	Tipo do dispositivo
Pin declaration	Declarações dos pinos	
Other declarations	Outras declarações	
Equations	Equações	
Test_vectores	Vetores de teste	
End	Nome do módulo	

Tabela 4.1 – Estrutura típica de um Programa em ABEL.

4.2.1.3.2 Declarações de um Programa em Abel

Um programa em linguagem ABEL, descrevendo um circuito de alarme é listado na figura 4.3. O programa começa na primeira linha com uma declaração de módulo seguida por um nome identificador do módulo. Os programas que consistem de vários módulos terão cada módulo com suas declarações de nome, título, dispositivos, pinos, equações, etc independentes [1].

A segunda linha é uma declaração opcional e identifica o projeto. O título deve estar entre aspas simples. Esta linha é ignorada pelo compilador, mas é conveniente para a documentação do projeto.

A terceira linha é uma declaração opcional e associa um nome identificador de dispositivo com um PLD específico. Esta declaração tem que terminar com um ponto e vírgula, e o nome do PLD específico deve estar entre aspas simples.

A quarta linha é uma declaração de pinos, que indica ao compilador quais os nomes simbólicos serão associados com os pinos dos dispositivos externos. Se o nome do sinal for precedido com o sinal de exclamação (!), indica um sinal ativo nível baixo, ou seja o sinal será invertido (NOT) e o complemento do sinal aparecerá no pino. A declaração do pino pode ou não incluir os números dos pinos. Se os números não forem dados, o compilador nomeia-os baseado na sua capacidade de nomeação de pinos do dispositivo.

A palavra chave *" istype*" é uma declaração de atributo opcional para um pino de saída. Por exemplo, usa-se a palavra *´com ´* para indicar que a saída é um sinal combinacional ou a palavra *´reg´* para indicar um sinal de clock. Este atributo só é válido para pinos de saída e deve ser sempre utilizado. Os comentários podem ser inseridos em qualquer lugar no programa e devem ser iniciados com aspas duplas. A declaração *" node*" tem o mesmo formato que a declaração de pinos. São sinais internos que não são conectados a pinos externos. Por exemplo: *tmp1 node [istype ´com ´]*.

A quinta linha "*outras declarações*" permite que o programador defina constantes, configurações, macros e expressões que podem simplificar o programa.

```
1 Module              Circuito_Alarme
2 Title              'Circuito de Alarme'
3 Device              ALARMCKT                  'EPF10K10TC144-4';
4 Pin declaration    "Inputs pins
                      Panico, Habilita, Saida   pin 1, 2, 3;
                      Janela, Porta, Garagem    pin 4, 5, 6;
                     "Outputs pins
                      Alarme                    pin 11 istype 'com';
5 Other declarations  "Constant definition
                      X=.X.;
                      "Intermediate equation
                      Seguranca=Janela & Porta & Garagem
6 Equations           Equations
                      Alarme= Panico # Habilita & !Saida & !Seguranca
7 Test_vectores       Test_vectores
                      ([Panico, Habilita, Saida, Janela, Porta, Garagem]->[Alarme])
                       [   1,     .X.,      .X.,   .X.,   .X.,     .X. ]->[  1   ];
                       [   0,      0,       .X.,   .X.,   .X.,     .X. ]->[  0   ];
                       [   0,      1,        1,    .X.,   .X.,     .X. ]->[  0   ];
                       [   0,      1,        0,     0,    .X.,     .X. ]->[  1   ];
                       [   0,      1,        0,    .X.,    0,      .X. ]->[  1   ];
                       [   0,      1,        0,    .X.,   .X.,      0  ]->[  1   ];
                       [   0,      1,        0,     1,     1,       1  ]->[  0   ];
8 End                 Circuito_Alarme
```

Figura 4.3 – Listagem do Programa em Linguagem ABEL.

A sexta linha é uma declaração de equação, que indica a equação lógica do circuito e define os sinais de saída em função dos sinais de entrada. As equações são escritas como na programação convencional, cada uma terminada por ponto e vírgula.

A sétima linha é uma declaração de teste de vetores, que associa combinações de entradas com valores de saídas esperadas. Essas combinações são utilizadas para simulação e teste do circuito. O compilador reconhece muitas constantes especiais, como por exemplo **.X.**, um único bit cujo valor é *'don´t care'*. A oitava e última linha é uma declaração de fim, que marca o fim do módulo [1].

4.2.1.4 Linguagem VHDL

A linguagem VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) é uma linguagem para descrever sistemas eletrônicos digitais, que foi desenvolvida nos anos de 1980 pelo Departamento de Defesa dos Estados Unidos. Em 1987, a linguagem VHDL foi normalizada pelo IEEE, tornando-se um padrão mundial, ao lado da linguagem *Verilog*, uma alternativa também expressiva no mercado de projetos de hardware. Atualmente, todas as ferramentas EDA de desenvolvimento de hardware aceitam essas linguagens como entrada, de forma que um projeto baseado em VHDL ou *Verilog* pode ser implementado com qualquer tecnologia [20].

A linguagem VHDL permite :

- Através de simulação verificar o comportamento do sistema digital;
- Descrever o hardware em diversos níveis de abstração, por exemplo em algorítmico ou comportamental ou transferência entre registradores (RTL);
- Simulação e síntese;

4.2.1.4.1 Estrutura Básica de um Projeto em VHDL

Todos os projetos em VHDL são expressos em termos de bibliotecas (*package*), pinos de I/O (*entity*), arquitetura (*architecture*) e configuração (*configuration*). A figura 4.4 ilustra a estrutura básica de um projeto em VHDL.

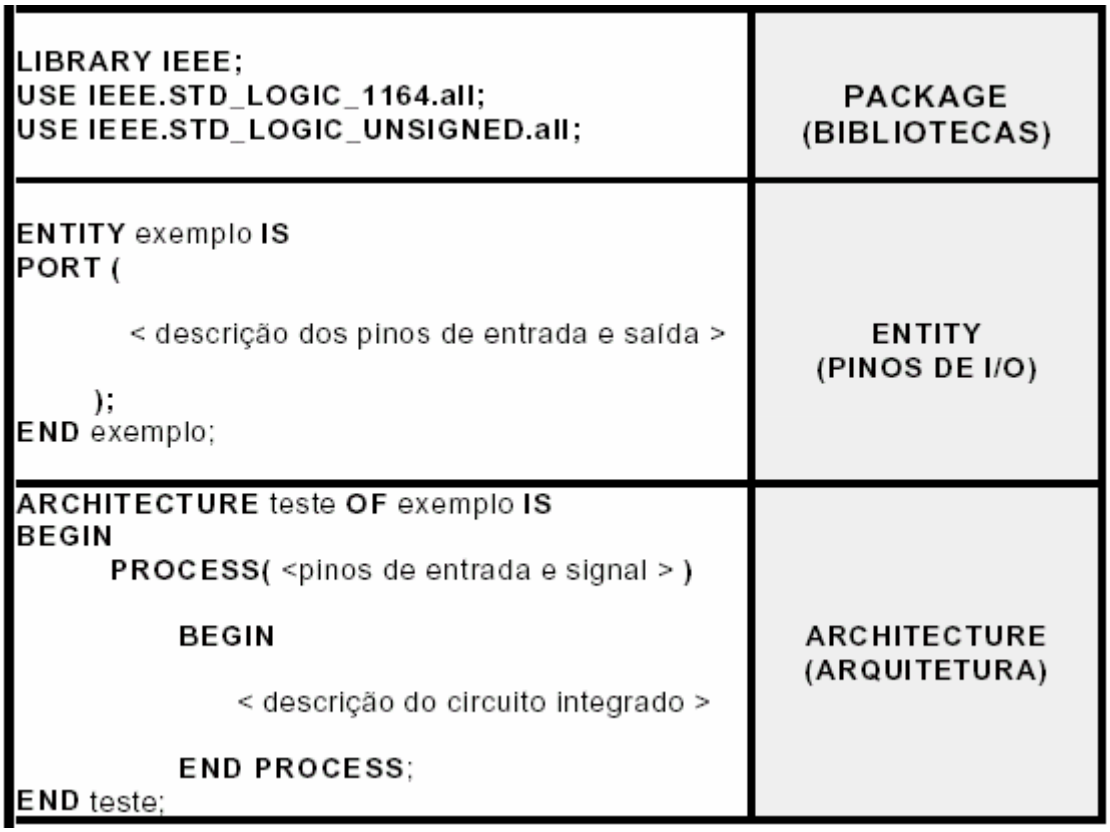

Figura 4.4 – Estrutura Básica de um Projeto em VHDL

4.2.1.4.2 Pacotes (*Package*)

Um pacote é uma coleção de tipos de dados e subprogramas normalmente usados em um projeto [2]. Os pacotes ou bibliotecas contém uma coleção de elementos incluindo descrição do tipo de dado, permite a reutilização de um código já escrito e armazena declarações de tipos, constantes, subprogramas e mnemônicos. A figura 4.5 apresenta um exemplo de pacote (*packge*).

```
PACKAGE < biblioteca > IS
    function soma (a,b:bit)return bit;
    subtype dado is bit_vector (32 downto 0)
    constante mascara : bit_vector (3 downto 0):="1100";
    alias terceiro_bit: bit IS dado (3);
END <biblioteca>;
```

Figura 4.5 – Exemplo de pacote (package) em VHDL.

4.2.1.4.3 Entidade (Entity)

Uma entidade é uma abstração que descreve um sistema, uma placa, um chip, uma função ou uma porta lógica [20]. Na declaração de uma entidade, descreve-se o conjunto de entradas e saídas. A figura 4.6 apresenta uma declaração de entidade. As portas (*Ports*) correspondem aos pinos de entrada e saída e definem os canais de comunicação entre a entidade de projeto e o mundo exterior. A definição de uma porta envolve descrição de seu modo e tipo. O modo da porta especifica a direção do fluxo da informação através da porta. No modo ***IN*** a informação flui para a entidade, no modo ***OUT*** a informação flui desde a entidade e no modo ***INOUT*** a informação flui em qualquer direção. Outro modo de propósito especial é o ***BUFFER***.

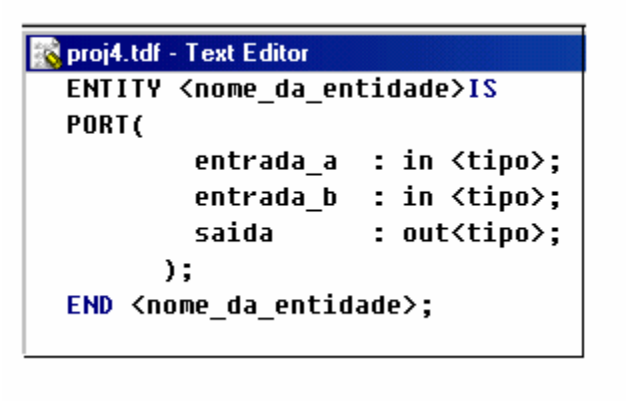

Figura 4.6 – Estrutura de uma declaração de entidade

O tipo de porta especifica o conjunto de valores que uma porta pode assumir. Os valores das portas podem ser representado por níveis de tensão (bit). valores falso ou verdadeiro, dígitos binários, etc. Cada um destes conjuntos é um tipo, e cada um pode ser uma forma de abstração do mesmo fenômeno eletrônico [19].

Tipo	Descrição
bit	Assume valores “0” ou “1”. **X: in bit;**
bit_vector	Vetor de bits. **X: in bit_vector (7 downto 0);** **X: in bit_vector (0 to 7);**
std_ logic	**X: in std_logic;**
std_logic_vector	**X: in std_logic_vector (7 downto 0);** **X: in std_logic_vector (0 to 7);**
boolean	Assume valores TRUE ou FALSE

Tabela 4.3 – Tipos mais utilizados de entradas e saídas

4.2.1.4.4 Arquitetura (Architecture)

A função de uma entidade é determinada pela sua arquitetura. A organização de uma arquitetura é dada por declarações (sinais, constantes, componentes, subprogramas, etc) e comandos (begin, end). A figura 4.7 apresenta a arquitetura de um comparador de 4 bits. A arquitetura pode ser descrita por três formas distintas: descrição comportamental, descrição por fluxo de dados (data flow) e descrição estrutural [20].

a) Descrição Comportamental

A descrição comportamental é a forma mais flexível e poderosa de descrição. Nela são definidos os processos concorrentes (*process*). A cada processo é associada uma lista de sensibilidade, que indica quais são as variáveis cuja alteração deve levar à reavaliação da saída [20].

b) Descrição por Fluxo de Dados

Neste tipo de descrição, os valores de saída são atribuídos diretamente, através de expressões lógicas. Todas as expressões são concorrentes no tempo [20].

c) Descrição Estrutural

A descrição estrutural apresenta *Netlists* e instanciação de componentes básicos, ou seja, é como se fosse uma lista de ligações entre componentes básicos pré- definidos.

4.2.1.4.5 Configuração

Uma declaração de configuração é usada para ligar uma instância de componente a um par de arquitetura *entity* (entidade). A configuração define as arquiteturas que serão utilizadas. A figura 4.11 apresenta um exemplo de configuração de um processador.

```
configuration test_config of processor is
    use work.processor_types.all
    for block_structure
        for control_unit
            configuration items
        end for;
        for data_path
            configuration items
        end for;
    end for;
end test_config;
```

Figura 4.10 – Exemplo de configuração de um processador.

5 CONTROLADOR LÓGICO PROGRAMÁVEL – CLP TRADICIONAL

5.1 Histórico e Evolução

Até o final da década de 60, quando a linha de produção tinha que ser mudada, os relés e os dispositivos eletromecânicos exigiam um empenho muito grande da equipe de operação e manutenção da fábrica. Além de apresentarem desgastes pelo fato de serem mecânicos, precisavam de manutenção constante e cara. A partir de então detectou-se a necessidade de se desenvolver novos controladores que fossem facilmente programados e reprogramados tanto por operadores quanto por pessoal especializado (técnicos e engenheiros), tivesse tempo de vida útil longa e fossem imunes ao ambiente hostil de fábrica [22].

Os CLPs são controladores microprocessados especiais, que foram concebidos por um grupo de engenheiros da divisão de hidramática da General Motors em 1968, operam como seqüenciadores de estados de uma máquina utilizando instruções de temporização, contadores, operações lógicas e aritméticas, movimentação de dados, etc. Atualmente, os CLPs implementam um algoritmo de controle de um processo seqüencial, a partir de um software armazenado em sua memória, que roda numa arquitetura tipo *Von Neumann*, baseada em um microcontrolador e perféricos ASICs.

5.2 Descrição de um CLP

Os componentes básicos de um CLP são a fonte de alimentação, Unidade Central de Processamento (CPU), memória e dispositivos de entrada e saída. Dependendo do fabricante, os componentes básicos podem vir num único encapsulamento ou em módulos separados, que podem estar juntos num mesmo *rack* ou separados [22]. A figura 5.1 apresenta a arquitetura básica de um CLP baseado em microcontrolador ou microprocessador. A fonte de alimentação converte 120/240 Vca para 5 Vcc ou 12 Vcc que alimentam os circuitos elétricos/eletrônicos do CLP. As entradas podem ser digitais ou analógicas e são provenientes de elementos de campo como sensores, botões, pressostatos, termostatos, chave fim de curso, etc. As saídas também podem ser digitais ou analógicas e se caracterizam pelo nível de tensão ou corrente, podendo ser solenóides, contatores, válvulas,etc [23]. As saídas e entradas são geralmente isoladas do campo através de acopladores óticos. A memória é constituída por memórias *RAM e FLASH EPROM*. Nelas são armazenados o sistema operacional, o programa de controle, resultados de operações, informações intermediárias, estado dos módulos de entrada/saída, etc. A CPU é formada pelo microprocessador ou microcontrolador e seus circuitos ASICs de controle e comunicação. A CPU lê os sinais de entrada, executa a lógica de controle segundo as instruções do programa de aplicação, realiza cálculos, executa operações lógicas, etc para em seguida enviar os sinais apropriados ás saídas [24].

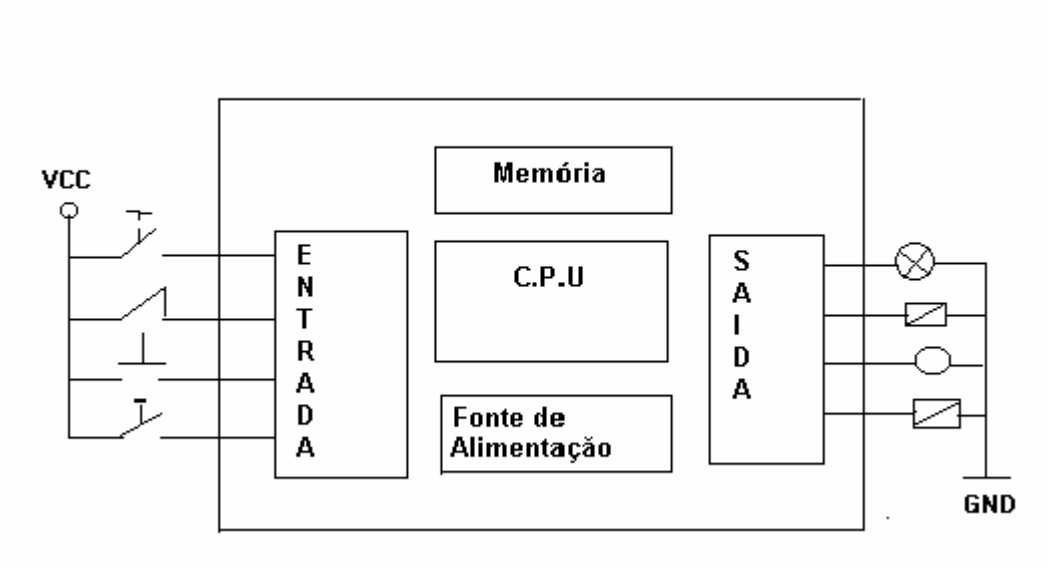

Figura 5.1 – Arquitetura de um CLP Tradicional.

5.2.1 Ciclo de trabalho da CPU

O processamento interno de uma instrução de programa em uma CPU é constituído basicamente por dois passos. O microprocessador lê as instruções armazenadas na memória, uma de cada vez e executa cada instrução. A execução do programa consiste na repetição seqüencial do processo de leitura e execução da instrução. Claro que a execução de uma instrução pode, pelo seu lado, envolver um certo número de passos. Pode-se justificar a divisão do processamento da instrução em dois estágios de "*busca*" e de "*execução*" da seguinte forma: A busca é uma operação comum para cada instrução e consiste na leitura de uma localização na memória.A execução da instrução pode envolver várias operações e depende da natureza da instrução.

O ciclo de busca da instrução não está diretamente relacionado com o processo no qual o CLP está inserido, mas é condição determinante para o microprocessador executar o programa de controle que está carregado em sua memória. Essa necessidade de busca da instrução demanda tempo do microprocessador o qual poderia estar sendo utilizado na execução das tarefas pertinentes ao processo de controle.

5.3 Princípio de Funcionamento de um CLP

O CLP funciona segundo um ciclo de varredura chamado *scan time*. A figura 5.2 ilustra um diagrama em blocos do funcionamento de um CLP. O *scan time* é o tempo que o CLP leva para ler as entradas, atualizar as suas imagens na memória, processar o programa de controle (instruções de usuário), acionar as saídas e atualizar suas imagens na memória [24]. Os elementos principais de um ciclo de varredura (*scan*) são:

- Varredura das entradas : Durante a varredura das entradas, o CLP examina os dispositivos externos de entrada quanto à presença ou ausência de tensão, isto é, um estado "*energizado*" ou "*desenergizado*". O estado das entradas é armazenado temporariamente em uma região da memória chamada "*tabela imagem da entrada*".
- Varredura do programa : Durante a varredura do programa, o CLP examina as instruções no programa de controle (aplicação), usa o estado das entradas armazenadas na tabela imagem de entrada e determina se uma saída será ou não "*energizada*". O estado resultante das saídas é armazenado em uma região da memória chamada "*tabela imagem de saída*".
- Varredura das saídas: Baseado nos dados da tabela de imagem de saída, o CLP "*energiza*" ou "*desenergiza*" seus circuitos de saída que exercem controle sobre dispositivos externos.

O *scan time* muitas vezes pode ocasionar problemas graves no controle de processos industriais, que envolvam sinais de entrada rápidos, por não reconhecer uma entrada durante o seu ciclo de varredura.

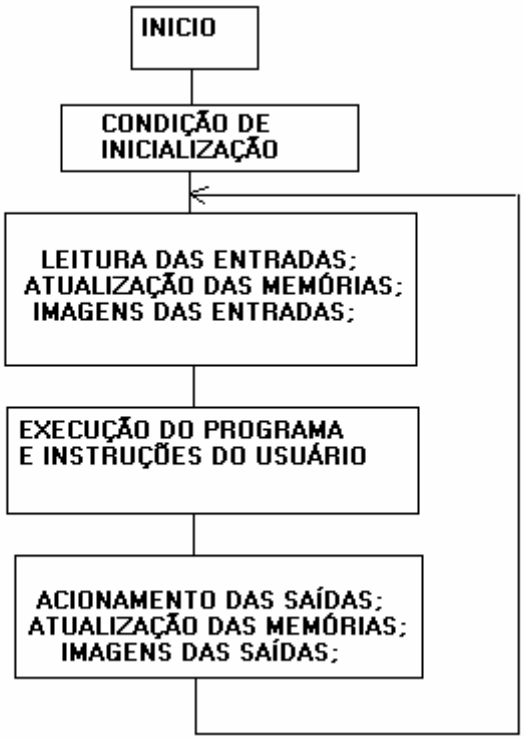

Figura 5.2 – Diagrama em blocos do funcionamento de um CLP Tradicional

5.4 Tempo de atraso para acionamento de uma Saída

Dependendo da duração dos sinais de entrada, o CLP pode demorar mais para acionar a saída ou mesmo nunca reconhecer uma entrada [25]. Para ilustrar este problema, a figura 5.3 apresenta os sinais de entradas de duração variável 1, 2 e 3. O sinal de entrada 1 é reconhecido somente durante o *san time* 2 (final de amostragem do sinal 1 e início do ciclo de *scan* 2), o sinal de entrada 2 somente no *scan time* 3 (final de amostragem do sinal 2 e início do ciclo de *scan* 3), mas a entrada 3

não será reconhecida porque tem duração muito curta (ocorre durante a execução do programa no ciclo de *scan* 3). Para evitar problemas desse tipo os CLP´s empregam técnicas de alongamento do pulso de entrada até o início do próximo ciclo de *scan* ou utilizam uma técnica de interrupção e reinício do ciclo de *scan* [22].

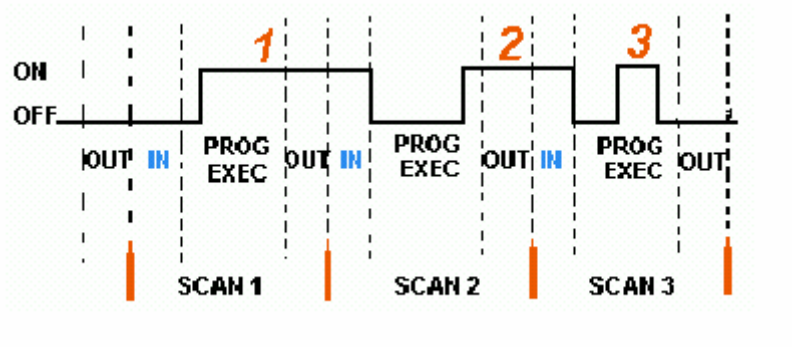

Figura 5.3 – Sinais de entrada rápidos durante os ciclos de *scan.*

Uma estimativa para duração do ciclo de varredura é considerar o atraso de 1 a 10 us por instrução mais o tempo de atualização das entradas e saídas, que é de aproximadamente alguns milisegundos. O atraso máximo que o acionamento de uma saída poderá ter é quando a entrada que deve acioná-la for sentida somente no segundo ciclo de varredura, quando nesse caso o atraso será de 2 ciclos de varredura menos 1 tempo de atualização das entradas [25). A figura 5.4 ilustra o maior tempo de atraso possível para acionamento de uma saída.

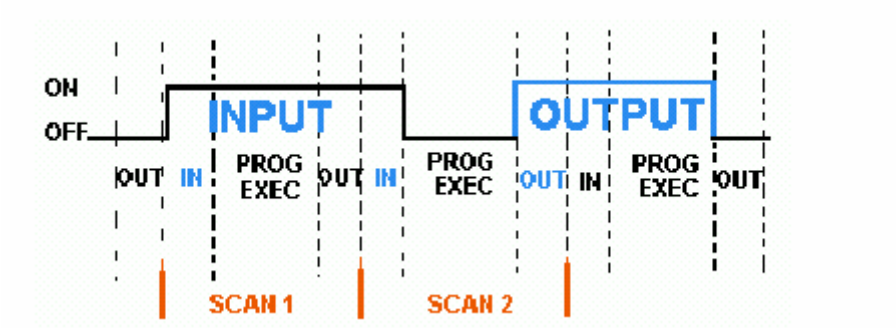

Figura 5.4 – Maior tempo de atraso possível para acionamento de uma saída.

6 CONTROLADOR LÓGICO PROGRAMÁVEL – CLP BASEADO EM LÓGICA PROGRAMÁVEL ESTRUTURADA

6.1 Arquitetura Proposta

A nova arquitetura proposta substitui a CPU da arquitetura tradicional do CLP, baseada em microcontrolador e periféricos ASICs, por um FPGA da família FLEX 10K. A figura 6.1 apresenta esta arquitetura. O FPGA possibilita definir vários blocos de hardware que operam em paralelo, aumentando a capacidade computacional do CLP. A família de dispositivos FLEX 10K da Altera possibilita uma completa integração de sistema em um único *chip*. Através de blocos lógicos estruturados, esta família de dispositivo acentua o desempenho dos dispositivos lógicos

programáveis (*PLDs*) que a antecederam, possibilitando flexibilidade de projeto e eficiência para aplicações de alto desempenho [5].

A arquitetura FLEX10K utiliza tecnologia de programação SRAM, os dados de configuração devem ser recarregados toda a vez que o dispositivo for inicializado ou quando novos dados de configuração forem necessários. O processo de se carregar fisicamente a memória SRAM com o algoritmo de controle é chamado de configuração.

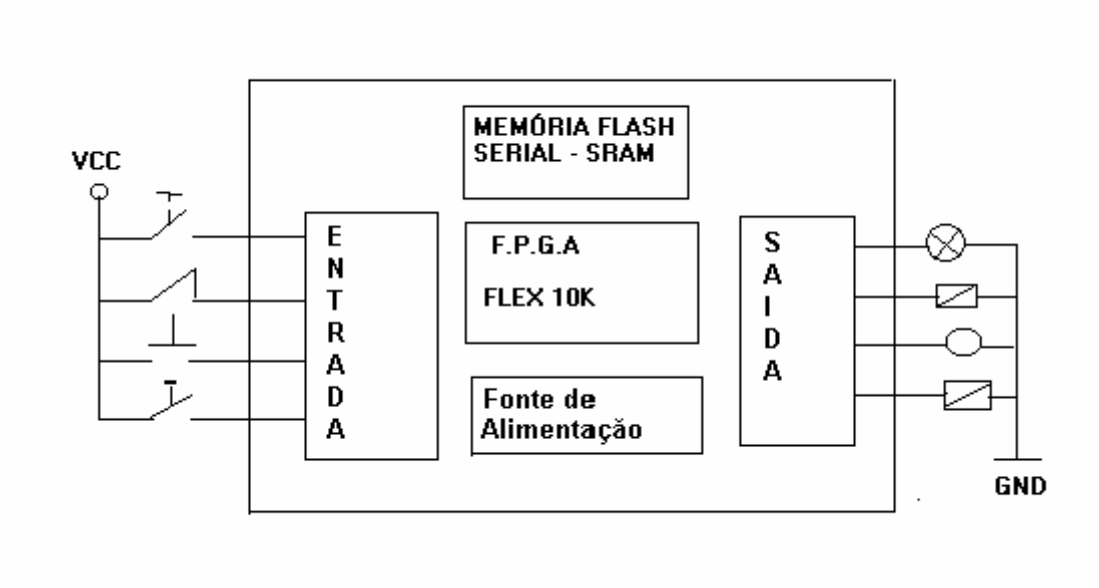

Figura 6.1 – Arquitetura de um CLP baseado em lógica programável estruturada.

6.2 Funcionamento do Controlador

O FPGA consiste de um grande arranjo de células lógicas ou blocos lógicos contidos em um único chip. Cada uma dessas células lógicas contém uma capacidade computacional para implementar funções lógicas e realizar roteamento para permitir a comunicação entre células [2]. O programa de controle do usuário (algoritmo de controle) é transformado em funções lógicas. Essas funções lógicas são implementadas nas células do FPGA, gerando um conjunto de células lógicas que são mapeadas e posicionadas numa localização específica do *chip*. Algoritmos típicos de posicionamento e mapeamento minimizam o número total de células, reduzem o caminho crítico entre elas, otimizando possíveis atrasos de sinais (*delay*) e minimizando o comprimento total das interconexões necessárias]2].

Desta forma o algoritmo de controle em forma de funções lógicas é implementado por hardware no FPGA, eliminando os ciclos de varredura (*scan*), ciclos de busca e execução de instruções, perda de sinais rápidos de entrada, tempos de atrasos de acionamento de saídas e etc que ocorriam no CLP de arquitetura tradicional baseada em microcontrolador.

Nesta nova arquitetura, o controlador utiliza todo o tempo na execução da tarefa (algoritmo de controle), não existe perda de tempo em outras tarefas não relacionadas ao processo.

6.3 Ferramentas de auxílio à programação

A programação do FPGA (configuração) é um dos processos mais importante ao se trabalhar com o novo controlador. Durante a programação é que será definido o algoritmo de controle e as suas funções lógicas correspondentes [7].

A programação consiste em carregar o controlador com o programa de controle em linguagem esquemática (portas lógicas) ou linguagem de descrição de hardware (*HDL, VHDL* ou *VERILOG*), como apresentada na seção 4. A maneira mais usada de se carregar esse programa de configuração no controlador é pela porta de comunicação serial (*COM*) ou paralela *(LPT*) de um microcomputador

PC. Para que cada célula lógica e suas interconexões sejam programadas são necessários algumas centenas de bits. Cada bit de configuração define o estado de uma célula de memória *SRAM* que, ou controla um bit de função LUT (*Look-Up Table*), ou seleciona uma entrada de um *multiplexador* ou define o estado de uma chave de comutação no FPGA.

Após a programação o controlador pode ser testado e o algoritmo de controle simulado. Uma característica interessante do controlador é que ele permite uma reconfiguração no próprio hardware. Ou seja, o usuário pode programar um novo hardware, como por exemplo uma nova porta de comunicação, um núcleo processador *core*, uma U.L.A (Unidade Lógica Aritmética),etc. Isso agiliza o processo de teste e possíveis reconfigurações com otimização do algoritmo de controle.

Cada fabricante disponibiliza programas que suportam seus próprios FPGAs. O software de programação e desenvolvimento que foi utilizado nesse controlador foi o QUARTUS II da empresa ALTERA. Ele permite soluções com lógica programável estruturada envolvendo várias etapas de projeto, num ambiente baseado em ferramenta EDA (*Eletronic Design Automation*). Possibilita a síntese lógica, arquivo *Netlist*, mapeamento, posicionamento, roteamento, simulação, testes, análise de funcionalidade, análise de temporização (clock), análise de consumo de energia e programação do dispositivo.

7 RESULTADOS OBTIDOS

No presente momento, além do estudo e discussão do modelo aqui descrito foram realizadas tarefas na implementação de um protótipo do controlador, a partir de um kit de desenvolvimento FPT-1 baseado na família FLEX 10K10, dispositivo EPC10K10TC144-4, com 10.000 portas lógicas, 576 células lógicas e 6.144 bits de memória RAM, que tem por objetivo validar a abordagem proposta.

Inicialmente encontramos uma resposta substancial, ainda que incompleta, com algumas contribuições no projeto e desenvolvimento de controladores lógicos programáveis para aplicações industriais. Seguem alguns resultados obtidos que irão contribuir na eficácia em termos de custos e desempenho dos controladores lógicos programáveis – CLPs:

- A partir da arquitetura baseada em lógica programável estruturada possibilidade de eliminação do ciclo de varredura (scan), dos ciclos de busca e execução de instruções executados pela CPU do CLP tradicional, dos atrasos para acionamento de saídas e da perda de entradas variáveis de curta duração (gerados por problemas de ciclo de scan);
- Possibilidade do aumento considerável na velocidade de processamento;
- Possibilidade na redução do consumo de energia elétrica;
- Implementação de diversos algoritmos de controle por hardware, funcionando em paralelo, sem execução de softwares em CPU;
- Possibilidade de redução de custos pela eliminação de componentes de hardware como microcontroladores e dispositivos ASICs, bem como redução no tamanho de placas de circuito impresso;

8 CONCLUSÕES

O controlador proposto neste trabalho traz uma série de características não encontradas em outros controladores similares academicamente ou comercialmente disponíveis. A simplicidade do controlador (hardware e software) também deve ser ressaltada. Se essa nova tecnologia, lógica programável estruturada, realmente tomar parte no mercado nacional de automação industrial, o projeto de controlador proposto neste trabalho estará pronta para ser utilizado.

9 AGRADECIMENTOS

Esta pesquisa está sendo apoiada pela empresa MINIPA Industria e Comercio Ltda na forma de bolsa de pesquisa.

10 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Wakerly, J. F.; *Digital Design – Principles & Practices*, Prentice Hall, ISBNO-13-769191-2, 3th Edition, New Jersey, Estados Unidos, 2000.

[2] Milani, G.C.A.; *SPmm1 Uma Máquina Paralela Reconfigurável*; Dissertação de Mestrado, UFSCAR, São Carlos,1998.

[3] Kugler, M.; Junior, T. J. ; Lopes, S. H. ; *Desenvolvimento de uma Rede Neural LVQ em Linguagem VHDL para Aplicação em Tempo Real;* VI Congresso Brasileiro de Redes Neurais, pp 103-108, 2 a 5 de junho, 2003, Centro Universitário da FEI, São Paulo, SP, Brasil.

[4] Gonsales, A.; Carro, L.; Lubaszewski, M.; Suzin, A.; *Projeto de um PLD com Características de Testabilidade,* GME – Grupo de Microeletrônica, UFRG – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS.

[5] Teixeira,M. A., *Técnicas de Reconfigurabilidade dos FPGAs da Família APEX 20K Altera.* Dissertação de mestrado, USP, São Carlos, 2002. Disponível por www em http//: www.teses.usp.br/teses/disponiveis/55/55134/tde-11092002-164901/publico/texto.pdf .

[6] Aragão, A.C.O.S*; Uma Arquitetura Sistólica para solução de Sistemas Lineares* Implementada *com Circuitos FPGAs*; ICMC, USP; Dezembro de 1998.

[7] Zaghetto, A.; Prado, A. C.; Tavares, A., *Trabalho sobre FPGA*; Departamento de Engenharia Eletrônica e de Computação, UFRJ. Disponível por www em http//: www.gta.ufrj.br/grad/01_1/pld/hcpld.htm

[8] Altera Corp.;*FLEX 10K Embedded Programmable Logic Family*; May 1998; Data Sheet; in http:/ www.altera.com/literature/ds/flex.pdf ;

[9] Lopes, A., Almeida, F. , Neumann, J. , Pinheiro, V.; *Dispositivos Lógicos Programáveis-PLD,* Trabalho Disciplina Circuitos Integrados, CEFET-RJ, Rio de Janeiro, Março, 1999. Disponível por www em http//: www.cefetrio.hpg.ig.com.br/ciencia_e_educacao/8/CI/pld2/default.htm

[10] Ribeiro, H. C. ; Dispositivos Lógicos Programáveis, apostila, depto de eletrônica, Escola Federal de Engenharia – EFEI, Itajubá, MG.

[11] Brow, S., Vranesic, Z. ; *Fundamentals of Digital Logic with VHDL, Design;* Mc Graw-Hill Series in Computer Enginnering, 2000;

[12] Mesquita, D. ; *Contribuições para Reconfigurações Parcial, Remota e Dinâmica de FPGAs.* Dissertação de Mestrado, PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Pós Graduação em Ciência da Computação, Porto Alegre, RS, Março 2002. Disponível em www por http://www.inf.pucrs.br/~dmesquita.

[13] Rose, J.; Gamal, A. E. ; Sangiovanni, A; *Architecture of Field Programmable Gate Arrays*; In proceedings of the IEEE; vol. 81, no. 7, pp. 1013-1029; Julho de 1993.

[14] Jasink, R.P. ; *Introdução a Linguagem VHDL*; Relatório Técnico, Laboratório de Microeletônica, CPDTT, CEFET-PR, Paraná, Curitiba, 2001, 47p.

[15] Brow, S.; *Routing Algorithms and Architectures for Field Programmable Gate Arrays*; Ph.D. Thesis, Dept. of Electrical Engineering, University of Toronto, February, 1992.
[16] Perry, D.L., ; *VHDL;* 3.a Edição, Mcgraw-Hill, New York, pages 1-16, 1998.

[17] Coffman, K.; *Real World FPGA design with Verilog*, Prentice Hall PTR. ISBN 0-13-099851-6, New Jersey, Estados Unidos, 1999.

[18] Moraes, F.G. ; Calazans, N.L.V. ; Ferreira, E.H. ; Liedke, D.C. , *Implementação Eficiente de uma Arquitetura Load/Store em VHDL,* Faculdade de Informática. PUCRS-RS.

[20] Casillo, A. L.; *VHDL – VHSIC Hardware Description Language,* apostila, disciplina Organização e Arquitetura de Computadores I, UFRN – Universidade Federal do Rio Grande do Norte, Fev, 2003. Disponível por www em http//: www.dimap.ufrn.br/~ivan/orgl/vhdl.pdf

[21] Ashenden, J. P. ; *The VHDL Cookbook*, first edition, Depto Computer Science, University of Adelaide South Austrália, July, 1990

[22] Pupo, S. M..; *Interface Homem Máquina para Supervisão de um CLP em Controle de Processo Através da WWW*; Tese de Mestrado, USP, Escola de Engenharia de São Paulo, Departamento de Engenharia Elétrica, 2002, São Carlos, SP.

[24] CORETTI,J. A. ; Manual de Treinamento Básico de Controlador Lógico Programável, Centro de Treinamento SMAR, Sertãozinho, SP, 1998.

11 DADOS DO AUTOR

César da Costa

UNITAU – Universidade de Taubaté
Departamento de Engenharia Mecânica
Pós-Graduação em Automação Industrial

MINIPA Industria e Comércio Ltda
Alameda dos Tupinás, 33
Planalto Paulista
04069-000 – São Paulo – SP
Tel.: (11) 5078-1876
Fax: (11) 577- 4561
Email: cost036@attglobal.net