UNIVERSIDADE DE TAUBATÉ – UNITAU

CESAR DA COSTA

PROPOSTA DE CONTROLADOR BASEADO EM LÓGICA PROGRAMÁVEL ESTRUTURADA

TAUBATÉ - SP

2005

UNIVERSIDADE DE TAUBATÉ – UNITAU

CESAR DA COSTA

PROPOSTA DE CONTROLADOR BASEADO EM LÓGICA PROGRAMÁVEL ESTRUTURADA

Dissertação apresentada para obtenção do Título de Mestre pelo Curso de Pós-Graduação do Departamento de Engenharia Mecânica da Universidade de Taubaté.
Área de Concentração: Automação e Controle Industrial.

Orientador: Prof. Dr. Francisco Parquet Bizarria

TAUBATÉ - SP

2005

COSTA, César da.
Proposta de controlador baseado em lógica
C837p programável estruturada./ César da Costa. – Taubaté: Unitau, 2005.

173 f. :il;30 cm.

Dissertação (Mestrado) – Universidade de Taubaté. Faculdade de Engenharia Mecânica. 2005
Orientador: Prof. Dr. Francisco Carlos Parquet Bizarria.

1. Hardware reconfigurável. 2. Lógica programável Estruturada. 3. Controlador – Mestrado. I. Universidade de Taubaté. Departamento de Engenharia Mecânica. II. Título.

CESAR DA COSTA

PROPOSTA DE CONTROLADOR BASEADO EM LÓGICA PROGRAMÁVEL ESTRUTURADA

Dissertação apresentada para obtenção do Título de Mestre pelo Curso de Pós-Graduação do Departamento de Engenharia Mecânica da Universidade de Taubaté.
Área de Concentração: Automação e Controle Industrial.

Data:_____

Resultado:_____

BANCA EXAMINADORA

Prof. Dr. Francisco Carlos Parquet Bizarria Universidade de Taubaté

Assinatura _____

Prof. Dr. Mauro Hugo Mathias FE/G UNESP

Assinatura _____

Prof. Dr. Álvaro Luiz Fazenda Universidade de Taubaté

Assinatura _____

Dedico este trabalho a minha esposa Elô e meus filhos Marcela e Caio, pela paciência infinda, compreensão, incentivo, energia e confiança, sem os quais o desafio de escrever este trabalho, não seria por mim superado.

AGRADECIMENTOS

Este trabalho estaria incompleto sem a menção a pessoas, cujo apoio e incentivo influenciaram-me de vários modos durante a sua elaboração.

Ao professor Francisco Borges, diretor acadêmico do IBTA, por colocar à disposição a estrutura de laboratórios de automação industrial do IBTA para os testes.

Ao Professor Dr. Francisco Carlos Parquet Bizarria pela orientação e apoio fornecido durante a elaboração deste trabalho.

Ao Sr. Joon Park, diretor Operacional da MINIPA Indústria e Comércio Ltda, pelo apoio na forma de bolsa de pesquisa.

Ao Prof. Dr. Luiz Octávio Mattos dos Reis da Engenharia elétrica da UNITAU, pelas sugestões e discussões iniciais que me ajudaram na elaboração desta dissertação.

A Professora Maria do Carmo Carrasco do IBTA, pela contribuição e sugestões na revisão ortográfica, gramatical e apresentação desta dissertação.

Aos meus pais *in memoriam* pelo amor e dedicação em minha formação.

Aos amigos e companheiros de mestrado que, de uma forma direta ou indireta, contribuíram para que este trabalho se concretizasse.

RESUMO

DA COSTA, C. (2005). **Proposta de controlador baseado em lógica programável estruturada**. 2005. 173p. Dissertação (Mestrado) – Departamento de Engenharia Mecânica, Universidade de Taubaté, Taubaté.

Este trabalho apresenta uma proposta para a implementação de Controlador Programável de uso geral, o qual utiliza na sua arquitetura dispositivos baseados em Lógica Programável Estruturada e macroinstruções para sua programação. A utilização de Lógica Programável Estruturada visa integrar em um único dispositivo as funções do microcontrolador e dos circuitos integrados de aplicações específicas ASICs (*Application Specific Integrated Circuits*), tipicamente utilizados em arquiteturas de Controladores Lógicos Programáveis (CLPs) tradicionais. Para validar a proposta apresentada neste trabalho, foi implementado um protótipo capaz de executar as macroinstruções gráficas desenvolvidas, bem como testar a possibilidade de reconfiguração do *hardware* utilizado. Os resultados satisfatórios obtidos nos ensaios práticos realizados com esse protótipo, associados com as metodologias empregadas no desenvolvimento das macroinstruções, mostram que essa proposta é viável e consistente, principalmente no que concerne a implementação de um sistema que equivale a um CLP tradicional, no qual foi possível integrar em um único dispositivo de Lógica Programável Estruturada, as funções do microcontrolador e dos seus circuitos integrados de aplicações específicas, reduzindo as dimensões do sistema, o consumo de energia elétrica e os tempos de processamento. Outra vantagem obtida foi a possibilidade de reconfiguração do *hardware* pelo próprio usuário.

Palavras-chave: *Hardware* reconfigurável e lógica programável estruturada.

ABSTRACT

DA COSTA, C. (2005). **Proposal of Controller based on Programmable Structured Logic**. 2005. 173p. Thesis (Master), – Department of Mechanical Engineering, Universidade de Taubaté, Taubaté.

This research presents a proposal of developing a Programmable Logic Controller (PLC) which uses Programmable Structured Logic in its architecture and macro instructions for its programming which will be used by a national enterprise in the instrumentation field as an aid in CLP training. The purpose of using Programmable Structure Logic in this research is to integrate the function of the micro-controller and of the Application Specific Integrated Circuits (ASICs) in a single device, typically used in traditional PLC. To validate the proposal presented in this research a prototype was built to execute the graphic macro instructions developed and to try out the reconfigurable *hardware* possibility. The satisfactory results obtained in the practical experiment accomplished with that prototype joined with *software* tools EDA (Eletronic Design Automation) and its methodology *hardware* development, show that the proposal is feasible and consistent, mainly as regards the *hardware* reconfiguration by end users, improving the system dimensions, minimizing the execution of the CPU operation cycles, rising the processing speed and reducing the use of power.

Key-words: Reconfigurable *hardware* and Programmable Structured Logic.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AHDL	*Altera Hardware Description Language*
ARC	*Automation Research Corporation*
ASIC	*Application Specific Integrated Circuit*
CAD	*Computer Aided Design*
CAM	*Content Addressable Memory*
CCM	*Custom Computing Machine*
CLB	*Configurable Logic Block*
CMOS	*Complementary Metal Oxide Semiconductor*
CPLD	*Complex Programmable Logic Device*
CPU	*Central Processing Unit*
CUPL	*Cornell University Programming Language*
DSP	*Digital Signal Processing*
EAB	*Embedded Array Block*
EDA	*Electronic Design Automation*
EIA	*Electronic Industries Association*
EEPROM	*Electrically Erasable Programmable Read Only Memory*
EPLD	*Erasable Programmable Logic Device*
EPROM	*Erasable Programmable Read Only Memory*
ESB	*Embedded System Block*
FPGA	*Field Programmable Gate Array*
GAL	*Gate Array Logic*

HCPLD	*High Complex Programmable Logic Device*
HDL	*Hardware Description Language*
ICR	*In Circuit Reconfigurable*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronic Engineers*
I/O	*Input/Output*
IOB	*Input/Output Block*
ISA	*Industry Standart Architecture*
ISP	*In System Programmability*
JTAG	*Joint Test Action Group*
LAB	*Logic Array Block*
LCA	*Logic Cell Array*
LE	*Logic Element*
LPM	*Library of Parameterized Modules*
LUT	*Look Up Table*
MOS	*Metal Oxide Semiconductor*
MPGA	*Mask Programmable Gate Array*
PAL	*Programmable Array Logic*
PCI	*Peripheral Component Interconnect*
PLA	*Programmable Logic Array*
PLD	*Programmable Logic Device*
PROM	*Programmable Read Only Memory*
RAM	*Random Acess Memory*

SMD	*Surface Mount Device*
SOPC	*System On a Programmable Chip*
SPLD	*Simple Programmable Logic Device*
SRAM	*Static Random Acess Memory*
STAPL	*Standard Test and Programming Language*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VLSI	*Very Large Scale Integration*

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO

Este capítulo é constituído pela descrição do problema, pela solução proposta, pelos objetivos do trabalho, por uma revisão bibliográfica que busca retratar o estado da arte em controladores baseados em Lógica Programável Estruturada e pela exposição da estrutura do trabalho. Na descrição do problema, é apresentada a atual forma de operação de uma arquitetura tradicional de Controlador Lógico Programável (CLP) e os principais limites gerados durante o seu ciclo de funcionamento. Na solução proposta, é apresentada uma nova abordagem de arquitetura para controladores, baseados em Lógica Programável Estruturada, foco desta dissertação. No item relativo aos objetivos são apresentados os benefícios que se deseja obter com o desempenho da nova arquitetura. Na parte reservada à revisão bibliográfica é abordado o estado da arte em tecnologia baseada em Lógica Programável Estruturada, disponível entre os fabricantes de CLPs. No item referente à estrutura do trabalho é mostrado de forma resumida o conteúdo de cada capítulo.

1.1 Descrição do Problema

Para os fabricantes de CLPs, a Unidade Central de Processamento (CPU)[1], formada por um microprocessador e um sistema de memória, é o principal componente da arquitetura interna do controlador. De um modo simplificado, o

[1] Em língua inglesa CPU, iniciais de *Central Processing Unit*.

seu ciclo de funcionamento utiliza a seguinte seqüência de atividades: A CPU lê as entradas, executa a lógica de controle segundo as instruções do programa de aplicação armazenado em sua memória, realiza cálculos e controla as saídas, respectivamente (ALLEN BRADLEY, 1996).

Durante o ciclo de funcionamento, o microprocessador do CLP busca as instruções armazenadas na sua memória e executa cada instrução. A execução do programa consiste na repetição seqüencial do processo de busca e execução das instruções. Pode-se justificar a divisão do processamento da instrução em dois principais estágios da seguinte forma: i) a busca da instrução é uma operação comum para cada instrução, consiste na leitura de uma localização de memória e o posterior carregamento do seu conteúdo no registro de instrução; ii) a execução da instrução pode envolver várias operações e depende da natureza da instrução (OLIVEIRA et al.,1983).

Tradicionalmente, a execução de algoritmos de controle com a utilização de CLPs convencionais, baseados em microprocessadores, embora apresentem alta flexibilidade para modificações, não são executados com a mesma velocidade dos algoritmos executados por *hardware* (COMPTON, 1999). O fato de os CLPs executarem de forma seqüencial suas tarefas faz com que os mesmos tenham tempos de processamento limites, que não são aceitáveis para muitas aplicações (MELO et al., 2004).

Como na CPU a busca da instrução é uma operação comum para cada instrução e consiste na leitura de uma localização de memória, essa atividade executada pelo CLP tradicional consome tempo de processamento, em uma tarefa

que não está diretamente relacionada com o controle de processo, no qual o controlador está inserido.

O tempo limite consumido pela CPU na execução do ciclo de busca das instruções em memória, juntamente com a possibilidade de reconfiguração de *hardware*, são exemplos de limites encontrados na arquitetura interna de um CLP tradicional, que serão abordados nesta dissertação.

1.2 Solução Proposta

Uma alternativa cada vez mais utilizada para o desenvolvimento de controladores dedicados, por exemplo, controladores de rede, controladores de vídeo, etc, com grande capacidade de processamento, é a implementação de algoritmos de controle diretamente em dispositivos baseados em Lógica Programável Estruturada. Esses dispositivos podem ser programados para executarem uma função específica (JASINSKI, 2001). Diferente dos microprocessadores que executam um *software* com instruções seqüenciais em uma arquitetura predefinida (von Neumann, Harvard, etc), os dispositivos baseados em Lógica Programável Estruturada tem sua arquitetura interna definida pelo usuário final, permitindo que a estrutura e a forma de funcionamento do *hardware* do dispositivo sejam particularizadas para uma determinada aplicação (KUGLER et al., 2003).

Uma das grandes vantagens da arquitetura baseada em dispositivo de Lógica Programável Estruturada é a possibilidade de se definir vários blocos de *hardware* que operam em paralelo, aumentando a capacidade computacional e a eficiência quando comparada com o processamento utilizado pelos controladores tradicionais. Além da característica de poder operar em paralelo, algoritmos implementados com Lógica Programável Estruturada são tipicamente mais rápidos que algoritmos implementados em microprocessador convencional (KUGLER et al., 2003).

O destaque dessa proposta de solução para a arquitetura interna de CLPs tradicionais é a utilização da Lógica Programável Estruturada, que possibilita a reconfiguração do *hardware* utilizado pelo usuário final e a otimização dos ciclos de funcionamento da CPU.

Será escolhido um ambiente para desenvolvimento, o qual deverá permitir a execução de simulação, testes ou reconfiguração do sistema de modo rápido, empregando-se novas metodologias de projeto de *hardware*, apoiadas em ferramentas de *software* EDA[2].

[2] A evolução da automação de projeto eletrônico, *Electronic Design Automation* (EDA), começou nos anos 70 com as ferramentas de projeto auxiliado por computador, C*omputer Aided design* (CAD), que davam assistência aos projetistas na geração de desenhos de circuitos.

1.3 Objetivos do Trabalho

O objetivo principal deste trabalho é propor a melhoria dos tempos de processamento da CPU de um CLP tradicional, substituindo-a por uma arquitetura baseada em Lógica Programável Estruturada. Para validar essa proposta, será implementado um protótipo de CLP com arquitetura baseada em Lógica Programável Estruturada, a qual deverá ser reconfigurável pelo usuário final, por meio de macroinstruções. Nesse protótipo, serão avaliadas as seguintes características:

- Possibilidade de integrar em um único dispositivo de Lógica Programável Estruturada as funções do microcontrolador e dos circuitos integrados de aplicações específicas (ASICs), reduzindo as dimensões do sistema.
- Capacidade de otimizar os ciclos de funcionamento da CPU, minimizando os tempos de processamento utilizados pelo microprocessador com gerenciamento de instruções, teste de condições e busca de instruções na memória.
- Aumentar a velocidade de processamento.
- Reduzir o consumo de energia elétrica.
- Possibilitar a reconfiguração do *hardware.*

1.4 Revisão Bibliográfica

Embora muitos fabricantes tenham dedicado extenso esforço à aplicação de Lógica Programável Estruturada na arquitetura de seus controladores, a sua aplicação tem sido restrita à prototipação de circuitos de aplicação específica (ASICs), que são utilizados como coprocessadores nas arquiteturas tradicionais de CLPs. Pode-se citar, como exemplo desse tipo de aplicação, a família de inversores de freqüência Ultra5000 e Ultra 3000i da Allen Bradley (ALLEN BRADLEY, 2003).

O estado da arte em controladores baseados em Lógica Programável Estruturada encontra-se em aplicações de sistemas de aquisição e controle industrial, com um novo conceito de CLP reconfigurável por *softwar*e. A proposta consiste em o usuário final programar o algoritmo de controle, definir as entradas e saídas de *hardware* diretamente no dispositivo, a partir de um *software* gráfico e, então, o CLP executar o programa de controle (BABB, 2004).

A primeira empresa a lançar uma família de CLPs com arquitetura baseada em Lógica Programável Estruturada e *software* de reconfiguração gráfico foi a National Instruments (BABB, 2004). A família de controladores é denominada de LabVIEW FPGA, sendo constituída por bastidores chamados CompactRIO e placas de expansão PCI/PXI para microcomputadores padrão IBM PC. A figura 1 mostra um exemplo dessa placa com a respectiva tela do *software* gráfico de reconfiguração intitulado LabVIEW FPGA.

Essa nova geração de CLPs, com arquitetura baseada em *hardware* reconfigurável, utiliza processamento paralelo. Segundo seu fabricante, permite executar aplicações determinísticas, mais rápidas que um CLP tradicional, bem como fazer aquisição de dados na ordem de vinte e cinco nanosegundos de tempo de resolução e executar controle de malhas com variáveis analógicas, com milhares de amostragem por segundo (NATIONAL, 2004).

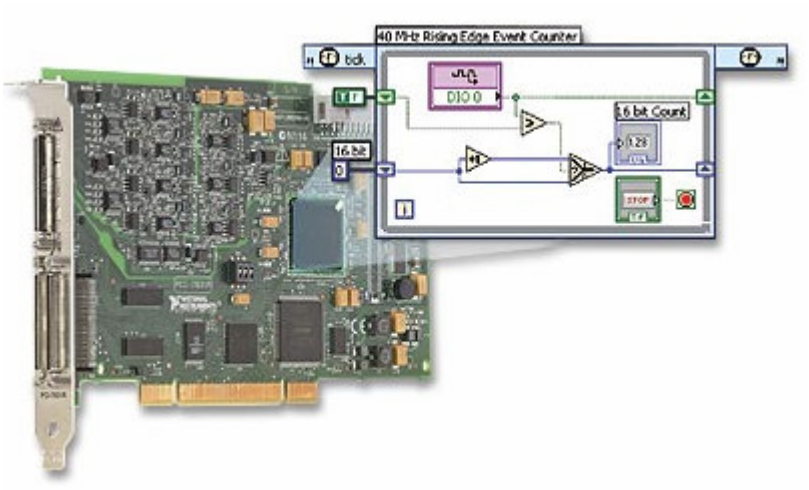

Figura 1 - Placa de expansão PCI 7831

O CLP CompactRIO é um sistema de aquisição de dados e controle industrial que utiliza tecnologia baseada em Lógica Programável Estruturada reconfigurável. Basicamente é constituído por um bastidor de dois, quatro ou oito ranhuras, módulo processador de tempo real baseado no microprocessador Pentium IV, módulos de entradas e módulos de saídas reconfiguráveis. A figura 2 apresenta um diagrama de blocos da arquitetura do sistema abordado. O

dispositivo FPGA é conectado a cada módulo de entrada e saída em topologia[3] estrela, de modo que possa controlá-los. Cada módulo possui condicionadores de sinais para conexão direta com sensores e atuadores. Um barramento[4] local PCI de alto desempenho fornece a conexão entre o FPGA e o módulo processador de tempo real (processador Pentium IV) (NATIONAL. 2004).

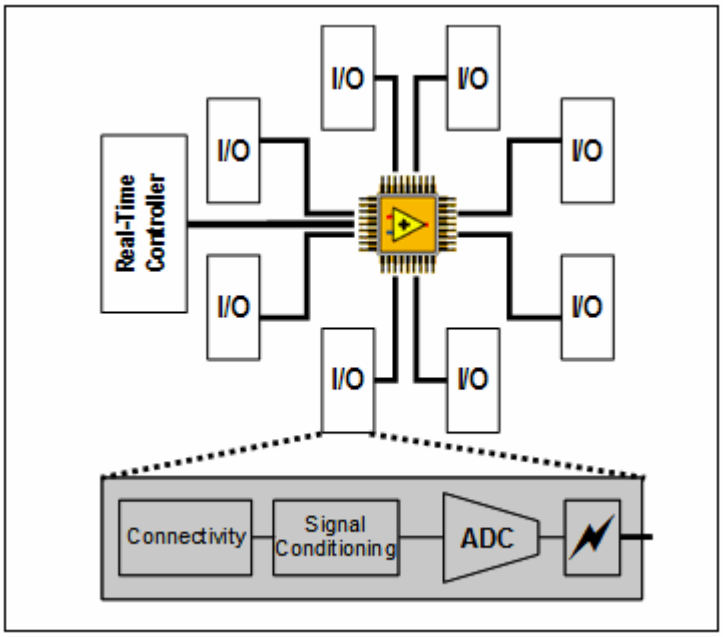

Figura 2 - Diagrama em blocos do controlador CompactRIO

O *software* gráfico LabVIEW FPGA permite a programação, através de símbolos gráficos, do algoritmo de controle da aplicação (funções lógicas, malhas com variáveis analógicas, contador, temporizador, comparação, etc). Ao

[3] Nessa topologia, os elementos são conectados a um elemento central, o qual cria uma topologia lógica interna, em anel ou barramento.
[4] Barramento PCI (*Peripheral Component Interconect*) foi desenvolvido pela Intel, na época do desenvolvimento do Pentium, no início dos anos 90, o objetivo era obter um padrão de barramento que substituísse o barramento ISA (Industry Standart Architecture) com maiores taxas de transferência de dados.

completar-se a programação, o *software* LabVIEW FPGA faz a síntese do código gerado e o transfere (*download*) para o dispositivo de Lógica Programável Estruturada (NATIONAL, 2004). O sistema pode ter uma das seguintes configurações:

- **Autônoma**: o dispositivo de Lógica Programável Estruturada inicia a aplicação independente do *software* operando no microcomputador padrão IBM PC (supervisório).

- **Combinada *off-line***: o dispositivo de Lógica Programável Estruturada inicia a aplicação em conjunto com o *software* operando no microcomputador, para onde os dados são transferidos periodicamente para posterior análise (arquivo de dados, gráfico de tendência, transferência para Interface Homem-Máquina, etc).

- **Combinada *on-line***: o dispositivo de Lógica Programável Estruturada inicia a aplicação em conjunto com o *software* operando em tempo real no microcomputador, para onde os dados são transferidos e analisados em tempo real.

1.4.1 Controlador PAC (*Programmable Automation Controller*)

A entidade americana ARC (*Automation Research Corporation*) criou a sigla PAC (*Programmable Automation Controller*) e o utiliza para descrever a nova geração de CLPs industriais, que combinam as funcionalidades de um CLP convencional e um microcomputador padrão IBM PC (VOORHORST, 2004). A

ARC identifica cinco características principais em um controlador PAC (TOOD, 2004):

- **Múltipla funcionalidade**: controle lógico, controle analógico, intertravamento, acionamento de motor, controle de processo e IHM (Interface Homem-Máquina) em uma única plataforma.
- **Plataforma de desenvolvimento única e multidisciplinar**: etiquetas (t*ags*) e base de dados única para todos os parâmetros;
- **Ferramentas de *software***: programas que utilizem a norma IEC 61131-3, e permitam desenvolvimento por fluxos de processo através de muitas máquinas ou unidades de processo.
- **Arquitetura modular e aberta**: aplicações industriais envolvendo organização de máquinas em fábrica e unidades de operação em plantas de processo.
- **Utilização de padrões**: interfaces de rede, linguagens, protocolos, etc.

A sigla PAC é utilizada tanto por fabricantes tradicionais de CLPs para descreverem seus últimos lançamentos de alta tecnologia como, por exemplo, ControlLogix da empresa Rockwell Automation e PACSystems da empresa GE Fanuc Automation, como por fabricantes de controladores baseados em Lógica Programável Estruturada como, por exemplo, o CLP CompactRIO da empresa National Instruments.

Os controladores PACs representam o início da mudança dos CLPs convencionais, que por quase três décadas têm proporcionado aos usuários finais e

fabricantes de equipamentos, métodos de controle eficientes e confiáveis. Contudo, os engenheiros de automação que trabalham com CLPs convencionais em campo têm passado grande parte do seu tempo planejando a atualização desses controladores instalados para conseguirem um melhor desempenho e melhorarem a sua capacidade de se comunicarem com os outros níveis computacionais da fábrica: i) nível II de Supervisão e Controle (engenharia e processos); ii) nível III de Administração (gerências de produção) (GE FANUC, 2004).

Os CLPs tradicionais, por possuírem arquitetura de controle rígida, não permitem que o usuário final faça uma reconfiguração de seu *hardware* (*upgrade*) por *software;* torna-se necessária a substituição definitiva do equipamento (*hardware e software*) por outro mais moderno com mais recursos de comunicação, protocolos, etc (GE FANUC, 2004).

Atualmente, muitas empresas têm procurado integrar em rede os CLPs convencionais instalados em suas máquinas, conectando-os em redes locais aos sistemas coorporativos da empresa, rumo ao CIM[5] (COSTA, 1990). Na sua grande maioria, os CLPs convencionais não permitem esse nível de integração em uma única plataforma. (GE FANUC, 2004). Sendo necessárias várias plataformas de *hardware* e *software*, constituídas de microcomputadores padrão IBM PC e CLPs.

Os controladores PACs apresentam integrados em uma única plataforma módulos processadores em tempo real e módulos de entradas e saídas reconfiguráveis, que permitem a integração de diversas aplicações de automação

[5] Em língua inglesa: CIM, iniciais de *Computer Integrated Manufacturing.*

industrial (supervisão, controle, redes locais, IHM, intertravamento, etc) em um único equipamento (TOOD, 2004).

1.5 Estrutura do Trabalho

O **capítulo 1** é composto pela descrição do problema, pela solução proposta, pelos objetivos do trabalho, por uma revisão bibliográfica sobre a aplicação de Lógica Programável Estruturada em controladores de alta tecnologia e pela descrição da estrutura do trabalho. Na caracterização do problema, é apresentada, de maneira resumida, a atual forma de operação de uma arquitetura de CLP tradicional e os principais limites gerados durante o seu ciclo de funcionamento. Quanto à solução proposta, é apresentada uma arquitetura de controladores baseada em Lógica Programável Estruturada, que será abordada nesta dissertação. No item relativo aos objetivos, são apresentados os benefícios que se deseja obter com o desempenho da arquitetura proposta. Na revisão bibliográfica, é abordado o estado da arte em tecnologia baseada em Lógica Programável Estruturada, disponível entre os principais fabricantes de CLPs. No item estrutura do trabalho, é mostrado de forma resumida o conteúdo de cada capítulo.

No **capítulo 2,** tem-se a descrição dos conceitos básicos sobre Controladores Lógicos Programáveis (CLPs), Lógica Programável Estruturada, Ambiente e Ferramentas de *Software* EDA, Linguagem de Programação VHDL, Simulação e Síntese. Esse capítulo tem por objetivo principal apresentar os conceitos básicos para o leitor compreender os termos e conceitos utilizados neste trabalho.

No **capítulo 3,** é relatado o desenvolvimento do trabalho, no qual são propostas soluções para os limites de tempos gerados durante o ciclo de funcionamento do CLP convencional e a reconfiguração do seu *hardware*. Apresenta uma nova arquitetura de CLP, baseada em Lógica Programável Estruturada, descrevendo o *hardware* dessa nova arquitetura, o seu princípio de funcionamento, o ambiente de programação e desenvolvimento da biblioteca de macroinstruções gráficas intitulada por PLCPROJECT, e o seu conjunto básico de instruções. Também são mostrados os meios de comunicação para programação remota do CLP proposto, bem como a descrição do *Kit* FPT1 e do CLP protótipo utilizado para simulação e validação do bloco FPGA da arquitetura proposta.

O **capítulo 4** apresenta o ensaio prático dos componentes de *hardware* e *software* desenvolvido no estudo da nova arquitetura de CLP, incorporando-o à implementação de um protótipo. O objetivo dessa fase é analisar o comportamento de todo o sistema (*hardware* e *software*) para validar a arquitetura baseada em Lógica Programável Estruturada proposta.

No **capítulo 5** são mostradas as conclusões do trabalho e algumas indicações para trabalhos futuros.

A dissertação é encerrada com a apresentação de um apêndice, contendo a listagem de programas de aplicações de CLPs baseados em Lógica Programável Estruturada e CLPs tradicionais, que foram utilizadas para a realização dos ensaios práticos mencionados neste trabalho.

CAPÍTULO 2 - CONCEITOS BÁSICOS

Este capítulo tem como objetivo apresentar, através de uma revisão analítica, visando à aplicação realizada neste trabalho, os diversos conceitos relativos a Controladores Lógicos Programáveis (CLPs), Lógica Programável Estruturada, Ferramentas de *Software* EDA (*Electronic Design Automation*), Linguagem de Programação VHDL (*Very High Speed Integrated Circuit Hardware Description Language*), Simulação e Síntese Lógica. Esses conceitos formarão a base teórica necessária para o leitor compreender o conteúdo e as terminologias utilizadas neste trabalho.

2.1 Controlador Lógico Programável (CLP)

O desenvolvimento dos CLPs começou em 1968, em resposta a uma requisição de engenharia da Divisão Hidramática da General Motors (GM). Naquela época, os engenheiros de fábrica da GM freqüentemente passavam dias ou semanas alterando os sistemas de controle baseados em relés, sempre que ocorriam mudanças em um modelo de carro ou era introduzido modificações na linha de montagem. Para reduzir o alto custo de instalação decorrente dessas alterações, a especificação do controlador da GM necessitava de um sistema de estado sólido, com a flexibilidade de um computador, mas que pudesse ser programado e mantido pelos engenheiros e técnicos de fábrica (ALLEN BRADLEY, 1996). Também era preciso que suportasse o ar poluído, a vibração, o

ruído elétrico e os extremos de umidade e temperatura encontrados normalmente num ambiente industrial.

2.1.1 Descrição Típica de um CLP

O CLP tradicional pode ser definido como um dispositivo de estado sólido, computador industrial, capaz de armazenar instruções para implementação de funções de controle tais como seqüência lógica, temporização e contagem, além de realizar operações lógicas e aritméticas, manipulação de dados e comunicação em rede. (GEORGINI, 2003).

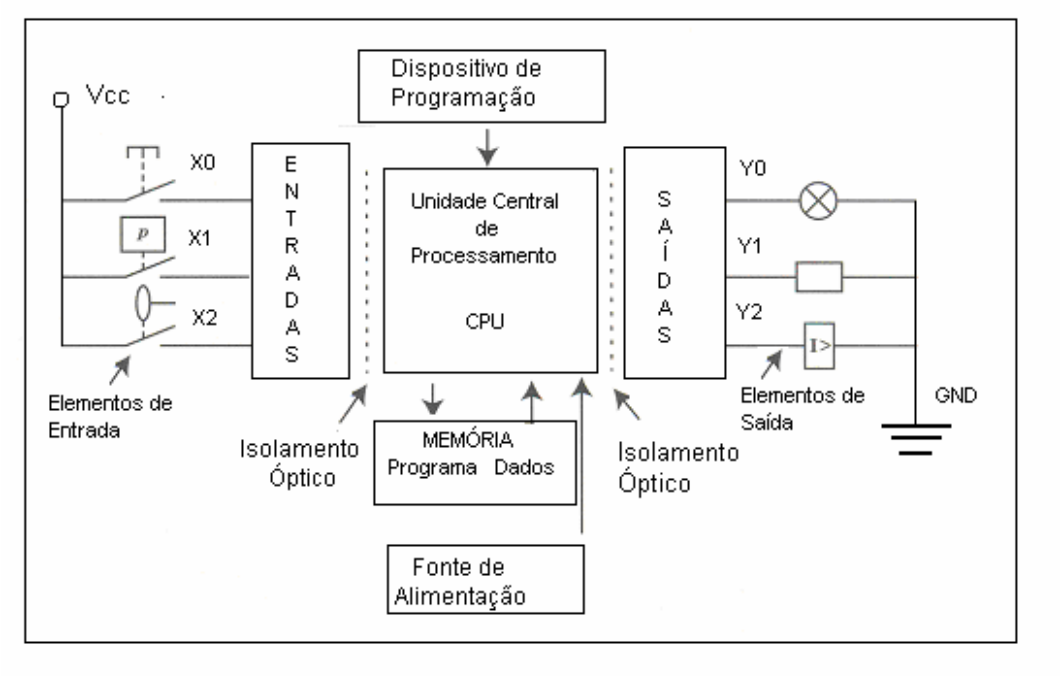

Figura 3 - Arquitetura básica de um CLP

Como apresentado na figura 3, a arquitetura é formado por uma fonte de alimentação, uma Unidade Central de Processamento (CPU) que compreende o processador, o sistema de memória ROM[6] e RAM[7] para armazenamento de programa e dados, e circuitos de entradas e saídas.

A fonte de alimentação é o componente responsável pelo fornecimento adequado de energia elétrica para a CPU e para os circuitos de entrada e saída. Os circuitos de entrada formam a interface pela qual os dispositivos enviam informações de campo para o CLP. As entradas podem ser digitais ou analógicas e são provenientes de elementos de campo como sensores, botões, pressostatos, chave fim-de-curso, etc. Os dispositivos de saída, tais como solenóides, relés, contatores, válvulas, luzes indicadoras e alarmes estão conectados aos circuitos de saída do CLP. As saídas de maneira similar às entradas podem ser digitais ou analógicas (CORETTI, 1998). As saídas e entradas são geralmente isoladas do campo por meio de isoladores galvânicos, como acopladores ópticos e relés.

O sistema de memórias é constituído tipicamente por memórias RAM e FLASH EEPROM[8]. O programa e os dados armazenados no sistema de memória são geralmente descritos utilizando-se os seguintes conceitos (MORAES et al., 2001):

---

[6] ROM (Read Only Memory) – Memória Somente de Leitura, contém o programa desenvolvido pelo fabricante do CLP, que determina como o sistema deve operar.

[7] RAM (Random Acess Memory) – Memória de Acesso Aleatório de Leitura e Escrita que armazena o programa de aplicação do usuário.

[8] A Memória FLASH EEPROM é semelhante a EEPROM (*Electric Erasable Programmable Read Only Memory*), porém as tensões de apagamento são baixas e o tempo de apagamento é pequeno, ao passo que a velocidade de gravação é rápida (*flash*).

- **Memória Residente**: contém os programas considerados parte integrante do sistema, permanentemente armazenados, que supervisionam e executam a seqüência de operações, as atividades de controle e comunicação com os dispositivos periféricos, bem como outras atividades.
- **Memória do Usuário**: armazena o programa aplicativo do usuário, ou seja, o programa de aplicação.
- **Memória de Dados ou Tabela de Dados**: nessa área são armazenados os dados associados com o programa de controle, tais como valores de temporizadores, contadores, constantes, etc.
- **Memória Imagem das Entradas e Saídas**: área que reproduz o estado de todos os dispositivos de entrada e saída conectados ao CLP.

A CPU do CLP tradicional é formada pelo microprocessador, um sistema de memória e circuitos ASICs de controle e comunicação. A CPU interpreta os sinais de entrada, executa a lógica de controle segundo as instruções do programa de aplicação, realiza cálculos, executa operações lógicas, para, em seguida, enviar os sinais apropriados às saídas (CORETTI, 1998).

Dependendo do fabricante, os componentes básicos citados anteriormente podem vir num único bloco, conhecido como CLP de estrutura fixa, ou em módulos separados, dispostos num mesmo bastidor interligado, conhecido como CLP modular (ALLEN BRADLEY, 1996). A figura 4 apresenta um CLP de estrutura fixa e um CLP modular.

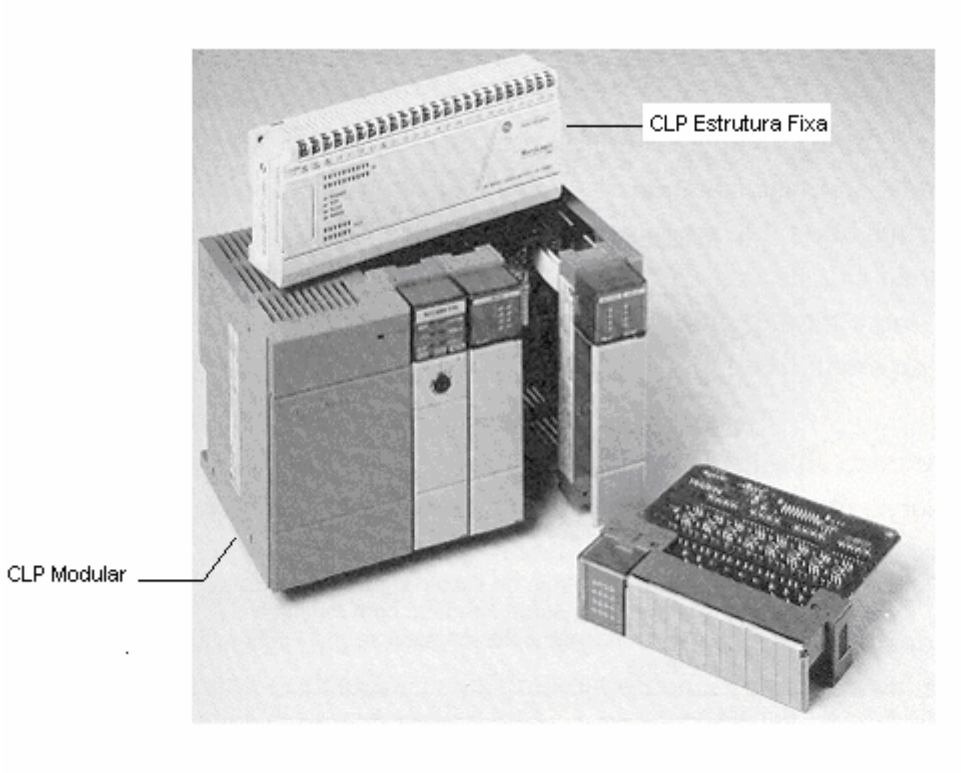

Figura 4 - Ilustração de CLP de estrutura fixa e um CLP modular

2.1.2 Programação

A programação de um CLP pode ser elaborada em várias linguagens de programação. A Organização Internacional IEC (*International Electrotechnical Committee*) é a responsável pela padronização das linguagens de programação para CLP, sendo a norma IEC 61131-3 *Programing Languages* a responsável pela classificação dessas linguagens. A tabela 1 ilustra a classificação das linguagens de programação conforme a norma IEC 61131-3 (MORAES et al., 2001).

TABELA 1 - Classificação das Linguagens de Programação

Classes	Linguagens
Tabulares	Tabela de Decisão
Textuais	IL (*Instruction List*) ST (*Structured Text*)
Gráficas	LD (Diagrama de Relés) FBD (*Function Block Diagram*) SFC (*Sequential Flow Chart*)

A forma de programação pode ser *off-line* (programação remota) ou *on-line* (programação local), através de teclados especiais, interfaces gráficas ou através de microcomputador padrão IBM PC. A programação é executada e posteriormente transferida para o CLP, via porta de comunicação *RS232C*[9] ou *RS485*[10], por exemplo.

2.1.2.1 Linguagem de Diagrama de Relés (*Ladder*)

Apesar das tentativas de padronização da norma IEC 61131-3, ainda não existe uma padronização rigorosa para programação em linguagem de diagramas de relés (*Ladder Diagram*), ou seja, a linguagem *Ladder* de um fabricante de CLP não funciona no CLP de outro fabricante (MORAES et al., 2001); o que existe é uma semelhança na representação gráfica dos diversos fabricantes, que representa esquematicamente o diagrama elétrico e é de fácil entendimento, tendo boa aceitação no mercado.

[9] A interface RS232C é uma interface de comunicação padronizada pela EIA (*Electronic Industries Association*), utilizada nas portas seriais dos computadores padrão IBM PC para comunicação serial com diversos periféricos a pequenas distâncias.

[10] A interface RS485 também é uma interface de comunicação e apresenta características que garantem a viabilidade de transmissão de dados seriais via cabo a grandes distâncias (1200 metros), sem detrimento da velocidade.

A linguagem de diagrama de relés (*Ladder*) é uma simbologia construída por linhas numa planilha gráfica, sendo que cada elemento é representado como uma célula (PUPO, 2002). Cada célula ou elemento gráfico é uma macro-instrução desenvolvida a partir de microinstruções do microprocessador. A figura 5 apresenta um painel de símbolos gráficos utilizados na programação *Ladder* pelo *software* FPWIN GR, configurador dos CLPs FP0, FP1 e FP-M, fabricados pela empresa japonesa Matsushita Electric Works (AROMAT, 2000).

Figura 5 - Painel de programação *Ladder* do *software FPWIN*

Um programa em linguagem de diagrama de relés assemelha-se bastante a um diagrama de contatos elétricos. Em um diagrama de contatos elétricos, os símbolos gráficos representam os dispositivos reais e a maneira como estão conectados (ALLEN BRADLEY, 1996). O programa em linguagem de diagrama de relés utiliza símbolos semelhantes; nesse caso os símbolos gráficos representam macroinstruções lógicas do programa de aplicação, armazenadas na memória do usuário. Não existe barra de alimentação nem o fluxo de corrente ao longo do programa. Outra diferença é que em um diagrama elétrico descrevem-se os dispositivos como abertos ou fechados (desenergizados ou energizados). No

programa em linguagem de diagrama de relés, as macroinstruções são verdadeiras ou falsas (ALLEN BRADLEY, 1996).

A figura 6 apresenta um trecho de um programa em linguagem de diagrama de relés. As macroinstruções mais freqüentemente usadas em um programa em linguagem *Ladder* são: a instrução “Normalmente Aberto”(N.A), a instrução “Normalmente Fechado”(N.F.) e a instrução “Energizar Saída”. Essas macroinstruções são representadas na forma de símbolos gráficos colocados nas linhas do programa, sendo por isso também conhecida como “simbologia de contatos de relés”.

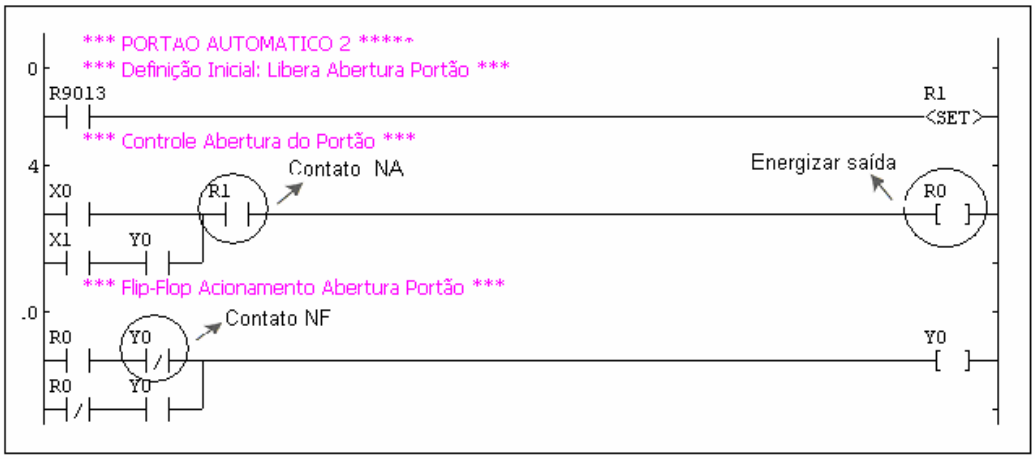

Figura 6 - Programa em linguagem de diagrama de relés

2.1.3 Princípio de Funcionamento de um CLP

O CLP funciona segundo um ciclo de varredura chamado *scan time*, que consiste de uma série de operações realizadas de forma seqüencial e repetida. A figura 7 ilustra, em forma de fluxograma, as principais fases do ciclo de varredura de um CLP (CORETTI, 1998). Os elementos principais de um ciclo de varredura são:

- **Atualização das entradas**: durante a varredura das entradas, o CLP examina os dispositivos externos de entrada quanto à presença ou à ausência de tensão, isto é, um estado "*energizado*" ou "*desenergizado*". O estado das entradas é atualizado e armazenado temporariamente em uma região da memória chamada "*tabela imagem das entradas*".
- **Execução do programa**: durante a execução do programa, o CLP examina as instruções do programa de controle (aplicação), usa o estado das entradas armazenadas na tabela imagem das entradas e determina se uma saída será ou não "*energizada*". O estado resultante das saídas é armazenado em uma região da memória chamada "*tabela imagem das saída"*.
- **Atualização 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.
- **Realização de diagnósticos**: ao final de cada ciclo de varredura a CPU verifica as condições do CLP, ou seja, se ocorreu alguma

falha em um dos seus componentes internos (fonte, módulos de entrada / saída, memória, etc).

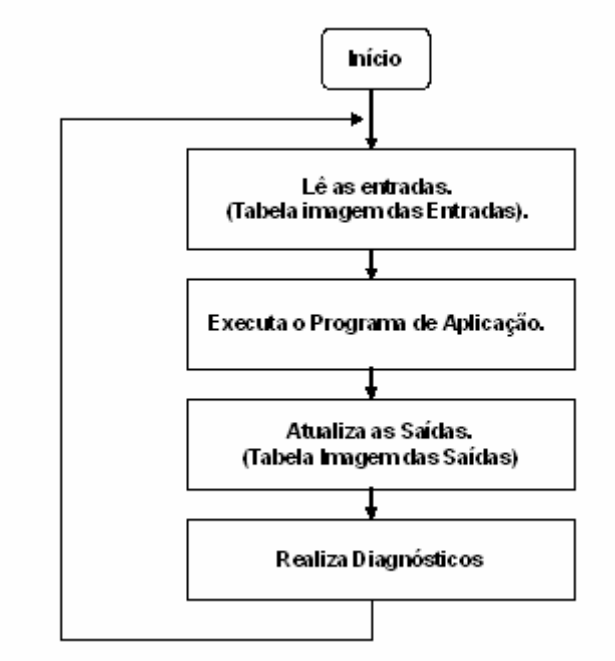

Figura 7 - Fluxograma básico de um ciclo de varredura

2.1.4 Ciclo de Trabalho da CPU

O CLP executa cada linha do programa de forma seqüencial[11], não volta atrás para executar a linha anterior, até que se faça a próxima varredura do programa. As linhas são normalmente ordenadas de forma a configurar uma seqüência de eventos, ou seja, a linha mais acima é o primeiro evento e, assim, sucessivamente. A figura 8 apresenta, por exemplo, uma seqüência de eventos em um programa em linguagem *Ladder*, que controla um estacionamento de veículos.

[11] O CLP não executa *loops* ou desvios como na programação tradicional. O seu processamento segue a seqüência do ciclo de varredura apresentado na figura 7, que não permite retroceder na seqüência de execução do programa.

Tanto nos diagramas elétricos como nos programas em linguagem *Ladder*, o estado das instruções de entrada (condição) de cada linha determina a seqüência em que as saídas são acionadas. (ALLEN BRADLEY, 1996).

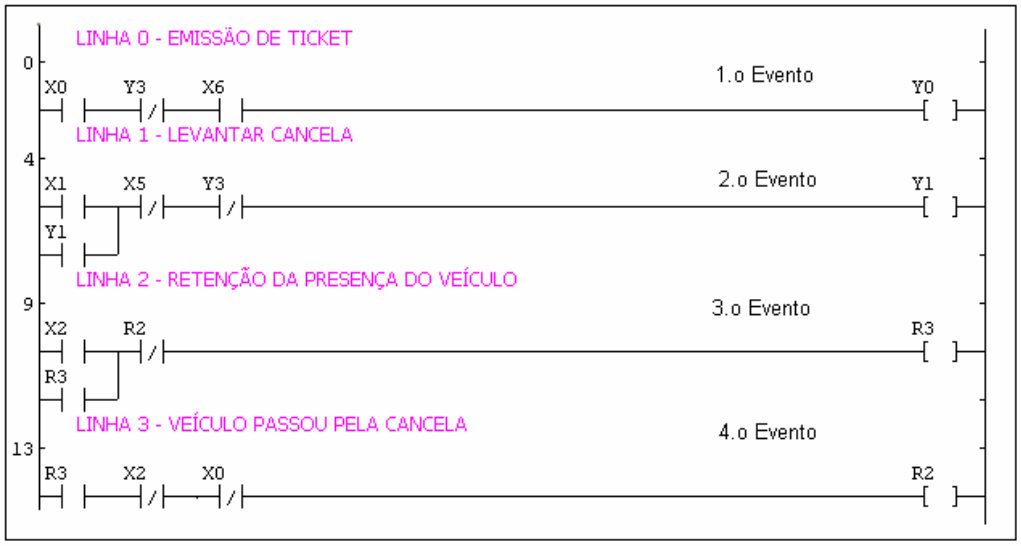

Figura 8 - Seqüência de eventos em programa *Ladder*

A tabela 2 ilustra os tempos de referência para execução de algumas instruções de linguagem *Ladder*, por exemplo, de um *CLP* fabricado pela empresa Allen Bradley. Segundo ALLEN BRADLEY (1996), o tempo em microssegundos, para executar uma instrução quando sua condição é verdadeira, é maior que o tempo necessário para executar a mesma instrução quando sua condição é falsa.

TABELA 2 - Tempos de referência de execução de instruções do CLP

Tipo de Instrução	Tempo de Execução (μs) (Instrução Falsa)	Tempo de Execução (μs) (Instrução Verdadeira)
Verificar se uma entrada está energizada	1,542	1,72
Verificar se uma entrada está desenergizada	1,54	1,72
Energizar uma saída	4,43	4,43
Temporizador (*ON*)	30,38	38,34
Contador (*UP*)	26,67	29,28
Contador (*DOWN*)	27,22	32,19
Comparação Igual	6,60	21,52
Comparação Maior que	6,60	23,60
Conversão BCD	6,78	49,64

2.1.5 Tempos de Referência para Processamento de CLP

O tempo de processamento é o espaço de tempo que o CLP leva para detectar uma entrada e energizar a saída correspondente (ALLEN BRADLEY, 1996). Os componentes do tempo de processamento incluem: tempo de atualização das entradas, tempo de execução do programa, tempo de atualização das saídas e tempo de *housekeeping*[12] da CPU (diagnósticos). A tabela 3 apresenta um exemplo do cálculo dos tempos de referência para processamento de um CLP FP0, da empresa Matsushita, para processar um ciclo de varredura de um programa de controle de um Transportador Automático de Peças.

[12] A cada início de um ciclo de *scan* a CPU verifica as condições iniciais do CLP, ou seja, se ocorreu alguma falha em um dos seus componentes internos (fonte, cartões de E/S, memória, etc).

TABELA 3 - Tempos de referência de um ciclo de *scan*

Descrição do Processamento	Tempo Máximo (μs)
Tempo de Atualização das Entradas	8,0
Tempo de Execução do Programa	9,7
Tempo de Atualização das Saídas	8,0
Tempo de *housekeeping* (diagnósticos)	18,0
Tempo Total de Processamento (máximo)	43,75

2.1.6 Tempos de Atraso no Processamento de CLP

O ciclo de varredura muitas vezes pode ocasionar problemas graves no controle de processos industriais, por não reconhecer uma entrada durante o seu ciclo de funcionamento. Isto pode ocorrer com sinais de entrada de resposta rápida, como por exemplo, sensores com resposta em torno de 10 kHz.

Dependendo do tempo de variação do estado lógico dos sinais de entrada, o CLP pode demorar mais tempo para acionar a saída ou mesmo nunca reconhecer uma entrada (PUPO, 2002). Para ilustrar essa situação, a figura 9 apresenta a variação de três sensores de resposta rápida (sensor 1, sensor 2 e sensor 3) ligados na entrada de um CLP, as variações ocorrem durante três ciclos de varredura (*scan*). Durante o primeiro ciclo de *scan*, o sensor 1 muda do estado desligado (*off*) para o estado ligado (*on*), durante a fase de execução do programa. Dessa forma, a transição do sensor 1 não é reconhecida nesse ciclo, sendo reconhecida somente na fase de leitura das entradas do segundo ciclo de *scan*.

O sensor 2 muda de estado na fase de atualização das saídas do segundo ciclo de *scan*, sua transição não é reconhecida durante esse ciclo. É reconhecida durante o terceiro ciclo de *scan*, durante a fase de atualização das entradas desse ciclo.

A mudança de estado do sensor 3 não será reconhecida, em nenhum desses ciclos de *scan*, porque tem resposta muito rápida e ocorre durante a fase de execução do programa no terceiro ciclo de *scan*. Para evitar esse tipo de problema, o tempo do ciclo de varredura (*scan*) do CLP deve ser menor que os tempos de amostragens dos sinais envolvidos no sistema.

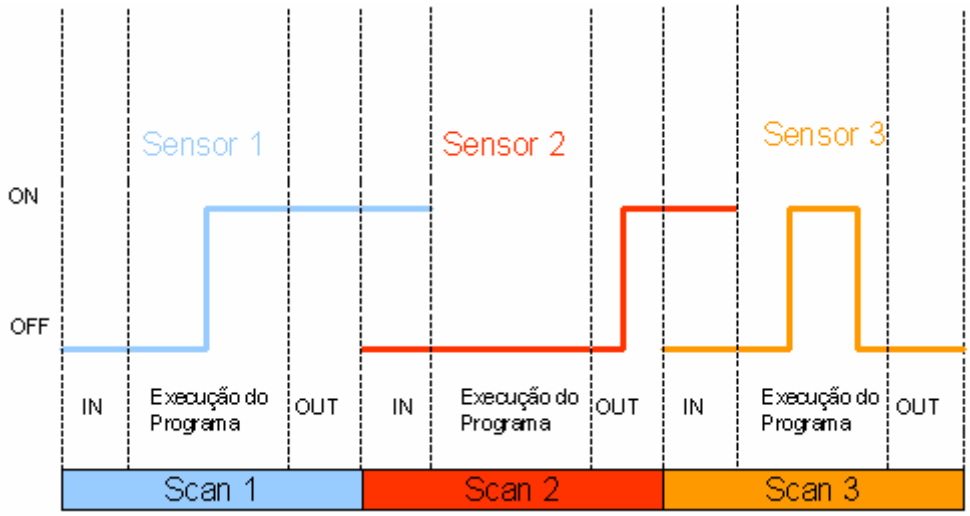

Figura 9 - Sinais de entradas variáveis durante os ciclos de *scan*

2.2 Lógica Programável Estruturada

Os circuitos integrados digitais implementados em pastilha de silício podem ser classificados como circuitos digitais padrões ou circuitos digitais de aplicações específicas (*Application Specific Integrated Circuits* - ASICs). (WAKERLY, 2000). Os circuitos padrões são constituídos por portas lógicas[13] (*AND, OR, NOT* e *Flip-Flops*) e necessitam de vários componentes externos para a realização de uma função específica. Os circuitos integrados ASICs são aqueles que necessitam de um processo de fabricação especial, que requer máscaras específicas para cada projeto. Outras características dos circuitos integrados ASICs são o tempo de desenvolvimento longo e os custos extremamente altos. Geralmente não necessitam de muitos componentes externos para a realização de uma função específica, pois a sua alta densidade torna-os aptos para implementação de vários tipos de aplicações. Porém, em ambos os casos, os circuitos integrados digitais possuem as suas funções internas definitivas, implementadas na sua construção no processo de fabricação (TEIXEIRA, 2002).

O desenvolvimento de projetos de circuitos digitais tem evoluído rapidamente nas últimas décadas. A utilização de *software* denominado de EDA (*Eletronic Design Automation*) e o aperfeiçoamento dos Dispositivos Lógicos Programáveis, PLDs (*Programmable Logic Devices*), tem simplificado e acelerado todo o ciclo de projeto (ARAGÃO, 1998).

[13] Circuitos lógicos básicos que permitem implementar expressões geradas pela álgebra de Boole.

Os 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 circuito integrado, 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 menor e custos reduzidos (TEIXEIRA, 2002).

2.2.1 Dispositivos de Lógica Programável

Os Dispositivos Lógicos Programáveis (PLDs) foram os dispositivos eletrônicos que possibilitaram a implementação da Lógica Programável Estruturada. Os PLDs podem ser classificados em função do número de portas lógicas que comportam, como descrito a seguir (ZAGHETTO et al., 2001):

- **SPLDs (*Simple Programmable Logic Devices*)**: são dispositivos simples de baixa capacidade, tipicamente contém menos de 600 portas lógicas, fabricados com tecnologia CMOS[14].
- **HCPLDs (*High Complex Programmable Logic Devices*)**: são dispositivos de alta capacidade, tipicamente contém mais do que 600 portas lógicas, os mais modernos podem atingir cerca de 250.000 portas e englobam os dispositivos CPLDs (*Complex*

[14] *Complementary Metal Oxide Semicondutor*(CMOS): trata-se de uma família de circuitos integrados, que tem seus circuitos construídos por transistores MOS-FET (*Metal Oxide Semicondutor- Field Effect Transistor)* do tipo canal N ou canal P.

Programmable Logic Devices) e FPGAs (*Field Programmable Gate Arrays*), todos fabricados com tecnologia CMOS.

2.2.2 Dispositivos Lógicos Programáveis Simples

Os PLAs (*Programmables Logic Arrays*) foram os primeiros Dispositivos Lógicos Programáveis Simples (SPLDs) criados especificamente para a implementação de circuitos lógicos. Introduzidos pela empresa Philips no início dos anos 70, esses dispositivos consistem de dois níveis de portas lógicas: um plano de portas *AND* seguido por um plano de portas *OR*, ambos programáveis. Um PLA é estruturado de tal forma que cada saída do plano *AND*, conforme apresentado na figura 10, pode corresponder a qualquer produto das entradas. Da mesma forma, cada saída do plano *OR* pode ser configurada para produzir a soma lógica de quaisquer saídas do plano *AND* (TEIXEIRA, 2002).

A figura 10 mostra um esquema simplificado de um PLA. Em virtude da estrutura montada, os PLAs são adequados para as implementações de funções lógicas na forma de produtos de soma, e eles se apresentam muito versáteis, pois tanto os termos *AND* como os termos *OR* podem ter muitas entradas. Entretanto, essa tecnologia também apresenta alguns problemas como alto custo de fabricação e baixo desempenho em termos de velocidade. Essas desvantagens existem devido aos dois níveis de lógica configurável. Os planos lógicos programáveis são difíceis de serem fabricados e introduzem atrasos significativos de propagação dos sinais elétricos (TEIXEIRA, 2002).

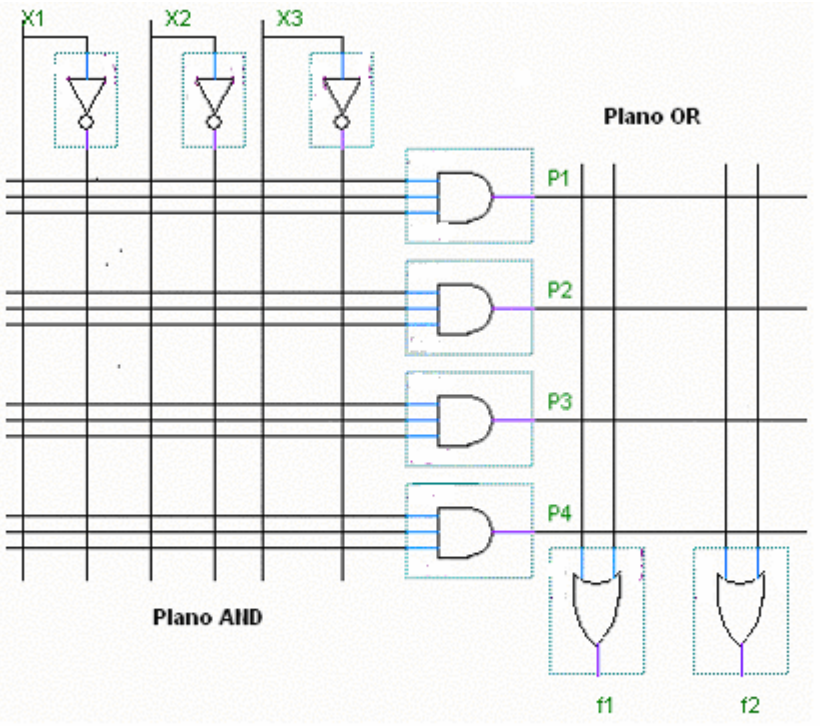

Figura 10 - Esquema simplificado de um PLA

A tecnologia PAL (*Programmable Array Logic*) foi então desenvolvida para superar as deficiências apresentadas pela tecnologia PLA. Os PALs apresentam um único plano *AND* configurável, um custo menor e um melhor desempenho (TEIXEIRA, 2002). A figura 11 apresenta um esquema simplificado de um dispositivo PAL.

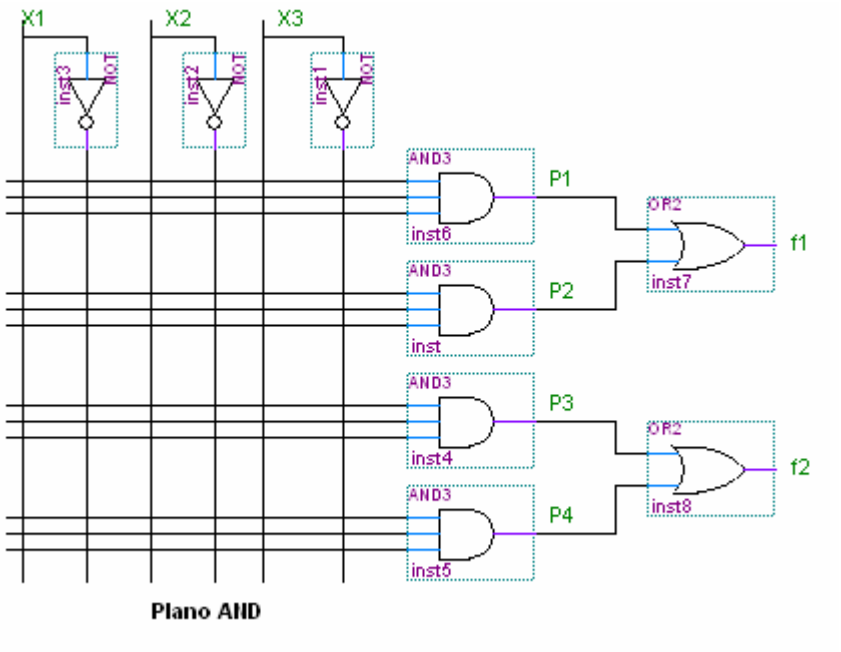

Figura 11 - Esquema simplificado de um PAL

2.2.3 Dispositivos Lógicos Programáveis de Alta Complexidade

Quanto maior o número de portas de um PLD, maior será sua complexidade. Os Dispositivos Lógicos de Alta Complexidade (HCPLD) dividem-se, basicamente em dois grupos: CPLD[15] e FPGA[16]. A diferença básica entre os dois dispositivos está na estrutura interna de suas células lógicas e na metodologia de interligação dessas células. De uma forma geral, internamente os HCPLDs podem ser vistos como dispositivos que integram na sua estrutura

[15] CPLD (*Complex Programmable Logic Device*) – Dispositivo PLD de alta complexidade.
[16] FPGA(*Field Programmable Gate Array*) – Dispositivo PLD que suporta tanto a implementação de circuitos lógicos relativamente grandes como processadores, e blocos simples como *flip-flops.*

centenas de macrocélulas programáveis, que são interligadas por conexões também programáveis.

2.2.3.1 Dispositivos Lógicos Programáveis Complexos

Os Dispositivos Lógicos Programáveis Complexos (CPLDs) foram introduzidos no mercado internacional pela empresa Altera Corp. em 1983, inicialmente como Dispositivos Lógicos Programáveis Apagáveis (EPLDs - *Erasable* PLDs) e, posteriormente, como CPLDs.

Os CPLDs são dispositivos programáveis e reprogramáveis pelo usuário, com alto desempenho, baixo custo por função e alta capacidade de integração. Um CPLD pode ser aplicado, por exemplo, como uma máquina de estado ou decodificador de sinais, substituindo centenas de circuitos discretos que implementariam a mesma função (ALTERA, 1995). Os CPLDs implementam capacidade lógica de até 50 dispositivos PLDs típicos. 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 defina os sinais elétricos conforme desejado: entradas ou saídas podem ocupar o mesmo terminal do dispositivo.

- **Teste e depuração**: as linguagens utilizadas na programação do dispositivo permitem a simulação, teste e depuração rápida do protótipo.

2.2.3.2 A Tecnologia FPGA

Os dispositivos ASICs, SPLDs e CPLDs, descritos nas seções anteriores, permitem a implementação de uma grande variedades de circuitos lógicos. Contudo, com exceção dos CPLDs, aqueles componentes possuem baixa capacidade lógica e são viáveis apenas para aplicações relativamente pequenas. Até mesmo para os CPLDs, apenas circuitos moderadamente grandes podem ser acomodados em um único circuito integrado (TEIXEIRA, 2002).

Para se implementar circuitos lógicos maiores, é conveniente utilizar-se de outro tipo de dispositivo HCPLD que possui capacidade lógica maior. O FPGA é um HCPLD que suporta a implementação de circuitos lógicos relativamente grandes. Consiste de um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar roteamento para comunicação entre elas. O primeiro FPGA disponível comercialmente foi desenvolvido pela empresa Xilinx Inc, em 1983 (ARAGÃO, 1998).

Os FPGAs não possuem planos *OR* ou *AND*, consistem de um grande arranjo de células configuráveis que podem ser utilizadas para a implementação

de funções lógicas. A figura 12 ilustra a estrutura interna simplificada de um FPGA. Basicamente é constituída por blocos lógicos, blocos de entrada e saída, e chaves de interconexão. Os blocos lógicos formam uma matriz bidimensional, e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas de blocos lógicos. Esses canais de roteamento possuem chaves programáveis que permitem conectar os blocos lógicos de maneira conveniente, em função das necessidades de cada projeto (TEIXEIRA, 2002).

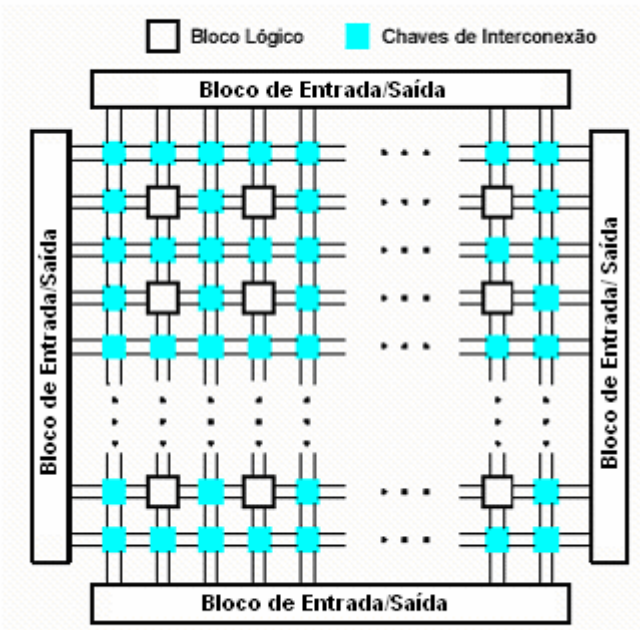

Figura 12 - Estrutura simplificada de um FPGA

2.2.3.2.1 Blocos Lógicos de FPGA

No interior de cada bloco lógico do FPGA existem vários modos possíveis para implementação de funções lógicas. O mais utilizado pelos fabricantes de FPGA como, por exemplo, a empresa Altera Corp, é o bloco de memória LUT[17] (*Look-Up Table*) (ALTERA, 1995). Esse tipo de bloco lógico contém células de armazenamento que são utilizadas para implementar pequenas funções lógicas. Cada célula é capaz de armazenar um único valor lógico, zero (0) ou um (1) (TEIXEIRA, 2002). Nos FPGAs disponíveis comercialmente como, por exemplo, da empresa Altera Corp., os blocos lógicos LUTs possuem geralmente quatro ou cinco entradas, o que permite endereçar 16 ou 32 células de armazenamento.

Quando um circuito lógico é implementado em um FPGA, os blocos lógicos são programados para realizar as funções necessárias, e os canais de roteamento são estruturados de forma a realizar a interconexão necessária entre os blocos lógicos. As células de armazenamento dos LUTs de um FPGA são voláteis, o que implica perda do conteúdo armazenado, no caso de falta de suprimento de energia elétrica. Dessa forma, o FPGA deve ser programado toda vez que for energizado. Geralmente utiliza-se uma pequena memória *FLASH* EEPROM (*Electrically Erasable Programmable Read Only Memory*) cuja função é carregar automaticamente as células de armazenamento, toda vez que o FPGA for energizado.

[17] LUT (*Look- UP Table*) – Um tipo de bloco lógico que contém células de armazenamento, utilizadas para implementar pequenas funções lógicas.

2.2.3.2.2 Granularidade

Granularidade é uma característica dos FPGAs relacionada com o grão[18]. A fim de classificar os FPGAs quanto ao bloco lógico, foram criadas algumas categorias (MESQUITA, 2002):

- **Grão grande**: os FPGAs dessa categoria podem possuir como grão unidades lógicas e aritméticas, pequenos microprocessadores e memórias.
- **Grão médio**: os FPGAs de grão médio freqüentemente contêm duas ou mais LUTs e dois ou mais *flip-flops*. A maioria das arquiteturas de FPGAs implementam a lógica em LUTs de quatro entradas.
- **Grão pequeno**: os FPGAs de grão pequeno contêm um grande número de blocos lógicos simples. Os blocos lógicos normalmente contêm uma função lógica de duas entradas ou um multiplexador 4x1 e um *flip-flop.*

2.2.3.2.3 Arquitetura Geral de Roteamento

A arquitetura de roteamento de um FPGA é a forma pela qual os seus barramentos e as chaves de interconexão são posicionados para permitir a interconexão entre as células lógicas (ROSE et al., 1993). Essa arquitetura deve permitir que se obtenha um roteamento completo e, ao mesmo tempo, uma alta densidade de portas lógicas. A figura 13 mostra uma arquitetura geral de roteamento de um FPGA. Para uma melhor compreensão dessa arquitetura é

[18] Grão é a menor unidade configurável da qual é composto um FPGA.

necessária a definição de alguns conceitos básicos, sendo que parte desses são exemplificados na figura 13:

- **Pinos**: entradas e saídas dos blocos lógicos.
- **Conexão**: ligação elétrica de um par de pinos.
- **Rede**: um conjunto de pinos que estão conectados.
- **Bloco de Comutação**: utilizado 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.

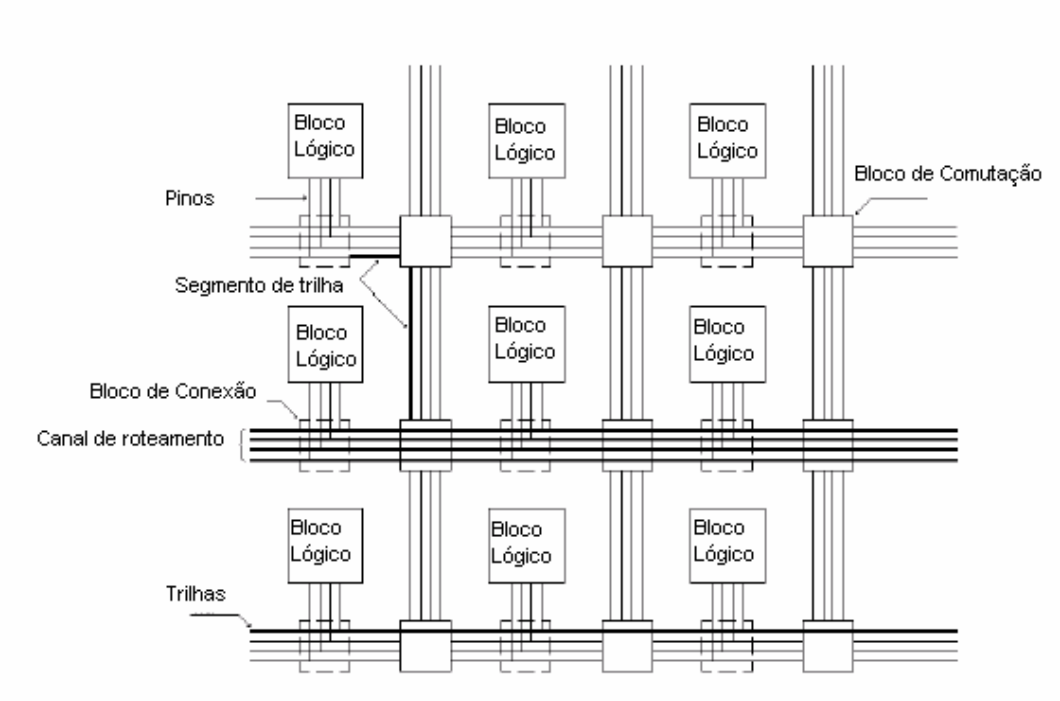

Figura 13 - Arquitetura geral de roteamento de um FPGA

2.2.3.2.4 Tecnologia de Programação

As chaves ou comutadores programáveis de roteamento apresentam algumas propriedades, tais como, tamanho, resistência, capacitância e tecnologia de fabricação, que afetam principalmente a velocidade e o tempo de propagação dos sinais, e definem características como volatilidade[19] e capacidade de reprogramação. Na escolha de um dispositivo reconfigurável, esses fatores devem ser avaliados (TEIXEIRA, 2002). Basicamente existem três tipos de tecnologia de programação das chaves de roteamento:

- ***SRAM* (*Static Random Access Memory*)**: nessa tecnologia, a chave de roteamento ou comutador é um transistor de passagem ou um multiplexador controlado por uma memória estática de acesso randômico SRAM. A figura 14 ilustra essa tecnologia de programação, na qual uma célula de SRAM é utilizada para controlar a porta (*gate*) do transistor de passagem. Devido à volatilidade dessas memórias, os FPGAs que se utilizam dessa tecnologia precisam de uma memória externa tipo *FLASH* EEPROM. Essa tecnologia ocupa muito espaço no circuito integrado, entretanto é rapidamente reprogramável (ARAGÃO, 1998);

[19] Volatilidade é a capacidade que um dispositivo de armazenamento tem em perder todos os seus dados quando a energia elétrica é removida.

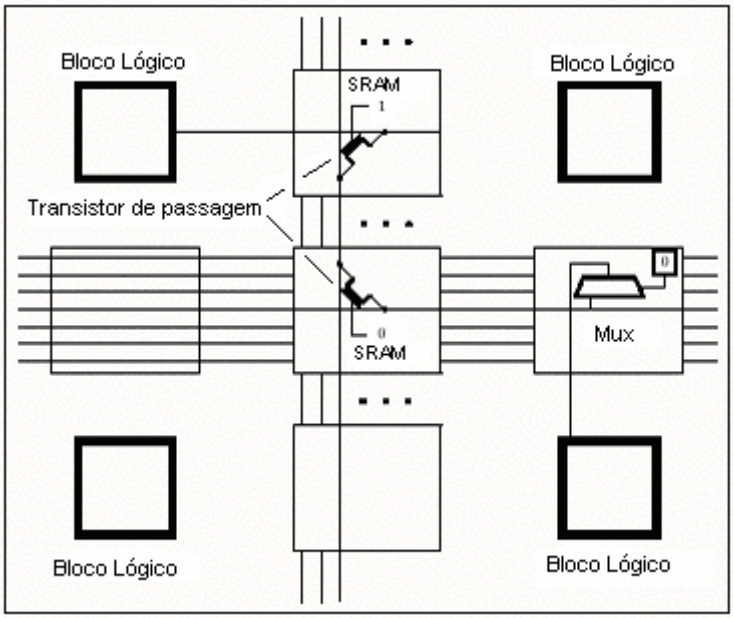

Figura 14 - Tecnologia de programação SRAM

- ***Antifuse***: essa tecnologia baseia-se num dispositivo de dois terminais, que no estado não programado apresenta uma alta impedância (circuito aberto). Aplicando-se uma tensão, por exemplo, entre 11 e 20 Vdc, o dispositivo forma um caminho de baixa impedância entre seus terminais (ARAGÃO, 1998);
- ***Gate* flutuante**: a tecnologia *Gate* flutuante baseia-se em transistores MOS[20] (*Metal Oxide Semiconductor*), especialmente construído com dois *gates* flutuantes semelhantes aos usados nas memórias EPROM (*Erasable Programmable Read Only Memory*) e EEPROM (*Electrical* EPROM). A figura 15 ilustra uma chave

[20] MOS (*Metal Oxide Semiconductor)* – Semicondutor de Óxido de Metal.

programável baseada em *Gate* flutuante. A maior vantagem dessa tecnologia é a sua capacidade de programação e a retenção dos dados. Além disso, da mesma forma que uma memória EEPROM, os dados podem ser programados com o circuito integrado instalado na placa, característica denominada ISP (*In System Programmability*) (ARAGÃO, 1998).

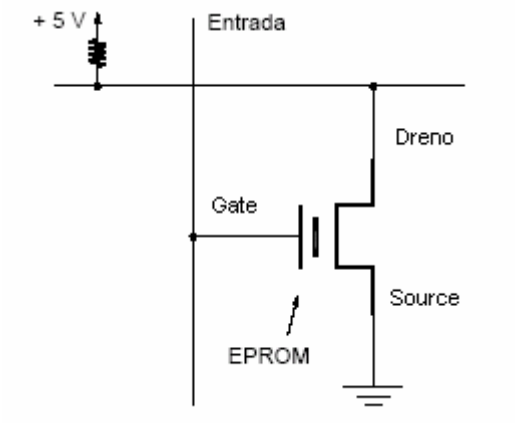

Figura 15 - Tecnologia de Programação *Gate* flutuante

2.2.4 A Família de FPGAs FLEX 10K

A família de FPGAs FLEX 10K, fabricada pela empresa Altera Corp., consiste de uma hierarquia de três níveis muito similar à encontrada nos CPLDs. Contudo, o nível mais baixo da hierarquia consiste de um conjunto de células lógicas LUTs (*Look-up Tables*), ao invés de blocos lógicos como os SPLDs (*Single Programmable Logic Devices*). Portanto, a família FLEX 10K pertence à categoria dos FPGAs (MILANI, 1998). Deve ser notado, entretanto, que a série

FLEX 10K é uma combinação das tecnologias de FPGAs e CPLDs (BROWN, 1996).

A série FLEX 10K é baseada em tecnologia SRAM e possui uma LUT de quatro entradas como seu elemento lógico básico LE (*Logic Element*). Sua capacidade lógica está na faixa de 10.000 a 250.000 portas lógicas (ALTERA, 1998). A arquitetura interna da família FLEX 10K é mostrada na figura 16, contém três tipos de células lógicas: elemento lógico (LE), bloco de matriz lógica (LAB) e bloco de memória embutido (EAB) (ALTERA, 1998).

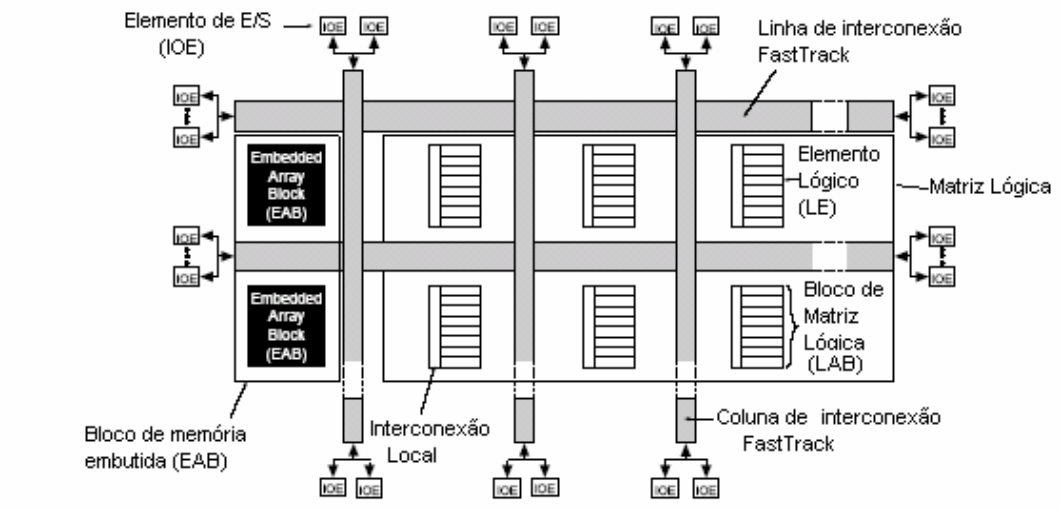

Figura 16 - Arquitetura *MultiCore* da família FLEX 10K

2.2.4.1 Arquitetura *MultiCore*

Os blocos de memória embutidos EABs (*Embedded Array Block*) constituem-se no elemento principal da arquitetura *MultiCore*. Cada bloco EAB contém 2.048 bits programáveis que podem ser configurados como RAM, FIFO

(*First-in First-out*), etc. Podendo ser configurado para atuar como um bloco de memória SRAM com tamanhos variáveis entre: 256 x 8, 512 x 4, 1K x 2 ou 2K x 1. Além disso, um bloco EAB pode ser configurado para implementar um circuito lógico complexo, como um multiplexador, microcontrolador, máquina de estado ou uma função DSP (*Digital Signal Processing*). Quando implementadas essas funções complexas, cada bloco EAB pode utilizar entre 100 a 600 portas lógicas (ALTERA, 1998).

Na arquitetura *MultiCore* FLEX 10K, o bloco lógico básico, chamado Elemento Lógico LE (*Logic Element*), contém uma LUT de quatro entradas, um *flip-flop* e circuitos de *carry*[21] de finalidade especial para circuitos aritméticos. O LE também inclui circuitos em cascata que permitem a implementação eficiente de funções *AND* de várias entradas. Na arquitetura apresentada na figura 17, os LEs são agrupados em dez conjuntos, chamado Bloco de Matriz Lógica LAB (*Logic Array Block*). Cada LAB contém interconexão local e cada trilha local pode conectar qualquer LE a outro LE na mesma LAB. A interconexão local também pode ser ligada à interconexão global, chamada *FastTrack*. Cada trilha *FastTrack* estende-se pela altura ou largura completa do dispositivo. Isso facilita a configuração automática feita pelas ferramentas EDA. Todas as *FastTrack* horizontais são idênticas, portanto os atrasos de interconexões na série FLEX 10K são mais previsíveis do que em outros FPGAs que empregam segmentos menores, pois há menos comutadores programáveis em caminhos longos.

[21] Circuito de transporte de saída equivalente ao “vai um” na operação aritmética de soma.

Os pinos de entrada e saída (E/S) são alimentados individualmente por um elemento chamado IOE (*Input Output Element*) localizado no final de uma linha ou coluna da interconexão global. Os elementos IOEs contém *buffers* bidirecionais e *flip-flops* que podem ser configurados como registros de entrada ou saída.

2.2.4.2 Interconexão MegaLab

A arquitetura *MultiCore* introduziu um novo conceito no nível de hierarquia chamado de estrutura *MegaLAB*. Essa estrutura contém 16 conjuntos de LABs e um bloco de memória EAB. Conjuntos de LABs e EABs são interligados por uma estrutura de interconexão *MegaLAB*, utilizando-se para isso de poucos recursos de roteamento disponíveis no componente. Em adição, a troca de sinais entre LABs adjacentes são realizadas através de um barramento de conexão local, aliviando assim a utilização das conexões chamadas *MegaLAB.* A figura 17 mostra a estrutura de interconexão *MegaLAB*. Para roteamento de sinais entre as estruturas MegaLAB e os pinos de entrada e saída utiliza-se a estrutura global ou *FastTrack*, apresentada na figura 16*,* formada por uma série de canais em forma de linhas e colunas contínuas dispostas ao longo de toda área útil do dispositivo (ALTERA, 1998).

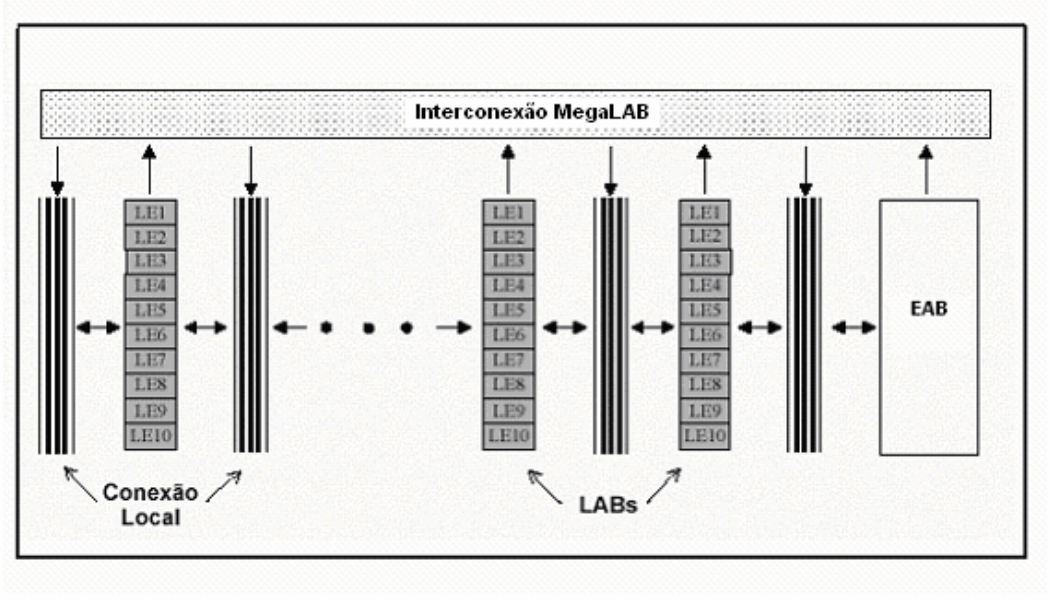

Figura 17 - Estrutura de interconexão *MegaLAB*

2.2.5 Esquemas de Programação de FPGAs

Nos FPGAs baseados em tecnologia de programação SRAM, vista no subitem anterior 2.2.3.2.4, a programação deve ser carregada toda a vez que o sistema for iniciado ou quando uma nova programação for necessária, devido a sua volatilidade. A família FLEX 10K suporta vários esquemas de configuração, que serão apresentados a seguir:

- **Programação via EPROM (*Erasable Programmable Read* Only *Memory*)**: esse esquema de programação utiliza uma EPROM externa para armazenamento dos dados de programação, por exemplo, as memórias EPROM EPC1 e EPC2 de fabricação da empresa Altera Corp., ou a EPROM 2764 podem ser utilizadas, sem a necessidade de um controlador externo. Os sinais de controle

da memória EPROM interagem diretamente com os sinais de controle do FPGA. A figura 18 apresenta um esquema de interligação com EPROM (ALTERA, 1998).

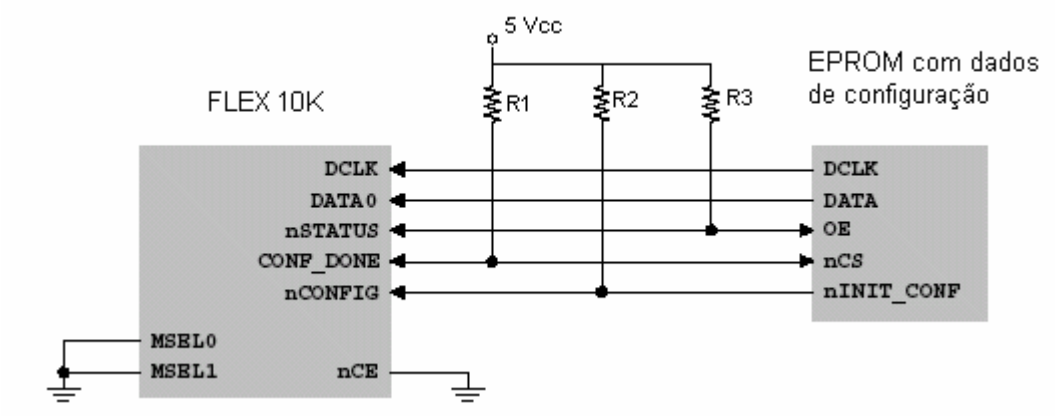

Figura 18 - Programação via EPROM externa

- **Programação serial passiva via cabo de *download***: nesse tipo de programação, um microcomputador padrão IBM PC transfere dados, a partir de um dispositivo de armazenamento para o FPGA, via porta serial, utilizando, por exemplo, um cabo padrão serial para impressora. A figura 19 ilustra um esquema de interligação com o cabo de *download* (ALTERA, 1998).

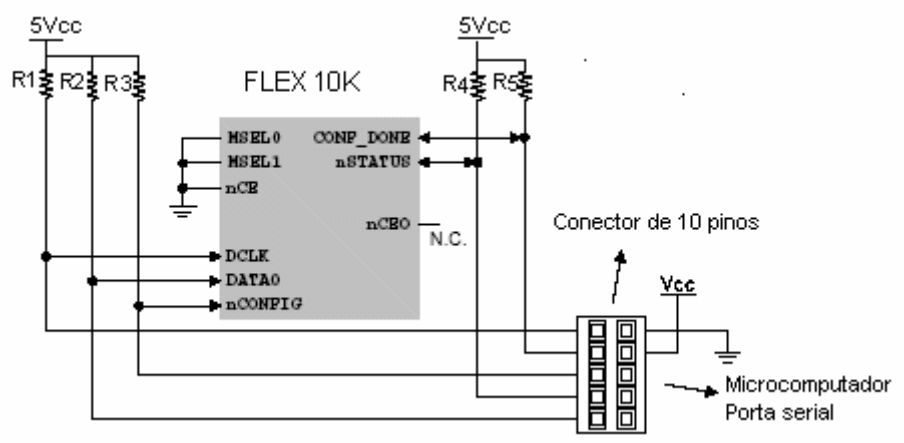

Figura 19 - Programação via cabo de *download*

- **Programação Serial Passiva Via Microprocessador**: nesse esquema de programação, um microprocessador transfere dados a partir de um dispositivo de memória, por exemplo, para o FPGA, via programação em *hardware*. A figura 20 ilustra um esquema de programação com microprocessador (ALTERA, 1998).

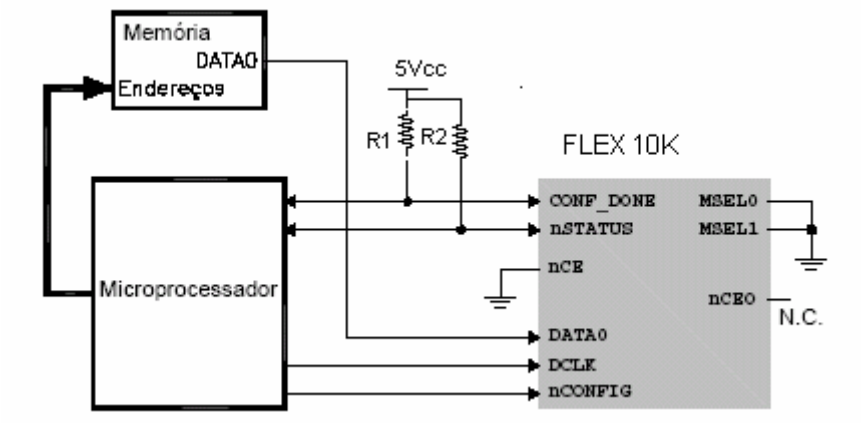

Figura 20 - Programação via microprocessador

- **Programação JTAG**: o padrão IEEE[22] 1149.1 comumente chamado JTAG (*Joint Test Action Group*) é um método de testes. Esse padrão provê maneiras de se assegurar a integridade de componentes individuais e as interconexões entre esses no nível de placa de circuito impresso (ALTERA, 1998). Dispositivos contendo esta arquitetura podem enviar dados através de seus pinos de entrada/ saída, de forma a testar suas conexões e também serem usados para testar o funcionamento de dispositivos específicos contidos em uma placa de circuitos. A figura 21 ilustra um esquema de programação via interface JTAG.

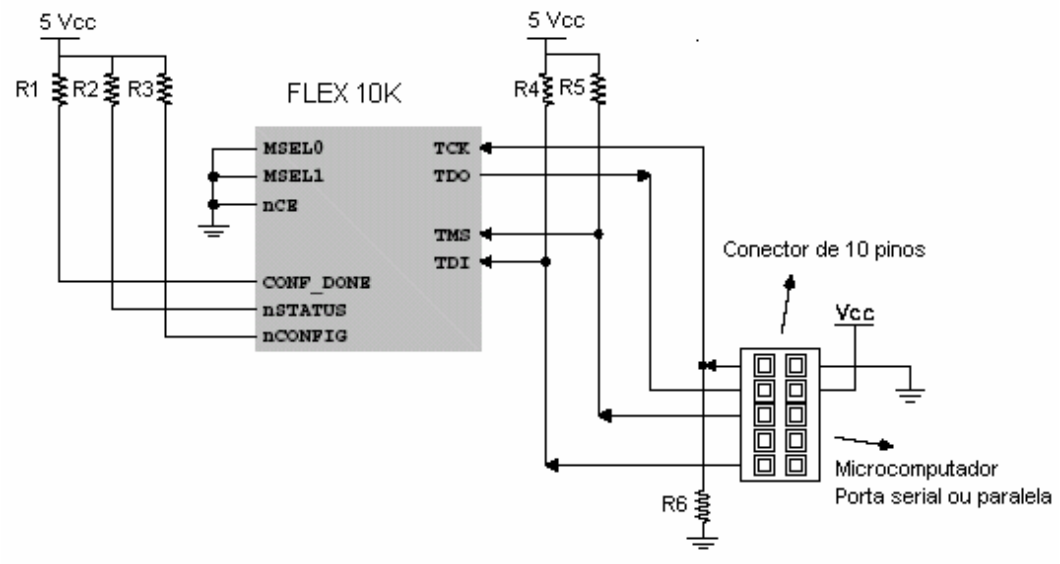

Figura 21 - Programação via interface JTAG

[22] IEEE (*Institute of Electrical and Electronic Engineers*) – Organização Internacional responsável pela padronização de normas técnicas elétricas e eletrônicas.

2.2.6 Aplicações de FPGAs

Uma arquitetura baseada em lógica programável estruturada, por exemplo, com FPGA, pode ter como meta uma das seguintes características: tolerância a falhas ou melhoria de desempenho (MESQUITA, 2002).

A tolerância à falha foi um dos primeiros campos em que a arquitetura baseada em FPGA foi implementada, pois durante a fabricação e a utilização, há uma certa possibilidade de que uma parte de um circuito integrado torne-se defeituosa e, numa arquitetura FPGA tolerante a falhas, o sistema ainda poderia continuar operando, pois seria capaz de detectar e corrigir as falhas a partir da reconfiguração do circuito (GONSALES et al., 2001).

O uso de arquiteturas baseadas em FPGAs para incrementar a velocidade em Controladores Lógicos Programáveis (CLPs) é uma tecnologia que emergiu rapidamente a partir da década de 80, principalmente pela possibilidade de implementação de algoritmos de controle diretamente em *hardware*.

2.2.6.1 Coprocessamento

A arquitetura baseada em Lógica Programável Estruturada é a tecnologia que impulsionou o desenvolvimento de sistemas com capacidade de unir as características de processadores de propósito geral e a flexibilidade de FPGAs. Existem diversas formas de se usar uma arquitetura baseada em lógica programável estruturada em conjunto com microcontroladores. Uma forma de classificar esses sistemas é através dos diferentes níveis de interação entre o FPGA e o microcontrolador de propósito geral (MESQUITA, 2002), como apresentado a seguir:

- **Coprocessamento**: o microprocessador envia uma instrução, ou seqüência de instruções que são detectadas e interpretadas pelo FPGA. Na realidade, o FPGA atua como um coprocessador.
- **Chamada remota de funções**: o microprocessador envia uma instrução, ou seqüência de instruções que é interpretada pelo FPGA como uma chamada remota de função (*Remote Procedure Call* – RPC). É semelhante ao coprocessamento, exceto pelo fato de que é uma interface com mais recursos, que usa sincronização explícita sempre que necessário;
- **Modelo Cliente-Servidor**: o algoritmo implementado no FPGA é um processo servidor que atua de forma semelhante ao mecanismo do RPC, mas em que comunicações podem chegar de qualquer processo que esteja sendo executado pelo microprocessador;
- **Processos paralelos**: os processos que são executados pelo FPGA são independentes daqueles que são executados pelo microprocessador. A comunicação entre processos pode acontecer a qualquer momento, via troca de mensagens.

2.2.6.2 Execução de Programas

Existem diversas formas do algoritmo do programa a ser configurado no FPGA (MESQUITA, 2002), sendo as principais apresentadas a seguir:

- ***Hardware* puro**: o algoritmo é convertido, por síntese[23] para *hardware*, em uma descrição de *hardware* que é carregada no FPGA.
- **Microprocessador de aplicações específicas**: o algoritmo é transformado de um código de máquina abstrato para um processador abstrato (ASIP - *Application Specific Instruction-Set Processor*). O processador ASIP é então melhorado para produzir a descrição de um processador de aplicação específica e o código de máquina para ele. A descrição do microprocessador pode então ser configurada em um FPGA.
- **Reutilização seqüencial**: o algoritmo pode ser muito grande para ser implementado no FPGA. Por razões de engenharia ou econômicas, divide-se o algoritmo em partes, de tal forma que ele seja executado parcialmente, usando a capacidade de reconfiguração dinâmica do FPGA. Os ganhos relacionados com a reutilização do *hardware* devem ser balanceados com o tempo em que é utilizado com a reconfiguração.
- **Uso múltiplo simultâneo**: se os recursos do FPGA são grandes o bastante, é possível haver diversos algoritmos residentes, e cada um deles pode interagir separadamente com o processador.

[23] Processo de transformação automática de uma descrição de projeto abstrata de nível mais alto em nível de portas lógicas.

- **Uso sob demanda**: existe a possibilidade de sistemas computacionais serem construídos onde o *hardware* não existe todo ao mesmo tempo, mas cuja demanda de tempo-real do sistema dita qual parte do *hardware* deve ser utilizada e qual parte deve ser abandonada. Há uma analogia razoável com sistemas de memória virtual e, por isso, esse esquema pode ser chamado de "*hardware* virtual".

2.2.7 Sistemas Digitais Baseados em Lógica Programável Estruturada

As arquiteturas baseadas em Lógica Programável Estruturada, por exemplo FPGAs, podem ser analisadas ao longo do tempo, em função dos problemas a que se dispuseram resolver. A partir do amadurecimento dessa tecnologia, alguns centros de pesquisas criaram as primeiras arquiteturas FPGAs, com o objetivo principal de aumentar o desempenho de algoritmos que até então eram executados em *software* (MESQUITA, 2002).

Num primeiro momento verificou-se a eficiência da utilização de FPGAs em domínio de aplicações específicas, tanto em termos de desempenho com relação a abordagens em *software,* quanto no que tange ao critério econômico, quando comparada a soluções ASICs. Contudo, também alguns problemas foram detectados, principalmente no fato de os FPGAs apresentarem tempo de reconfiguração muito alto (MESQUITA, 2002). O avanço tecnológico ocorrido nos FPGAs possibilitou a reconfiguração dinâmica. Isso permitiu que as arquiteturas baseadas em Lógica Programável Estruturada pudessem ser configuradas sem que precisassem parar totalmente de desempenhar suas funções.

2.2.8 Desenvolvimento de Projetos Utilizando FPGAs

O processo de projeto com FPGAs envolve várias etapas que geralmente são automatizadas. Atualmente, a utilização de ferramentas de *software* EDA (*Eletronic Design Automation*) tem simplificado e acelerado todo o ciclo de projeto (MILANI, 1998). Um sistema típico de desenvolvimento de projetos, com ferramentas de *software* EDA, consiste de vários programas interconectados, conforme ilustrado na figura 22. Esse processo envolve as seguintes etapas:

- Especificação e entrada do projeto.
- Síntese e mapeamento da tecnologia.
- Posicionamento e roteamento.
- Verificação e teste.
- Programação do FPGA.

2.2.8.1 Especificação e Entrada de Projeto

A entrada de projetos pode ser realizada de duas formas: um diagrama esquemático, desenvolvido a partir de um *software* gráfico, por exemplo, no qual é possível utilizar portas lógicas e macroinstruções, ou através de uma linguagem de descrição de *hardware* HDL (*Hardware Description Language*).

A especificação do projeto é apresentada em termos abstratos ou em métodos formais, seguida pela análise da viabilidade da implementação por meio de simulação de alto nível. Nessa fase é importante que a linguagem utilizada seja o mais próximo possível da linguagem humana (ARAGÃO, 1998).

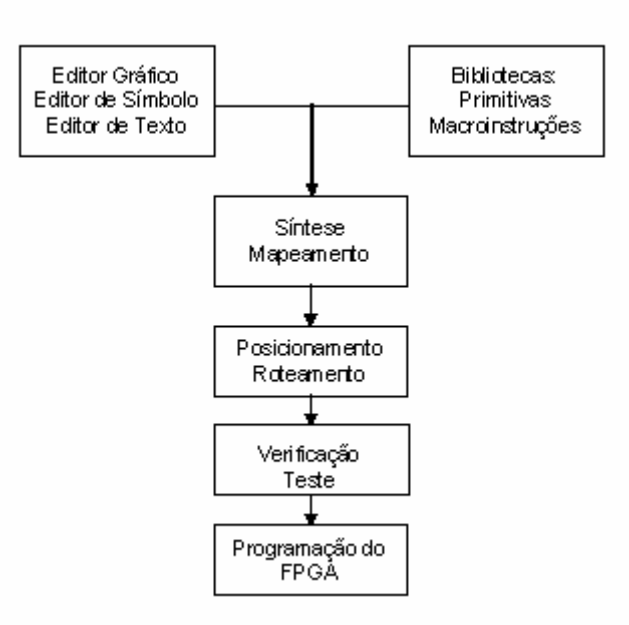

Figura 22 - Ambiente de Desenvolvimento EDA

2.2.8.1.1 Editores Esquemáticos

As ferramentas de captura de esquemático ou editores gráficos permitem que o projetista especifique o circuito como um diagrama lógico em 2D[24], conectando componentes lógicos com recursos de roteamento. A figura 23 mostra a tela de edição do editor de projetos do *Software* QUARTUS II[25]. Os componentes lógicos estão contidos em uma biblioteca de macroinstruções fornecidas pelo *software* ou podem ser definidas pelo próprio usuário. Geralmente, as bibliotecas contem portas lógicas, pinos de entrada e saída, *buffers*, multiplexadores, *flip-flops*, *latches*, decodificadores, registradores, contadores, comparadores, memórias, funções aritméticas, e outras funções

[24] 2D significa em duas dimensões.
[25] QUARTUS II é um *software* utilizado para desenvolvimento e programação de Dispositivos Lógicos Programáveis fabricados pela empresa Altera Corp.

especiais. Estão disponíveis também símbolos especiais para controle do mapeamento, posicionamento e roteamento durante a fase de implementação do projeto (MILANI, 1998).

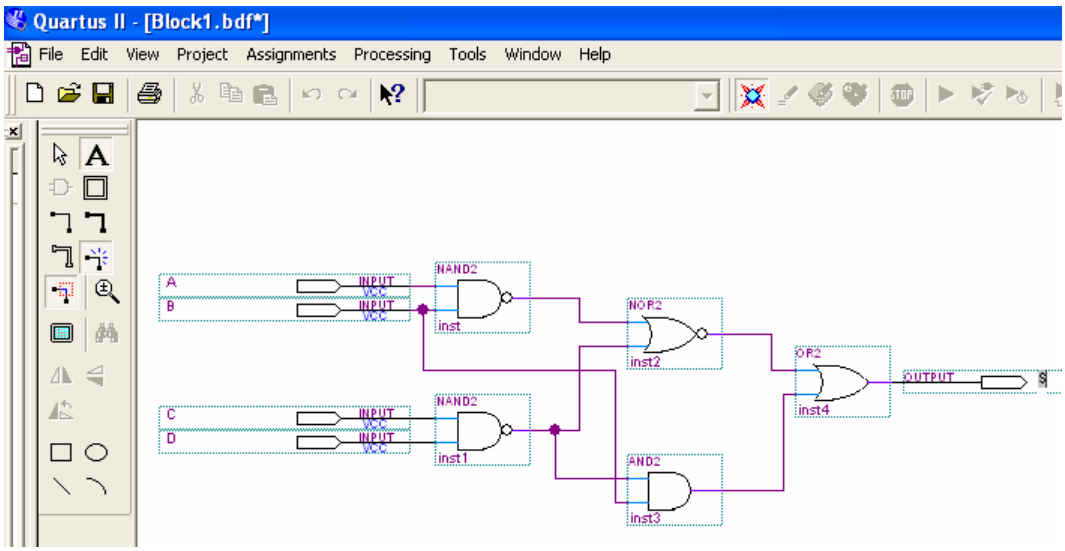

Figura 23 - Editor gráfico do *software* QUARTUS II

2.2.8.1.2 Linguagem de Descrição de *Hardware*

À medida que os projetos ficam mais complexos, as descrições em nível de portas lógicas tornam-se inviáveis, fazendo-se necessário descrever esses 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 (MILANI, 1998).

Existem diversas linguagens de descrição de *hardware* disponíveis, sendo as mais comumente utilizadas: ABEL (*Advanced Boolean Equation Language*),

VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) e Verilog. A linguagem ABEL foi a primeira linguagem HDL a ser desenvolvida. Foi criada pela empresa americana Data I/O Corp para programar dispositivos SPLD, sendo uma linguagem mais simples que a linguagem VHDL. Já a linguagem VHDL e Verilog é capaz de programar sistemas de maior complexidade como, por exemplo, os dispositivos FPGA (WAKERLY, 2000).

As linguagens de descrição de *hardware* HDL são utilizadas para descrever o comportamento de um sistema digital de variadas formas, inclusive equações lógicas, tabelas da verdade e diagramas de formas de onda, que utilizam declarações de constantes, estados, configurações, bibliotecas, módulos, etc, como a linguagem C (COFFMAN, 1999).

2.2.8.2 Síntese Lógica e Mapeamento

A síntese lógica consiste de duas fases distintas: otimização lógica para minimizar equações Booleanas e mapeamento da tecnologia para converter equações em células da biblioteca da tecnologia alvo.

Como a lógica inicial não está otimizada, algoritmos de síntese são utilizados para simplificar as equações Booleanas geradas. A síntese na prática permite a redução de área a ser ocupada no circuito integrado, como também reduz o atraso de propagação (*delay*) dos sinais envolvidos. A fase de mapeamento da tecnologia seleciona um conjunto de portas lógicas de uma dada biblioteca para implementar as representações abstratas, enquanto melhora a área, o atraso ou a combinação de ambos, levando em consideração as restrições arquiteturais da tecnologia alvo, nesse caso os FPGAs (ARAGÃO, 1998).

2.2.8.3 Posicionamento e Roteamento

Após a minimização lógica e o mapeamento da tecnologia, o projeto consiste de uma representação textual de componentes lógicos a serem designados aos componentes físicos de uma arquitetura de FPGA (ARAGÃO, 1998). O posicionamento e roteamento são dois processos mutuamente dependentes. O posicionamento é atribuição de componentes particulares do circuito integrado aos componentes lógicos do projeto. O roteamento é a atribuição de trilhas e elementos programáveis, consumindo os recursos disponíveis de interconexão para a comunicação entre os componentes. O *software* de roteamento aloca os recursos de roteamento do FPGA para interconectar as células posicionadas. As ferramentas de roteamento devem assegurar que 100% das conexões requeridas sejam realizadas, e deve procurar maximizar a velocidade das conexões críticas, porém, essa meta nem sempre é alcançada.

2.2.8.4 Verificação e Teste

A simulação é o tipo mais comum de verificação utilizada em projetos com FPGAs. A simulação é realizada geralmente na fase inicial, para verificação funcional, podendo ser realizada em nível comportamental ou em nível de portas lógicas.

A simulação também é realizada antes da configuração do FPGA, para verificar restrições de temporização. Vários simuladores para a tecnologia FPGA estão disponíveis comercialmente como, por exemplo, os *softwares* EDA: i) *Viewsim* desenvolvido pela empresa Mentor Graphics; ii) *Synopsys* desenvolvido pela empresa Synopsys.

2.2.8.5 Programação do FPGA

Após a verificação e teste, a implementação do projeto é completada, mas ainda resta um passo final que é a programação do FPGA. Nesse ponto, é gerado um arquivo de configuração, que deve ser carregado no dispositivo alvo.

Um FPGA pode ser programado de diversos modos, como visto anteriormente no item 2.2.5. O modo "programação serial passiva com cabo de *download*" é o mais recomendado, pois o arquivo de configuração pode ser transferido, através da porta de comunicação serial ou paralela do computador diretamente para o dispositivo FPGA, interface JTAG, por um cabo de impressora padrão C*entronics* ou cabo especial, fornecido pelo fabricante, por exemplo, Altera Corp (ARAGÃO, 1998).

2.2.8.6 Ambiente e Ferramentas de *Software* EDA

O *Software* QUARTUS II da empresa Altera possui o ambiente e as ferramentas de *software* EDA, compatíveis com o desenvolvimento deste trabalho. A entrada de projetos pode ser realizada nos seguintes modos:

- Editor Gráfico.
- Editor de Texto.
- Editor de Símbolo Gráfico.
- Editor de Formas de Onda.

2.2.9 Linguagem VHDL

A linguagem VHDL (VHSIC *Hardware Description Language*) surgiu como resultado do programa "*Very High Speed Integrated Circuit*" (VHSIC),

organizado pelo Departamento de Defesa dos Estados Unidos, no início dos anos 80. No decorrer desse programa, tornou-se clara a necessidade de uma linguagem normalizada para descrever a estrutura e a funcionalidade de circuitos integrados. Assim, foi criada a linguagem VHDL. Em 1986, a linguagem VHDL foi proposta como norma IEEE e foi aceita (Standard IEEE 1076-1987) após uma série de revisões e alterações em 1987 (FERNANDES, 1994).

A construção da linguagem VHDL foi influenciada pela linguagem ADA[26] (LEDGARD, 1981). Muitos conceitos foram fornecidos por essa linguagem, entre os quais se incluem (FERNANDES, 1994):

- Especificação separada de interfaces e implementações.
- Pacotes e a possibilidade de importar declarações deles.
- Bibliotecas de programas e compilação separada.
- Tipos definidos pelo usuário.
- Funções de diversas formas.
- Vetores dinamicamente limitados.
- Constantes enumeradas de diversas maneiras.
- Notações compactas para especificar vetores e estruturas em termos dos seus componentes.

Atualmente, todas as ferramentas de *software* EDA disponíveis no mercado e todos os *software*s de desenvolvimento oferecidos pelos fabricantes de FPGAs, aceitam a linguagem VHDL como entrada de projeto. Dessa forma, um

[26] Linguagem de programação de alto nível criada em um concurso realizado pelo Departamento de Defesa dos Estados Unidos, sendo o principal projetista da equipe o francês Jean Ichbiah.

projeto baseado em VHDL pode ser implementado em qualquer tecnologia (ASHENDEN, 1990). A linguagem VHDL permite:

- Através de simulação verificar o comportamento do sistema digital.
- Descrever o *hardware* em diversos níveis de abstração (comportamental e estrutural).
- Simulação e síntese.

2.2.9.1 Estrutura Básica de um Projeto em VHDL

Um modelo VHDL é constituído, na sua forma mais simples, por pacotes (*package*), entidade (*entity*), arquitetura (*architecture*) e configuração (*configuration*) quando necessária. A figura 24 ilustra a estrutura básica de um modelo VHDL.

```
1  LIBRARY IEEE;
2  Use IEEE.std_logic_1164.all;
3  Use IEEE.std_logic_Unsigned.all;
4
5  ENTITY contador IS
6      GENERIC
7      (prop_delay : Time := 2ns);
8      PORT
9      (clock : IN bit;
10      q1,q0 : OUT bit);
11 END contador;
12
13 ARCHITECTURE estrutural OF contador IS
14     SIGNAL ff0, ff1, inv_ff0: STD_LOGIC;
15 BEGIN
16     bit_0: entity work. Tflip_flop
17            port map(ck=>clock, q=>ff0);
18     inv:   entity work.inversor
19            port map(a=>ff0, y=>inv_ff0);
20     bit_1: entity work. Tflip_flop
21            port map(ck=>inv_ff0, q=>ff1);
22     q0 <=ff0;
23     q1 <=ff1;
24 END estrutural;
```

Pacote: Coleção de Informações - Biblioteca.

Entidade: Declaração da Interface Externa

Arquitetura: Descrição Estrutural do Módulo

Figura 24 - Estrutura básica de um modelo VHDL

2.2.9.2 Pacote

Um pacote é uma coleção de informações que pode ser referenciada pelo modelo VHDL, normalmente utilizada no projeto (CASILLO, 2003). Consiste em duas partes: declaração do pacote e estrutura do pacote (opcional). As declarações contêm a definição de funções básicas, tipos, constantes, subprogramas, mnemônicos[27] e permitem a reutilização de um código já escrito. A figura 25 apresenta um exemplo de uma declaração de biblioteca de um pacote; normalmente todos os programas escritos em VHDL iniciam com esse tipo de declaração de biblioteca. Observa-se, por exemplo, que na linha de número dois é declarada a biblioteca IEEE 1164, que contém tipos de padrões lógicos e funções relacionadas.

```
1 LIBRARY IEEE;
2 Use IEEE.std_logic_1164.all;
3 Use IEEE.std_logic_Unsigned.all;
4
```

Figura 25 - Estrutura básica de uma declaração de biblioteca

2.2.9.3 Entidade

Uma entidade é uma abstração, usada para definir a vista externa de um modelo, declara as interfaces do projeto (pinos de entrada/saída), constantes

[27] Representação resumida de instruções de programa (códigos), por meio de três ou quatro letras.

genéricas e etc, que descrevem um sistema, uma placa, um circuito integrado, uma função ou uma porta lógica (ASHENDEN, 1990). Na declaração de uma entidade, descrevem-se as constantes genéricas e o conjunto de entradas e saídas. A figura 26 especifica que a entidade denominada contador tem uma entrada chamada *clock* e duas saídas denominadas de *q1* e *q0*, cujos valores são do tipo *bit* e podem assumir lógica zero (0) ou um (1).

As constantes genéricas são opcionais e podem ser usadas para controlar a operação e o comportamento da entidade. Definem, por exemplo, o tempo de propagação de um sinal, o tamanho de um barramento de uma entrada ou saída da entidade, etc. Na figura 26, a constante genérica chamada *prop_delay* é usada para especificar o tempo de atraso de propagação da entidade, que será de 10 ηs (nanossegundos).

As portas correspondem aos pinos de entrada/ saída e definem os canais de comunicação entre a entidade de projeto e os dispositivos externos. A definição de uma porta envolve a descrição de seu modo e tipo. O modo especifica a direção do fluxo da informação através da porta. No modo de entrada (*IN)* a informação flui para a interface, no modo de saída (*OUT)* a informação flui da interface e no modo entrada/ saída (*IN/ OUT)* a informação flui em qualquer direção. Outro modo de propósito especial é o modo *BUFFER*[28].

[28] Indica que o sinal é uma saída da entidade, cujo valor pode ser lido do interior da arquitetura da entidade.

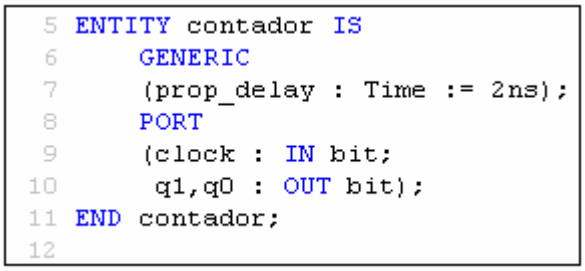

Figura 26 - Estrutura básica de uma declaração de entidade

O tipo da porta especifica o conjunto de valores que os pinos podem assumir. Os valores dos pinos podem ser representados por níveis de tensão (bit), valores falso ou verdadeiro (Booleano), conjunto de bits (vetor) ou valores padrões de uma biblioteca, por exemplo. Cada um desses conjuntos é um tipo e cada um pode ser uma forma de abstração do mesmo fenômeno eletrônico (ASHENDEN, 1990). A tabela 4 apresenta os tipos de declarações mais utilizados em pinos de entrada e saída em VHDL.

TABELA 4 - Tipos de declarações de pinos de entrada e saída

Tipo	Descrição
bit	Assume lógica zero (0) ou um (1).
bit_vector	Assume conjunto de bits: Por exemplo: “001100”ou “X00FF”.
std_logic	Assume padrões de uma biblioteca, por exemplo: *std_logic_1164 Library*.
Std_logic_vector	Assume conjunto de bits padrões de uma biblioteca.
boolean	Verdadeiro (*true*) ou Falso (*false*).

2.2.9.4 Arquitetura

A arquitetura descreve a funcionalidade e a temporização do modelo, devendo estar associada a uma entidade. A funcionalidade da arquitetura depende dos sinais de entrada e saída e dos parâmetros definidos na descrição da entidade.

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*) (FERNANDES, 1994). A figura 27 apresenta o diagrama de um contador de dois bits. A sua arquitetura pode ser descrita por duas formas distintas:

- Descrição comportamental: descreve como o modelo opera, podendo ser descrito em forma de transferências de registros no tempo (RTL[29]) ou funcional sem temporização (CASILLO, 2003);
- Descrição estrutural: descreve o modelo em nível de componentes ou lista de ligações.

[29] Em língua inglesa RTL, iniciais de *Register Transfer Level*.

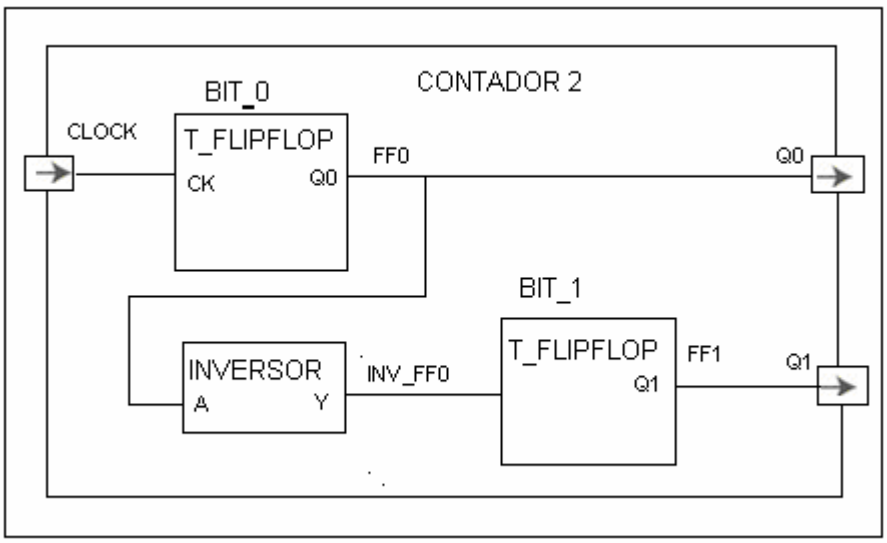

Figura 27 - Diagrama de um contador de dois bits

2.2.9.4.1 Descrição Comportamental

A descrição comportamental é a forma mais flexível e completa de descrição, em que são definidos os processos concorrentes (*process*). A cada processo é associada uma lista de sensibilidade, que indica quais são as variáveis cujas alterações devem levar à reavaliação da saída (CASILLO, 2003). A figura 28 mostra a descrição comportamental em VHDL do contador de dois *bits* apresentado na figura 27. Nessa descrição, o comportamento é implementado por um processo chamado *count-up*, que é sensível à entrada de *clock.* Possui uma variável chamada *count_value* para armazenar o estado corrente do contador. A variável é iniciada em zero para início da simulação e retém esse valor entre ativações do processo. Quando a entrada de *clock* muda do valor lógico zero (0) para o valor lógico um (1), a variável *count value* é incrementada e transições

lógicas são realizadas nas duas portas de saída *q0* e *q1* baseadas no novo valor. A transferência para as saídas *q0* e *q1* utiliza uma constante chamada *prop_delay*, que determina quanto tempo depois de ocorrida a mudança do *clock,* a transição nas saídas serão realizadas. Quando o controle alcança o fim da estrutura do processo, o mesmo é suspenso até que outra mudança ocorra na entrada de *clock*.

```
13 -- Descrição Comportamental do Contador de 2 bits --
14 ARCHITECTURE comportamental OF contador IS
15     SIGNAL q : std_logic_vector(1 dowto 0);
16 BEGIN
17  count_up:process (clock)
18     variable count_value:integer:=0;
19 BEGIN
20     if clock ='1'then
21           count_value:=(count_value + 1)mod 4;
22           q<= conv_std_logic_vector (count_value,2) after prop_delay;
23     End if;
24 End process count_up;
25 q0 <= q(0);
26 q1 <= q(1);
27 End comportamental;
```

Figura 28 - Descrição comportamental do contador de 2 bits

2.2.9.4.2 Descrição Estrutural

Um circuito digital pode ser descrito como um módulo com entradas e saídas. Os valores elétricos nas saídas são uma função dos valores das entradas. Uma forma possível de descrever o funcionamento de um sistema elétrico é indicar como ele é composto. A descrição estrutural apresenta uma lista de ligações (*netlists*) e instanciação[30] de componentes básicos, ou seja, é como se fosse uma lista de ligações básica entre componentes pré-definidos

[30] Em VHDL instanciação é uma decomposição do componente em sub-módulos.

(FERNANDES, 1994). A figura 29 apresenta uma descrição estrutural em VHDL do contador de dois bits mostrado na figura 27. Nessa arquitetura são declarados dois tipos de componentes (*Flip_Flop* tipo T e um inversor) e três sinais internos (*ff0*, *inv_ff0* e *ff1*). Cada componente é dividido em instâncias[31] e suas conexões são mapeadas em sinais internos de entrada e saída da entidade. Por exemplo, *Bit_0* é uma instância do componente *Flip_Flop* tipo T, com sua entrada *Ck* conectada a entrada de *Clock* da entidade CONTADOR 2 e sua saída *q0* conectada ao sinal interno *ff0*. Os sinais *ff0* e *ff1* atualizam as saídas da entidade CONTADOR 2, sempre que os valores dos sinais internos mudarem.

```
13 -- Descrição Estrutural do Contador de 2 bits --
14 ARCHITECTURE estrutural OF contador IS
15     SIGNAL ff0, ff1, inv_ff0 : std_logic;
16 BEGIN
17  bit_0:  ENTITY work.Tflip_flop
18          PORT MAP (ck=>clock, q=>ff0);
19  inv:  ENTITY work.inversor
20          PORT MAP (a=>ff0, y=>inv_ff0);
21  bit_1:  ENTITY work.Tflip_flop
22          PORT MAP (ck=>inv_ff0, q=>ff1);
23 q0 <= ff0;
24 q1 <= ff1;
25 End estrutural;
26
```

Figura 29 - Descrição estrutural do contador de dois bits

2.2.9.4.3 Descrição por Transferência de Registros (RTL)

A descrição por transferência de registros (RTL) ou fluxo de dados é um tipo de descrição comportamental em que os modelos são descritos em termos de

[31] Em VHDL instância é uma declaração opcional de sub-módulo do componente.

registros no tempo. Os valores de saída são atribuídos diretamente, através de expressões lógicas. O sistema é representado por um conjunto de equações concorrentes. Cada uma dessas equações envolve funções definidas pelo projetista e operadores lógicos e aritméticos, que manipulam sinais de tipos complexos. Essas equações expressam o fluxo de dados através dos módulos funcionais RTL que as funções e os operadores implicitamente determinam (FERNANDES, 1994). A figura 30 apresenta um exemplo de uma descrição por transferência de registros (RTL) do *FlipFlop* tipo T da figura 27.

```
1  -- Descrição por Transferência de Registros (RTL) --
2
3  ENTITY TFlip_Flop IS
4      PORT
5      (ck : IN    STD_LOGIC;
6       q  : OUT   STD_LOGIC);
7  END TFlip_Flop;
8
9  ARCHITECTURE comportamental OF TFlip_Flop IS
10     SIGNAL regQ : STD_LOGIC := 0;
11     BEGIN
12       q <= regQ;
13       PROCESS (ck)
14           BEGIN
15               IF (ck'event and ck='1')THEN regQ <= NOT regQ;
16               END IF;
17       END PROCESS
18 END comportamental;
19
```

Figura 30 - Descrição por transferência de registros (RTL)

2.2.9.5 Configuração

Uma declaração de configuração é usada para fazer associações dentro de modelos VHDL. Associa uma arquitetura com uma entidade ou um componente com uma entidade ou arquitetura. Muito utilizada no ambiente de simulação, fornece um caminho rápido e flexível para as alternativas de projeto (ASHENDEN, 1990). A figura 31 apresenta um exemplo de configuração da entidade *Flip_Flop* tipo T da figura 29 com sua arquitetura, para fins de simulação.

```
1  -- Associação da Entidade com a Arquitetura --
2
3  ENTITY TFlip_Flop IS
4      PORT
5      (ck : IN    STD_LOGIC;
6       q  : OUT   STD_LOGIC);
7  END TFlip_Flop;
8
9  ARCHITECTURE comportamental OF TFlip_Flop IS
10     SIGNAL regQ : STD_LOGIC := 0;
11     BEGIN
12       q <= regQ;
13       PROCESS (ck)
14           BEGIN
15              IF (ck'event and ck='1')THEN regQ <= NOT regQ;
16              END IF;
17       END PROCESS
18 END comportamental;
19
20 CONFIGURATION TFlip_Flop_conf OF TFlip_Flop IS
21         FOR comportamental
22         END FOR;
23 END TFlip_Flop;
24
```

Figura 31 - Exemplo de uma declaração de configuração

2.2.9.6 Simulação e Síntese

A linguagem VHDL atinge a sua plenitude em aplicações de simulação, propósito para o qual foi criada (FERNANDES, 1994). Contudo, VHDL também é usada com sucesso em outras áreas de aplicação que não a simulação como, por exemplo, a síntese. Porém, o seu propósito inicial fez com que a semântica para síntese não esteja totalmente definida (FERNANDES, 1994). Uma aplicação de síntese é um processo que se deseja o mais automático possível. Quando integrado num ambiente de projeto de sistemas digitais (EDA), o seu objetivo é transformar uma especificação abstrata numa descrição no nível mais baixo do domínio físico. Essa transformação é feita tendo em consideração as restrições que acompanham o projeto (área, velocidade, tecnologia, consumo de energia, testabilidade[32], ciclo de relógio, etc). O resultado desse processo é normalmente uma *netlist*. A síntese de sistemas digitais pode decompor-se na seguinte seqüência de tarefas de síntese mais simples (FERNANDES, 1994):

- **Síntese de sistema**: o objetivo da síntese de sistema é dividir um sistema em subsistemas, partindo da sua especificação abstrata e mínima. Esses subsistemas, que podem ser vistos como processos concorrentes que se comunicam entre si, são caracterizados pela sua descrição comportamental ao nível de algoritmo.
- **Síntese de alto nível**: a síntese de alto nível é definida como sendo o processo que sintetiza uma estrutura física, a partir de uma

[32] Diversas situações devem ser previstas durante a fase de projeto de um sistema, o que justifica o uso de técnicas de projeto que visam o teste ou a testabilidade do sistema final.

descrição comportamental. O resultado da síntese de alto nível é uma descrição estrutural com duas componentes: uma componente de manipulação de dados (unidades funcionais, unidades de memória e unidades de interligação) e uma componente de controle especificada por um diagrama de transição.

- **Síntese de transferência de registros (RTL): a** síntese a partir de código VHDL concorrente é essencialmente síntese de transferência de registro (RTL). Esse tipo de síntese consiste em mapear os operadores VHDL para componentes primitivos permitidos nas ferramentas a serem usadas posteriormente como, por exemplo, sintetizadores lógicos. Os sistemas comerciais de síntese, na sua grande maioria, situam-se nesse nível.
- **Síntese lógica**: o resultado da síntese RTL, que consiste numa descrição de nível de transferência de registro constituída por blocos de lógica combinatória e elementos de memória, é passado para a síntese lógica. A ênfase da síntese lógica reside na minimização lógica, tendo por objetivo obter a mínima área possível. Essa tarefa de síntese permite abstrair a lógica combinatória da tecnologia a utilizar e do tipo de projeto.
- **Mapeamento na tecnologia**: o mapeamento na tecnologia recebe como entrada uma rede de portas lógicas abstratas e produz um conjunto de células físicas de uma biblioteca de uma dada tecnologia.

Não há atualmente nenhum ambiente que inclua a totalidade das tarefas de síntese anteriormente mencionadas. As sínteses de sistema e alto nível apresentam problemas de formalismo (FERNANDES, 1994).

CAPÍTULO 3 - DESENVOLVIMENTO DO TRABALHO

Este capítulo apresenta uma proposta para implementação de Controlador Lógico Programável (CLP), o qual utiliza na sua arquitetura sistêmica Lógica Programável Estruturada e macroinstruções personalizadas para sua programação. Dessa arquitetura é apresentado o modo de endereçamento de entradas e saídas e a seqüência de execução do programa de aplicação, o ambiente de desenvolvimento, as ferramentas utilizadas para programação do CLP e o modo de desenvolvimento da biblioteca de símbolos gráficos intitulada de PLCPROJECT, analisando cada macroinstrução implementada com sua função lógica, tabela verdade e diagrama funcional.

3.1 CLP Proposto

O Controlador Lógico Programável (CLP) proposto neste trabalho utiliza em sua arquitetura sistêmica Lógica Programável Estruturada, com grande capacidade de processamento. Como visto no item 2.2, do capítulo 2 deste trabalho, foram os Dispositivos Lógicos Programáveis (PLDs) que possibilitaram a implementação da Lógica Programável Estruturada em projeto de controladores digitais.

Neste trabalho, a Lógica Programável Estruturada é representada por FPGA, que suporta a implementação de circuitos lógicos relativamente grandes em um único circuito integrado, permite a reconfiguração do seu *hardware* e utiliza a tecnologia CMOS, que proporciona um baixo consumo de energia elétrica. O objetivo principal da aplicação de Lógica Programável Estruturada no

CLP proposto é melhorar a capacidade computacional e a eficiência em relação aos CLPs tradicionais, otimizando os tempos de processamento e possibilitando a reconfiguração do *hardware* pelo usuário final.

3.1.1 Arquitetura Proposta

- A figura 32 mostra uma proposta de arquitetura para CLPs. As principais diferenças que podem ser observadas entre essa arquitetura proposta e a arquitetura tradicional de CLPs, apresentada na figura 3, no capítulo 2, são:
- A eliminação do bloco CPU (microcontrolador e circuitos de controle).

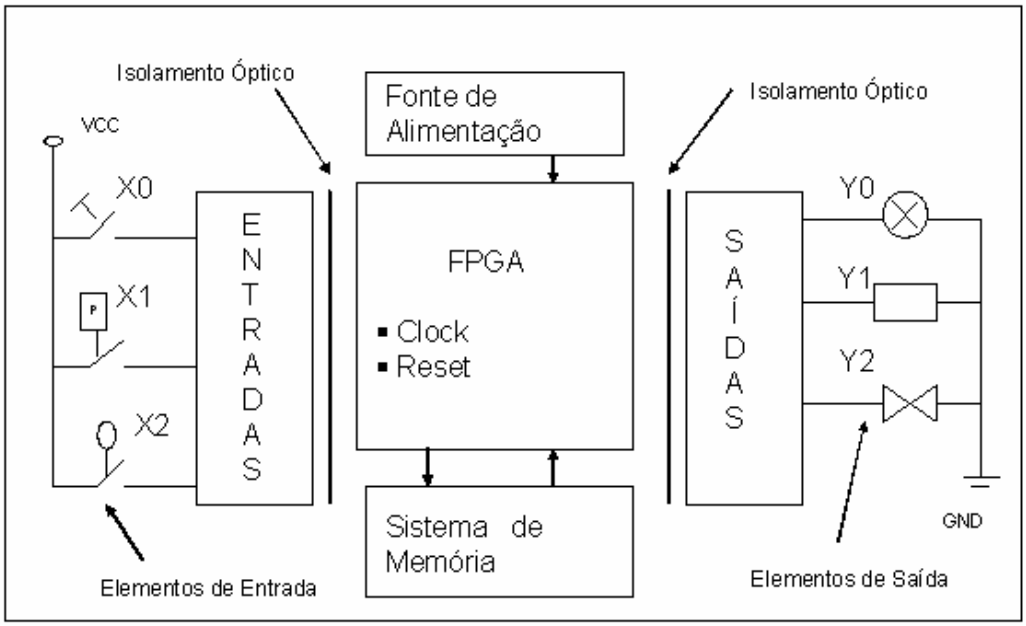

Figura 32 - Proposta de arquitetura para CLP

- A implementação de um bloco FPGA em substituição ao bloco CPU.
- Mudanças na execução do programa de aplicação como: a eliminação dos ciclos de busca de instruções em memória e eliminação dos ciclos de *scan time* ou varredura.

3.1.2 Descrição da Arquitetura Proposta

O CLP proposto, cuja arquitetura é apresentada na figura 32, basicamente é constituído pelos seguintes blocos: dispositivo FPGA, circuitos de entrada, circuitos de saída, memória de configuração e fonte de alimentação.

3.1.2.1 Bloco Dispositivo FPGA

O bloco Dispositivo FPGA introduz mudanças consideráveis no princípio de funcionamento da arquitetura proposta, em relação aos CLPs tradicionais. Entre os mais importantes, podem-se citar:

- Execução do programa de aplicação.
- Endereçamento de entradas e saídas.

3.1.2.1.1 Execução do Programa de Aplicação

O bloco CPU do CLP tradicional, representado na figura 33, executa o programa de aplicação de forma seqüencial, realizando inicialmente a leitura de todos os pontos das entradas. Cada ponto de entrada corresponde a uma posição

de memória específica na tabela imagem das entradas[33]. Inicia a execução do programa de aplicação a partir da primeira linha, executando-o da esquerda para a direita, e de cima para baixo, linha a linha, até encontrar a instrução de FIM (*END*). Constrói, assim, uma nova tabela de imagem de saídas na memória, gerada a partir da lógica executada. Após a execução do programa, o conteúdo da tabela imagem das saídas é enviado aos pontos de saída correspondentes (GEORGINI, 2000).

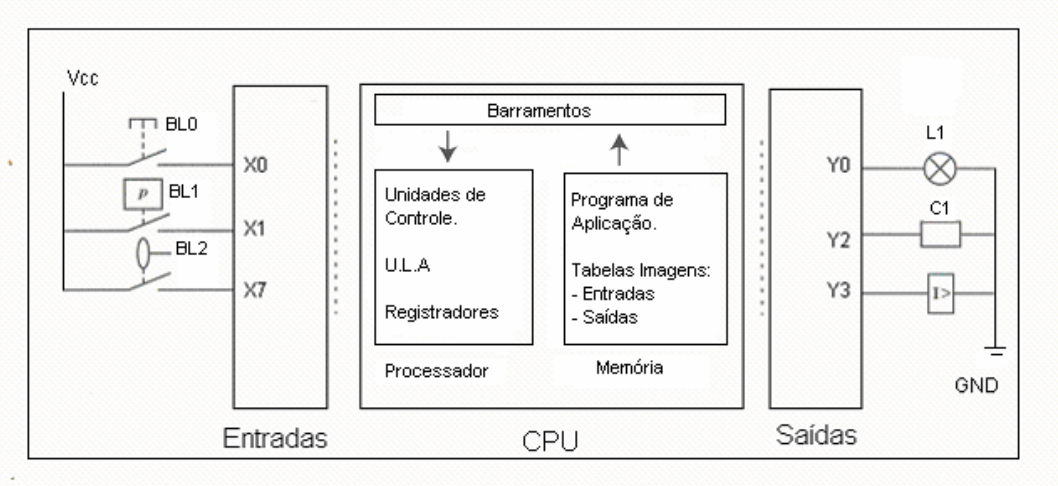

Figura 33 - Bloco CPU do CLP tradicional

No CLP proposto, o bloco Dispositivo FPGA consiste de um grande arranjo de células lógicas ou blocos lógicos contidos em um único circuito integrado, como descrito no capítulo 2, item 2.2.4.1 deste trabalho. Cada uma dessas células lógicas contém uma capacidade computacional para implementar funções lógicas independentes e realizar roteamento para permitir a comunicação entre elas (MILANI, 1998). O programa de aplicação é transformado em funções

[33] Tabela imagem das entradas é o local da memória onde a CPU armazena o estado de todos os pontos de entrada durante o ciclo de varredura.

lógicas, conforme apresentado esquematicamente na figura 34 e descrito no capítulo 2, item 2.2.8 deste trabalho.

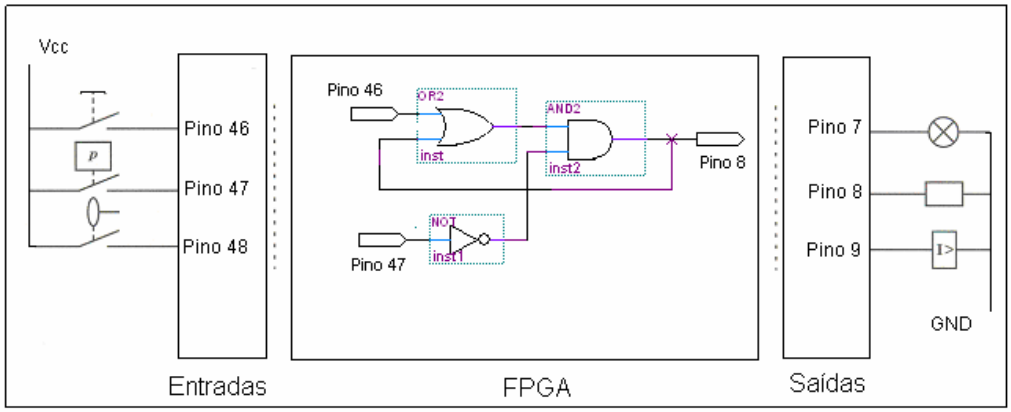

Figura 34 - Programa de aplicação no Bloco FPGA

O FPGA executa o programa de aplicação de forma paralela, considerando cada função lógica como um algoritmo de controle executado por *hardware*. Esse tipo de processamento não demanda tempo com ciclos de varredura (*scan time*), com atualização de tabelas imagens das entradas e saídas, com busca de instruções em memória, etc. Sendo que essas demandas são necessárias na operação dos CLPs tradicionais.

TABELA 5 - Tempos de processamento entre CLPs (I)

CLP Tradicional	**CLP Proposto**
Tempo de Scan = 20 µs	Tempo de execução = 12,9 ηs
Menor tempo de Scan = 20 µs	Menor tempo de execução = 12,9 ηs
Maior tempo de Scan = 30 µs	Maior tempo de execução = 13,4 ηs
Tempo médio de execução por instrução = 1,5 µs	

A tabela 5 mostra uma comparação entre os tempos de processamento para executar uma linha de programa, em um CLP tradicional[34] e o protótipo de CLP estudado[35], com a arquitetura proposta neste trabalho. A linha de programa apresentada na figura 35 é editada em linguagem *Ladder* para o CLP tradicional.

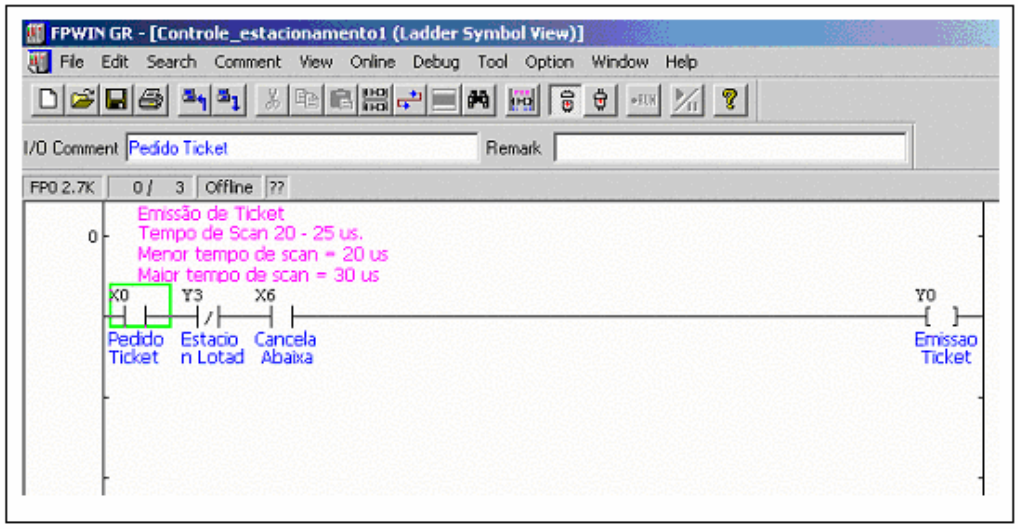

Figura 35 - Linha de programa no CLP tradicional

A figura 36 apresenta a linha de programa editada com as macroinstruções gráficas, desenvolvidas para o CLP proposto.

[34] CLP FP0 fabricado pela empresa Matsushita Electric Works.
[35] Kit de desenvolvimento FPT1 – CPLD/FPGA, baseado no FPGA EPF10K10TC144-4, da família FLEX 10K, fabricado pela empresa ALTERA.

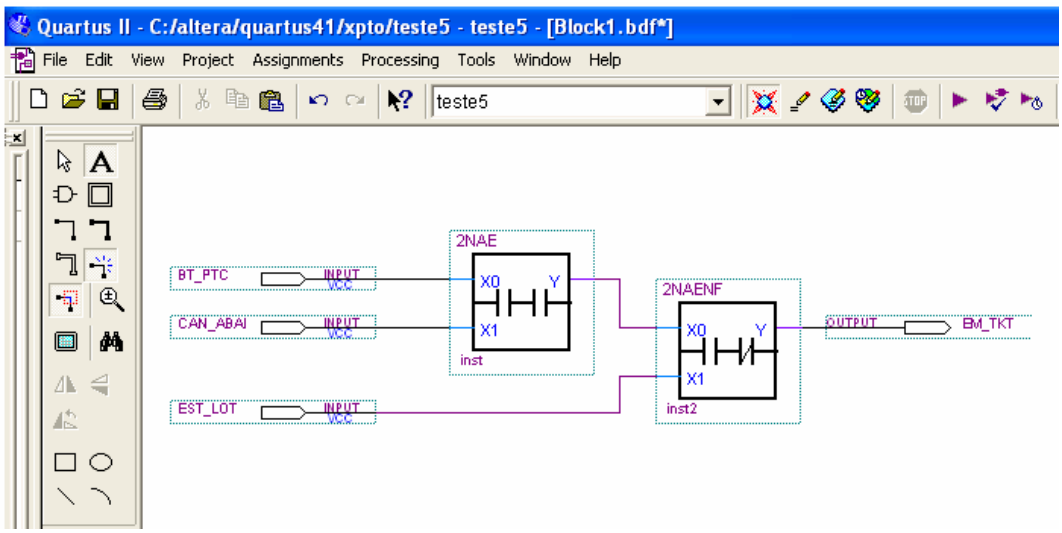

Figura 36 - Linha de programa no CLP proposto

3.1.2.1.2 Endereçamento de Entrada /Saída

Nos CLPs tradicionais, cada ponto de entrada ou saída dos módulos discretos corresponde a um bit de um determinado endereço da tabela de dados na memória (tabela de imagens das entradas e tabela imagem das saídas), a qual é acessada durante a execução do programa de aplicação (GEORGINI, 2000). A figura 37 apresenta um exemplo genérico da relação entre os pontos de entrada e a tabela de imagem das entradas na memória de um CLP tradicional.

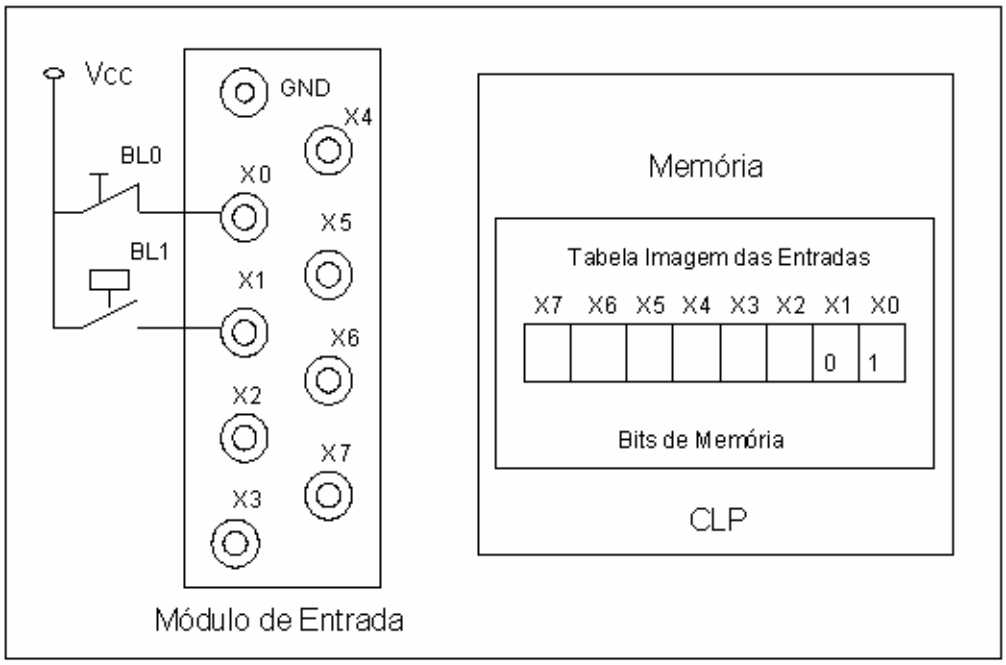

Figura 37 - Tabela de imagem das entradas na memória

A forma de identificação dos pontos de entrada e saída depende do fabricante do CLP. O CLP FP0 da Matsushita identifica os pontos de entrada pela letra “X”, seguida pelo endereço relacionado ao terminal no qual os dispositivos de entrada estão conectados. O endereço X0 corresponde ao bit 0 (zero) da tabela imagem das entradas, terminal número 0 (zero) do módulo de entrada (AROMAT, 2002).

Os pontos de saída são identificados pela letra “Y”, seguida pelo endereço relacionado ao terminal no qual os dispositivos de saída estão conectados. O endereço Y8 corresponde ao bit 8 (oito) da tabela de imagem das saídas, terminal número 8 (oito) do módulo de saída (AROMAT, 2002). A figura 38 mostra, uma

linha de programa *Ladder* com endereços de entrada e saída de um CLP tradicional.

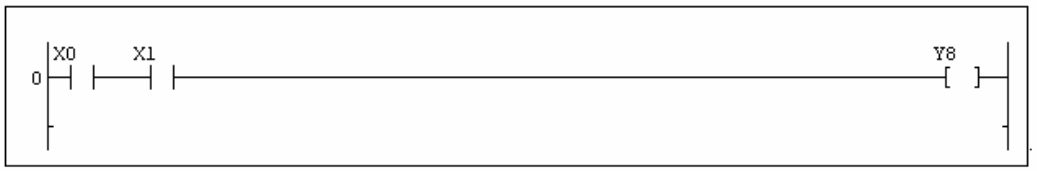

Figura 38 - Programa *Ladder* com endereços de entrada e saída

O CLP proposto não utiliza o conceito de tabela de dados (tabela de imagens das entradas e tabela de imagem das saídas) na memória. O bloco FPGA realiza o processamento do programa de aplicação de forma paralela, diferentemente do CLP tradicional, que executa as instruções do programa de aplicação de forma seqüencial, consultando as tabelas de imagens de entrada e de saída na memória.

No bloco FPGA, o programa de aplicação é transformado em circuitos lógicos, implementados por *hardware* no próprio dispositivo numa localização específica. Os elementos de entrada, tais como sensores, botões, etc., não estão relacionados com nenhum bit de memória. São variáveis de entrada de circuitos lógicos, que estão conectados diretamente aos pinos de entrada do FPGA.

A definição do nome dos terminais dos circuitos de entrada ou saída do CLP proposto é realizada pelo usuário durante a edição do programa de aplicação. A seleção dos pinos de entrada e saída é feita automaticamente, pelo *software* QUARTUS II, descrito no item 2.2.8.1.1 deste trabalho, durante a compilação do

programa de aplicação em função da família do dispositivo selecionado, ou manualmente pelo usuário.

As tabelas 6 e 7 apresentam a listagem dos sinais de entrada e saída da primeira versão do protótipo do CLP, proposto neste trabalho, em função dos pinos de entrada e saída do dispositivo FLEX10K10, utilizado no projeto.

TABELA 6 - Endereçamento dos circuitos de entradas

Nome do Sinal	**Pinos do EPF10K10TC144-4 (*)**
X0	Pino 46
X1	Pino 47
X2	Pino 48
X3	Pino 49
X4	Pino 51
X5	Pino 59
X6	Pino 60
X7	Pino 62

TABELA 7 - Endereçamento dos circuitos de saídas

Nome do Sinal	**Pinos do EPF10K10TC144-4 (*)**
Y0	Pino 7
Y1	Pino 8
Y2	Pino 9
Y3	Pino 10
Y4	Pino 11
Y5	Pino 12
Y6	Pino 13
Y7	Pino 14

(*) Definido pelo usuário durante a edição do programa no *software* QUARTUS II.

3.1.2.1.3 *Kit* de Desenvolvimento FPT1

O *kit* de desenvolvimento FPT1 (LEAP, 2002), da empresa Leap Electronic Co, foi o *hardware* utilizado como base para efetuar a implementação da arquitetura do CLP proposto neste trabalho. A vista superior desse *kit* pode ser observada na figura 39.

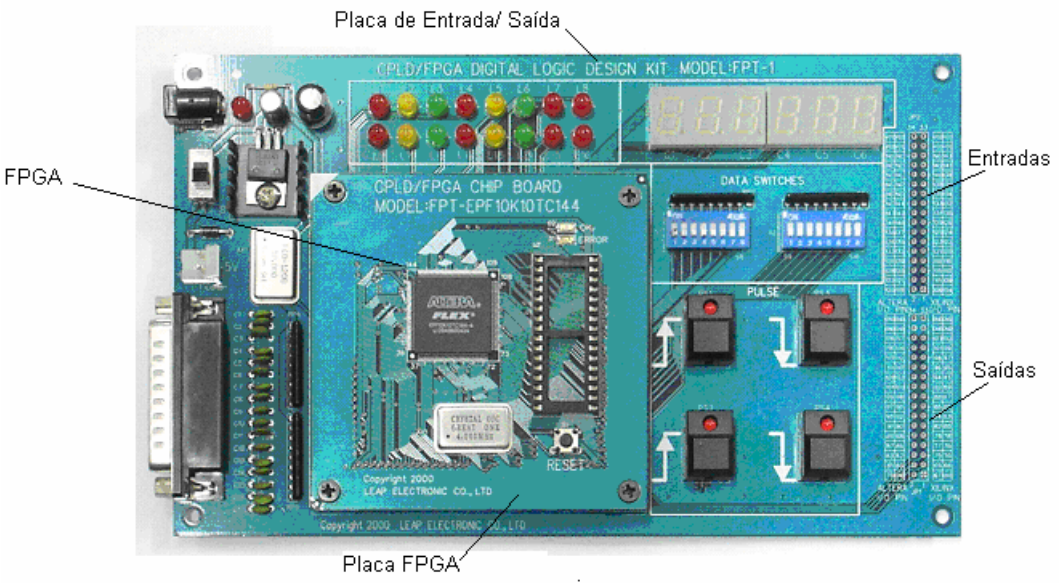

Figura 39 - *Kit* de desenvolvimento FPT 1

O *kit* FPT 1 é constituído por duas placas de circuito impresso: placa do dispositivo FPGA e placa dos dispositivos de E/S.

3.1.2.1.3.1 Placa do Dispositivo FPGA

A figura 40 mostra a placa do dispositivo FPGA, que inclui um circuito FPGA, fabricado pela empresa Altera, série EPF10K10TC144-4, um soquete para

memória de configuração EPROM (*Erasable Programmable Read Only Memory*), um botão de *Reset* e um conjunto de LEDs (*Light Emiter Diodes*) SMD[36] para sinalização dos estados dos pinos de entrada e saída do FPGA.

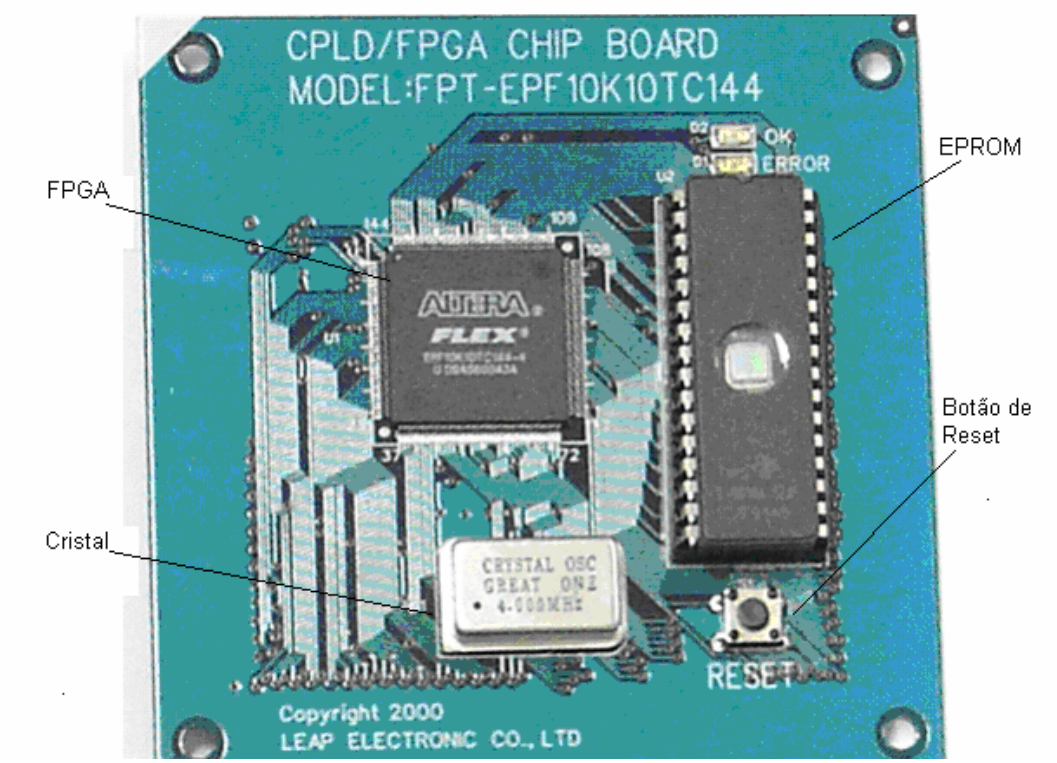

Figura 40 - Placa do dispositivo FPGA

O FPGA que acompanha o *kit* de desenvolvimento FPT 1 possui 144 pinos de entrada e saída, 10.000 portas lógicas, 576 Elementos Lógicos (LEs), 72 Blocos de Matrizes Lógicas (LABs), 3 Blocos de Memória Embutida (EABs) e 6.144 *bits* de memória RAM (*Random Acess Memory*) (ALTERA, 1998).

[36] Dispositivo de Montagem em Superfície (SMD – *Surface Mount Device*).

3.1.2.1.3.2 Placa de Dispositivos de Entrada e Saída

Existem 42 (quarenta e dois) dispositivos de entrada e saída no *kit* FPT 1, sendo 2 (dois) conjuntos de 8 (oito) *LEDs* (Vermelho, Amarelo e Verde), 6 (seis) *Displays* de 7 (sete) segmentos, 2 (dois) conjuntos de 8 (oito) Chaves *Dip Switches*, 4 (quatro) Chaves de pulso *Push Bottom* e uma entrada de *Clock*.

A figura 39 apresenta o *kit* FPT 1 e os seus dispositivos de entradas e saídas mais comuns, utilizados em circuitos lógicos, o que possibilitou o ambiente adequado para simulação e testes do protótipo de CLP.

O *kit* FPT 1 permite transferências de programas provenientes do microcomputador padrão IBM PC, através de sua porta paralela. Para tal, utiliza um cabo padrão *Centronics*, para impressora paralela, interligando o microcomputador PC ao *kit* FPT 1.

O *kit* FPT 1, utilizado com o *software* QUARTUS II, forneceu um ambiente integrado adequado para o projeto, simulação e verificação do CLP protótipo baseado em FPGA.

3.1.2.1.4 O Ambiente de Desenvolvimento Quartus II

O *software* de desenvolvimento QUARTUS II foi utilizado na programação, na simulação e nos testes do protótipo. Esse *Software* suporta soluções em nível de sistema com editoração de blocos, definição de pinos de entrada e saída, e um avançado suporte para macrofunções. Além disso, possui um sistema de análise lógica embutido, o qual permite ao usuário testar a funcionalidade e a temporização do dispositivo FPGA, por exemplo, observando

os valores de sinais internos à velocidade de *clock* do sistema. O *software* QUARTUS II é um ambiente que reúne os elementos necessários para o desenvolvimento de lógica de controle para CLPs baseado em Lógica Programável Estruturada.

3.1.2.2 Bloco de Circuitos de Entrada

Os blocos de circuitos de entrada podem ser classificados como discretos (digitais) ou analógicos. Neste trabalho, será tratado apenas do discreto.

Neste trabalho os blocos de circuitos de entrada são constituídos por oito entradas discretas, sinal digital (ligado/ desligado). Recebe sinais de dispositivos de entrada, tais como: sensores, chaves e transdutores, e os converte em níveis adequados de tensão, por exemplo, 5 Vdc para serem processados pela lógica de controle, configurada no dispositivo FPGA.

O bloco de circuito de entrada apresentada na figura 41 é dotado de isolação[37] óptica para proteção do FPGA, fonte de alimentação e demais componentes como diodos de proteção, capacitor de filtragem e resistores limitadores de corrente. Nesse caso, não há conexão elétrica entre os dispositivos de entrada (chaves, sensores, etc) e os pinos de entrada do FPGA. A figura 41 mostra uma configuração típica de uma entrada digital discreta do protótipo testado.

[37] Consiste de uma fonte de luz (diodo emissor de luz) que é atuado por um sinal elétrico de entrada, transferindo-o na forma de sinal luminoso para outro dispositivo sensor de luz (foto transistor), que o converte para sinal elétrico em sua saída. Dessa forma, protege por meio de isolamento ótico, os elementos internos do circuito eletrônico.

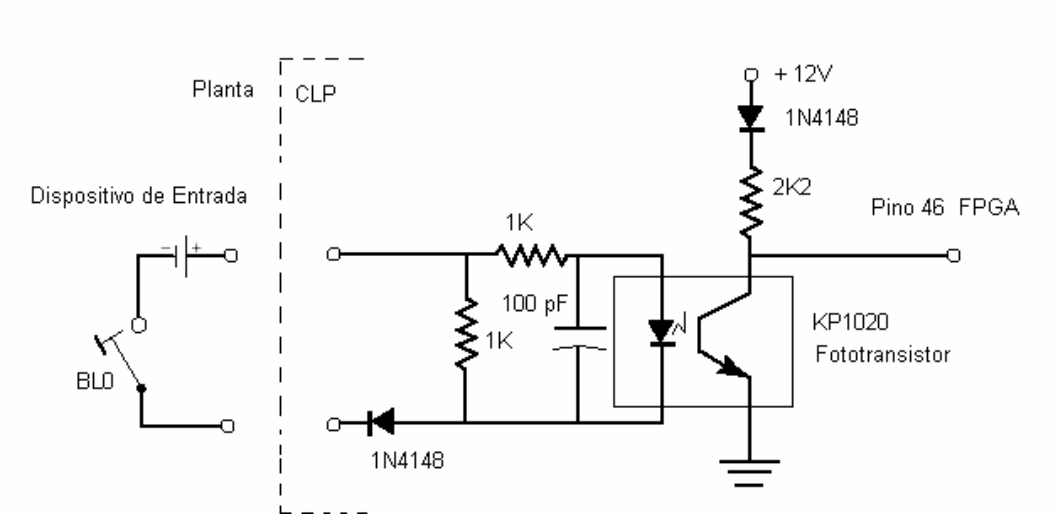

Figura 41 - Configuração típica de uma entrada

3.1.2.3 Bloco de Circuitos de Saída

Neste trabalho, o bloco de circuito de saída discreta é constituído por oito circuitos de saída. Esse envia os sinais aos dispositivos de saída, tais como: contatores, solenóides, relés, sinalizadores, etc. Esses sinais são resultantes da lógica de controle, pela execução do programa de aplicação. O bloco de circuitos de saída também é dotado de isolação óptica. Nesse caso, também não há conexão elétrica entre os dispositivos de saída e os pinos do FPGA. Neste trabalho serão utilizados oito circuitos de saída baseados em relés, com um contato normalmente aberto, conforme mostrado na figura 42.

Uma medida importante que deve ser tomada quando se utilizam saídas a relé, para acionamento de cargas indutivas, é a utilização de circuitos de proteção RC (resistor e capacitor) chamado *Snubber*. Esses circuitos de proteção devem ser utilizados em tensões alternadas e contínuas. Para tensões contínuas

recomenda-se a utilização de diodos de proteção *Schottky,* devido à alta velocidade de comutação, para proteção dos contatos contra tensões inversas (GEORGINI, 2000).

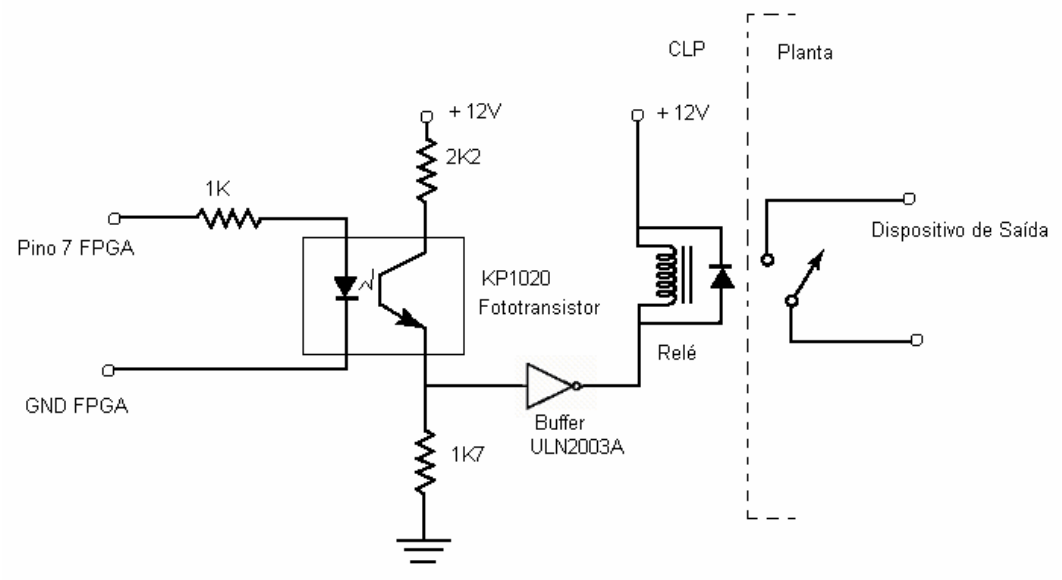

Figura 42 - Configuração típica de uma saída

3.1.2.4 Bloco de Memória de Configuração

A série FLEX10K10 é baseada em tecnologia de programação SRAM (*Static Random Acess Memory*), em que os dados de configuração devem ser transferidos toda a vez que o dispositivo FPGA for iniciado (ALTERA, 1998).

A função do bloco de memória de configuração é armazenar o programa de configuração elaborado pelo usuário e transferi-lo para o dispositivo FPGA. O CLP proposto possui dois modos de transferência de programação (*download*):

- **Microcomputador padrão IBM PC**: a transferência da configuração pode ser realizada diretamente do microcomputador,

por meio da porta de comunicação paralela (LPT1), para a placa que contem o dispositivo FPGA, interface JTAG (*Joint Test Action Group*), descrita no item 2.2.5 deste trabalho, utilizando-se um cabo padrão *Centronics* para impressora, por exemplo. A transferência é parametrizada e controlada pelo *software* QUARTUS II.

- **EPROM Paralela**: a transferência da configuração é realizada via memória EPROM (*Erasable Programmable Read Only Memory*). Por Exemplo, a memória EPROM 2764, disponível no bloco de memória de configuração, é a maneira mais fácil e rápida de se configurar o dispositivo FPGA (LEAP, 2002). A programação da EPROM 2764 é realizada pelo *software* QUARTUS II, depois que o programa de aplicação é totalmente compilado. A transferência da configuração da EPROM para o FPGA é automática, ocorrendo toda a vez que o CLP proposto for iniciado.

3.1.2.5 Bloco da Fonte de Alimentação

O bloco da fonte de alimentação desempenha importante papel na operação do sistema do CLP proposto. Fornece todos os níveis de tensão para alimentação dos blocos FPGA, circuitos de entrada e saída, e memória de configuração. As faixas de tensão de operação são:

- Entrada: AC (85 – 132V) ou (170 – 264V).
- Saída: DC (5V – 2A) e (24V – 2A).

A saída 5Vdc fornece tensão para alimentação dos componentes internos do CLP e possui proteção contra curto-circuito. A saída 24Vdc proporciona uma tensão auxiliar para alimentar os dispositivos de campo (sensores, etc.) e possui proteção contra curtos-circuitos.

3.1.2.6 Protótipo do CLP

O protótipo do CLP que foi elaborado para atender a segunda fase dos ensaios práticos pertinentes a este trabalho é apresentado na figura 43. Esse protótipo utiliza uma placa de circuito impresso na qual está instalado o dispositivo FPGA, descrito no item 3.1.2.1.3.1, uma placa de montagem dedicada (*protoboard*[38]) na qual está montada a interface de entrada e a interface de saída, baseadas nos circuitos descritos nos itens 3.1.2.2 e 3.1.2.3 deste trabalho.

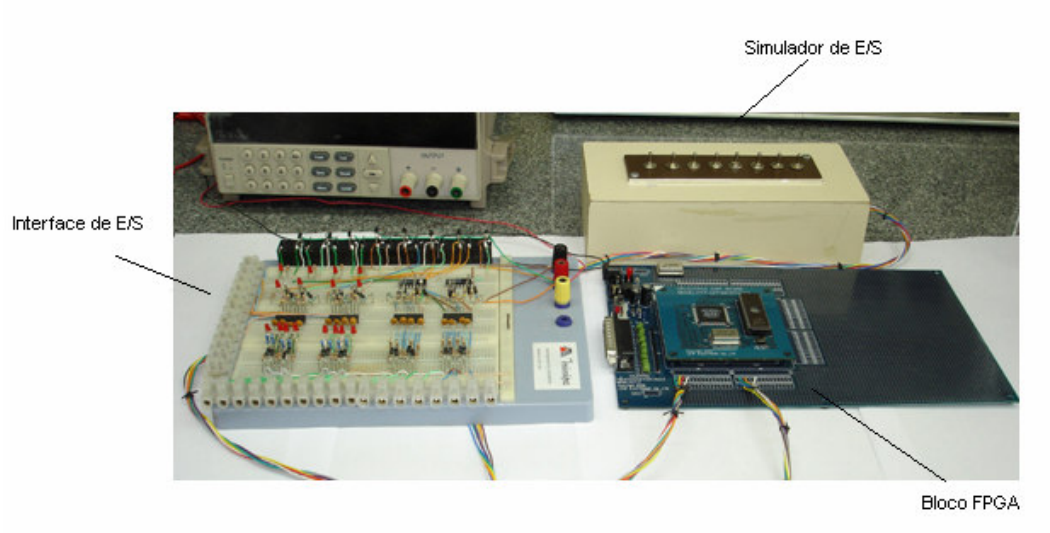

Figura 43 - Protótipo do CLP proposto

[38] Placa de montagem do tipo matriz de pontos, onde os componentes podem ser fixados e interligados.

3.2 Ferramentas para Programação do CLP Proposto

A principal ferramenta utilizada para a programação do CLP proposto é o *software* QUARTUS II. A figura 43 mostra o diagrama de blocos que representa as fases pertencentes ao ambiente de desenvolvimento desse *software*. Esse ambiente foi descrito no item 2.2.8 do capítulo 2, deste trabalho.

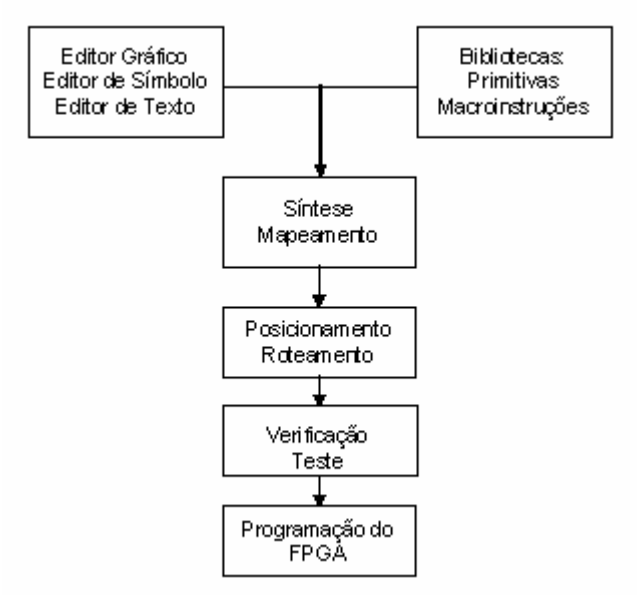

Figura 44 - Diagrama de bloco do ambiente de *software* QUARTUS II

3.2.1 O *Software* QUARTUS II

Vários *softwares* para a configuração de FPGAs estão disponíveis no mercado nacional, como por exemplo, o *software* Viewsim desenvolvido pela empresa Mentor Graphics e o *software* Synopsys desenvolvido pela empresa Synopsys. Neste trabalho foi utilizado o *software* QUARTUS II, da empresa Altera Corp., para o desenvolvimento de macroinstruções de programação do CLP

proposto. A escolha do *software* QUARTUS II levou em consideração vários fatores, dos quais se destacam os seguintes:

- **Editor Gráfico**: permite editar o programa por meio de diagramas lógicos, conectando portas lógicas, multiplexadores, registradores, decodificadores, *flip–flops*, comparadores, somadores, funções especiais, etc. Todos os componentes utilizados estão contidos em bibliotecas: primitiva e macroinstruções fornecidas pelo *software*, ou definidas pelo próprio usuário.
- **Editor de Símbolo**: permite ao usuário definir sua própria biblioteca de macroinstruções e editar o programa no modo Editor Gráfico.
- **Editor de Texto**: permite editar o programa em linguagens de descrição de *hardware* (*Hardware Description Language* - HDL) como: VHDL, Verilog e AHDL.
- **Verificação e Simulação**: permite uma análise bastante detalhada dos resultados de um programa, por meio da observação de diagramas funcionais e temporais de entradas e saídas definidos no editor gráfico.
- **Linguagem JAM STAPL (*Standard Test and Programming Language*)**: gera o código fonte, arquivo *JAM STAPL*, para a programação direta do FPGA durante a operação normal do sistema, através de uma interface chamada JTAG, descrita no capítulo 2, item 2.2.5. Essa tecnologia é conhecida como ICR (*In* -

Circuit Reconfigurable), que permite a configuração, reconfiguração e testes do FPGA, utilizando a porta paralela ou serial de um microcomputador pessoal padrão IBM PC.

3.2.2 Linguagem *Ladder*

Mesmo tendo sido a primeira linguagem destinada especificamente à programação de CLPs, a linguagem *Ladder* mantém-se ainda como a mais utilizada, estando presente praticamente em todos os CLPs atuais. Por ser uma linguagem gráfica baseada em símbolos gráficos (semelhantes aos utilizados nos esquemas de comandos elétricos), a linguagem *Ladder* é facilmente assimilada pelos usuários finais (GEORGINI, 2000).

3.2.2.1 Compilador para a Linguagem *Ladder*

Como resultado da pesquisa realizada neste trabalho, considerava-se a possibilidade de se desenvolver um módulo de compilação que recebesse uma entrada escrita numa linguagem fonte (*Ladder*) e codificasse uma saída numa linguagem destino (VHDL), correspondente ao controlador FPGA.

A primeira decisão consistiu em pesquisar a existência de um *software* que permitisse a compilação da linguagem *Ladder* para VHDL e, dessa forma, permitisse ao usuário final programar o CLP proposto em linguagem *Ladder* e transferir a configuração, reconfiguração e testes, diretamente para o dispositivo

FPGA via portas paralela, serial ou USB[39] de um microcomputador pessoal padrão IBM PC.

Durante o estudo realizado não foi encontrado o *software* mencionado, e seu desenvolvimento completo é um trabalho de longo prazo, o qual poderia ser realizado numa segunda etapa de melhorias do CLP proposto. Então, optou-se por escolher um ambiente integrado de desenvolvimento com as seguintes características:

- Permitir o desenvolvimento em curto prazo.
- Possuir portabilidade da aplicação gerada.

A escolha recaiu no *software* QUARTUS II. Esse *software* permitiu o desenvolvimento de uma biblioteca de macroinstruções, baseada em símbolos gráficos. Para tal, utilizou-se o Editor de Símbolos, adequado para criar os símbolos gráficos equivalentes aos utilizados nas instruções da linguagem *Ladder*.

3.2.3 Biblioteca PLCPROJECT

A filosofia adotada no desenvolvimento da biblioteca PLCPROJECT partiu do princípio que as macroinstruções gráficas implementadas na nova biblioteca deveriam ser desenvolvidas a partir de funções booleanas disponíveis nas bibliotecas primitivas do *software* QUARTUS II, utilizando-se o modo Editor de Símbolos para criá-las. As macroinstruções representam alguns símbolos da linguagem *Ladder*, por exemplo, contatos de relés e cada símbolo gráfico criado possui a sua função booleana correspondente.

[39] O Barramento USB (*Universal Serial Bus*) é serial e comporta até 127 dispositivos ligados por meio de conectores de expansão do próprio barramento ou por meio de hubs USB.

Dessa forma, o programa de aplicação pode ser elaborado no Editor Gráfico, utilizando-se as macroinstruções da biblioteca PLCPROJECT, e o *software* QUARTUS II gera o código fonte, arquivo *JAM STAPL* para a configuração direta do dispositivo FPGA.

3.2.4 Instruções da Biblioteca PLCPROJECT

A biblioteca PLCPROJECT é constituída por instruções do tipo relé e instruções de temporização e contagem.

As instruções mais avançadas de manipulação de dados, aritméticas e transferência de dados serão desenvolvidas numa terceira fase de melhorias do CLP proposto.

3.2.4.1 Instruções do Tipo Relé

As instruções do tipo relé permitem examinar o estado (ligado /desligado) de um dispositivo de entrada, da seguinte maneira:

- **Contato Normalmente Aberto (NA):** a instrução está associada a um dispositivo de entrada (pino do FPGA). Se a sua condição for ligada, então esse contato se fecha e assegura a continuidade lógica entre a entrada e saída da instrução. Se a condição for desligada, então o contato permanece aberto interrompendo a continuidade lógica entre a entrada e a saída da instrução. A figura 45 apresenta o símbolo gráfico *Ladder* da instrução NA e a sua função lógica correspondente.

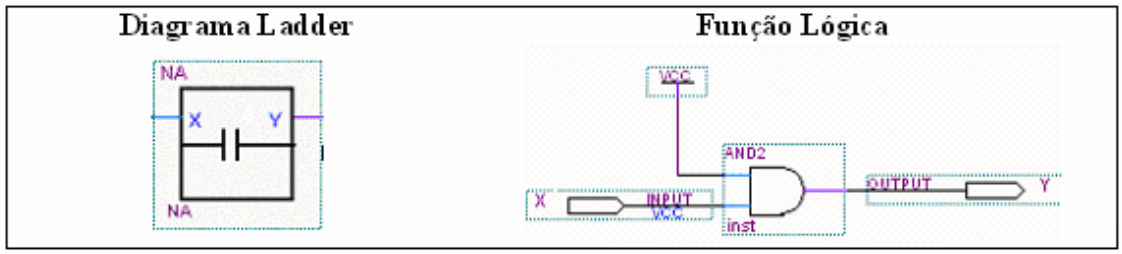

Figura 45 - Símbolo gráfico e função lógica da instrução NA

A figura 46 mostra o diagrama funcional da instrução NA, nesse pode-se observar que o dispositivo de entrada (X) transfere o seu estado diretamente para a saída (Y). Ou seja, se a entrada (X) é ativada, a saída (Y) é ativada e se a entrada (X) é desativada, a saída (Y) é desativada.

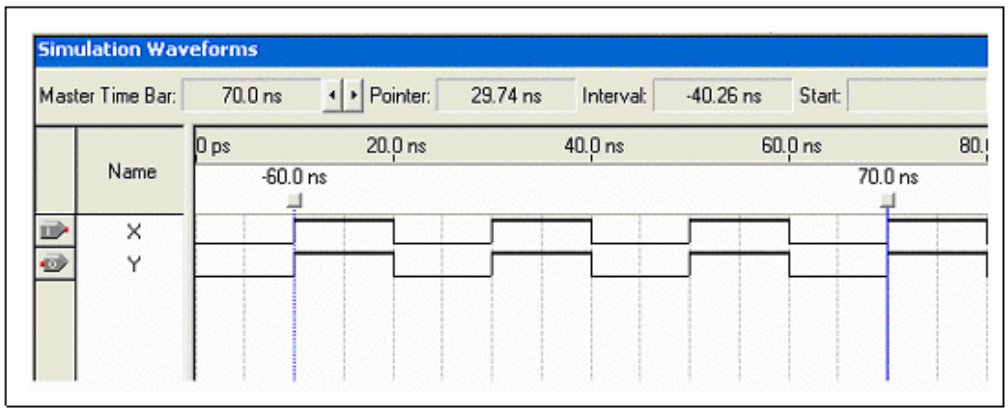

Figura 46 - Diagrama funcional da instrução NA

- **Contato Normalmente Fechado (NF):** a instrução está associada a um dispositivo de entrada (pino do FPGA). Se a sua condição for desligada, então esse contato permanece fechado e assegura a continuidade lógica entre a entrada e a saída da instrução. Quando

a condição for ligada, o contato abre-se interrompendo a continuidade lógica entre a entrada e a saída da instrução.

A figura 47 apresenta o símbolo gráfico *Ladder* da instrução NF e a sua função lógica correspondente.

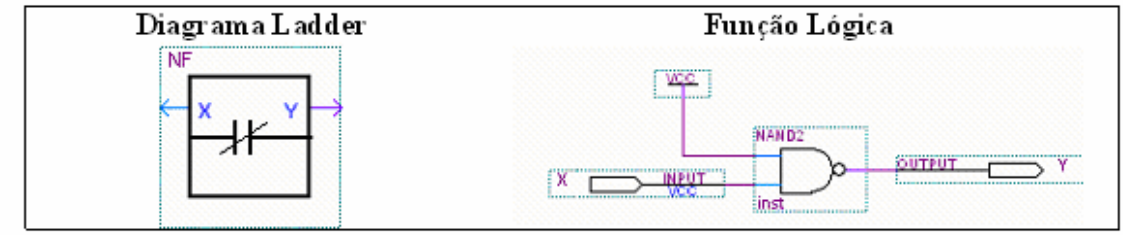

Figura 47 - Símbolo gráfico e função lógica da instrução NF

A figura 48 mostra o diagrama funcional da instrução NF. Nesse pode-se observar que o dispositivo de entrada (X) transfere o inverso do seu estado para a saída (Y), ou seja, se a entrada (X) é ativada, a saída (Y) é desativada e se a entrada (X) é desativada, a saída (Y) é ativada.

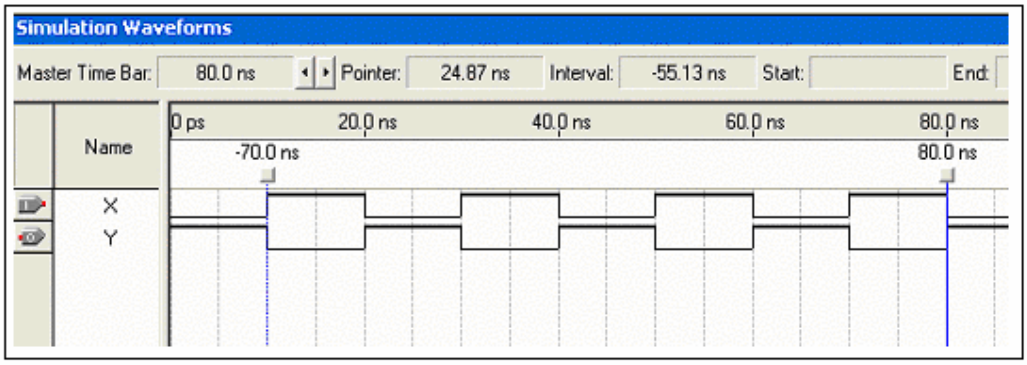

Figura 48 - Diagrama funcional da instrução NF

3.2.4.1.1 Associação de Instruções do Tipo Relé

As instruções do tipo relé resumem-se em Contato Normalmente Aberto (NA) e Contato Normalmente Fechado (NF). Porém, a associação dessas instruções define instruções distintas:

- **Instrução 2NAE**: essa instrução executa a lógica *AND*[40] entre dois Contatos Normalmente Abertos em série. A instrução está associada com dois dispositivos de entrada X0 e X1, conforme apresentado na figura 49 e na tabela verdade da tabela 8.

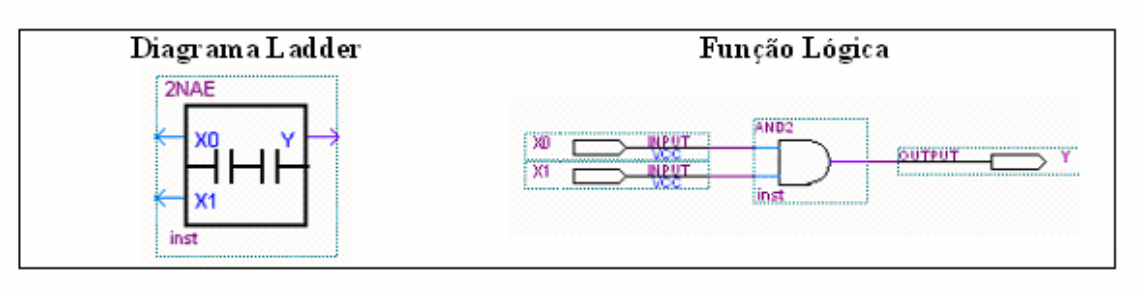

Figura 49 - Símbolo gráfico e função lógica da instrução 2NAE

A figura 49 apresenta o símbolo gráfico *Ladder* da instrução 2NAE e a sua função lógica correspondente.

[40] Operação Booleana que executa o produto entre duas variáveis de entrada.

Tabela Verdade da Instrução 2NAE

X0	X1	Y(Continuidade Lógica)
Desligado	**Desligado**	**Não**
Desligado	**Ligado**	**Não**
Ligado	**Desligado**	**Não**
Ligado	**Ligado**	**Sim**

A figura 50 apresenta o diagrama funcional da instrução 2NAE. Pode-se verificar que, quando os contatos X0 e X1 forem ligados, então esses contatos se fecham e asseguram a continuidade lógica. Se um dos contatos for desligado, o contato abre interrompendo a continuidade lógica da associação em série.

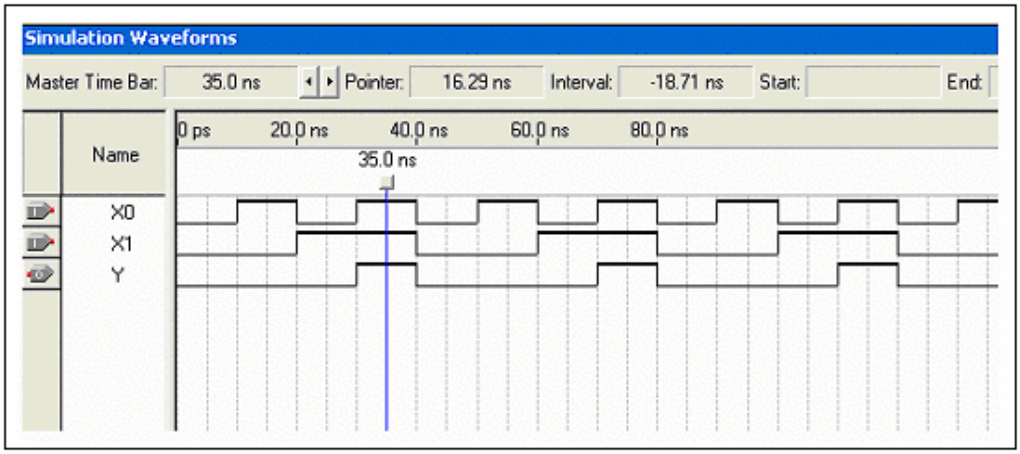

Figura 50 - Diagrama funcional da instrução 2NAE

No decorrer dos testes funcionais da instrução 2NAE realizados no CLP protótipo, observou-se a dificuldade de se obter essa instrução a partir da associação de dois blocos NA em série. Tal fato decorreu em virtude do CLP proposto não utilizar a mesma técnica de execução do programa em memória, como o CLP tradicional, apresentado no item 3.1.2.1.1 deste trabalho.

No CLP tradicional, por exemplo, uma associação de dois contatos em série (lógica *AND*) é realizada em memória com dois bits da tabela imagens das entradas, que representam os estados lógicos dos contatos. A figura 51 apresenta uma associação em série de contatos NA, lógica *AND*, em um CLP tradicional.

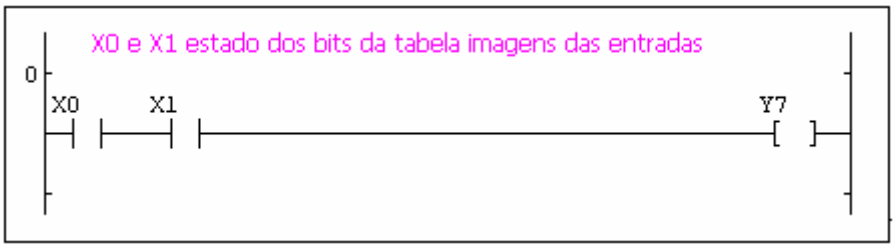

Figura 51 - Lógica *AND* no CLP tradicional

No CLP proposto, a associação em série de dois Contatos Normalmente Abertos, lógica *AND*, não é realizada em memória. A operação é realizada em *hardware* nas células internas do dispositivo FPGA. Assim, a associação em série de dois blocos NA pode ser substituída por um único bloco lógico AND de duas entradas, cujas entradas são pinos do FPGA. Dessa forma, economiza-se células lógicas do FPGA, após a síntese do programa de aplicação. A figura 52 apresenta o bloco lógico *AND* (instrução 2NAE) proposto em substituição à associação em série de duas instruções NA.

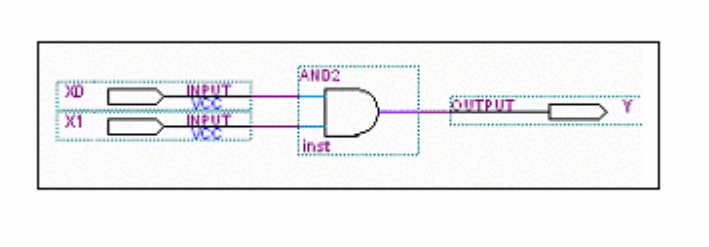

Figura 52 - Lógica *AND* no CLP proposto

Na figura 53 é apresentada associação de dois blocos NA em série, configuração não recomendável para ser implementada no CLP proposto, em virtude de utilizar mais células internas do FPGA, após a síntese do programa de aplicação.

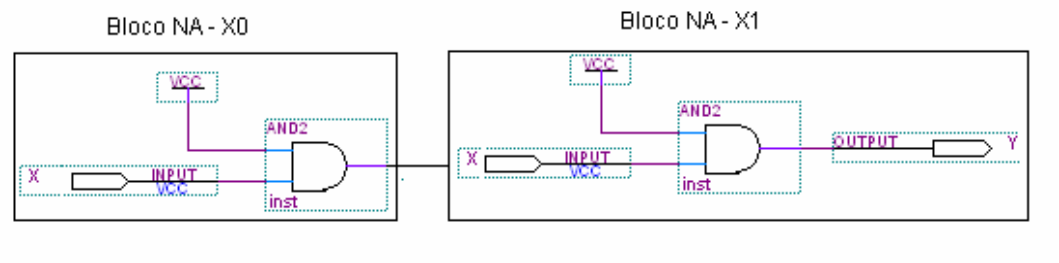

Figura 53 - Associação de dois blocos NA

- **Instrução 2NAOU**: essa instrução executa a lógica OR[41] entre dois Contatos Normalmente Abertos em paralelo. A instrução está associada a dois dispositivos de entrada X0 e X1, conforme apresentado na figura 54 e a tabela verdade da tabela 9.

[41] Operação Booleana que executa a soma entre duas variáveis de entrada.

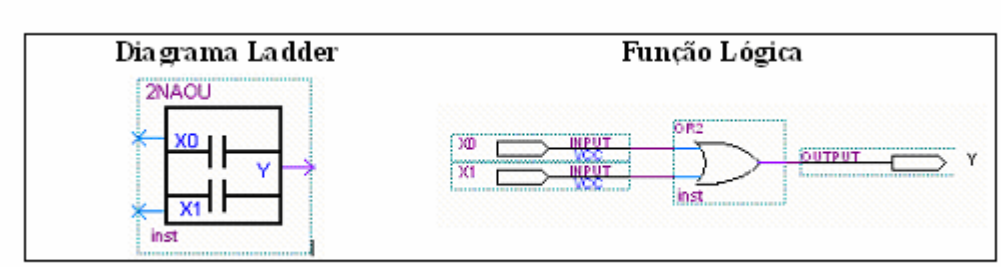

Figura 54 - Símbolo gráfico e função lógica da instrução 2NAOU

A figura 54 apresenta o símbolo gráfico *Ladder* da instrução 2NAOU e a sua função lógica correspondente.

TABELA 8 - Tabela Verdade da Instrução 2NAOU

X0	X1	Y(Continuidade Lógica)
Desligado	**Desligado**	**Não**
Desligado	**Ligado**	**Sim**
Ligado	**Desligado**	**Sim**
Ligado	**Ligado**	**Sim**

A figura 55 mostra o diagrama funcional da instrução 2NAOU, pode-se observar que, se pelo menos um de seus contatos for ligado, então este contato se fecha e assegura a continuidade lógica. Se os dois contatos forem desligados, os contatos abrem-se interrompendo a continuidade lógica da associação em paralelo.

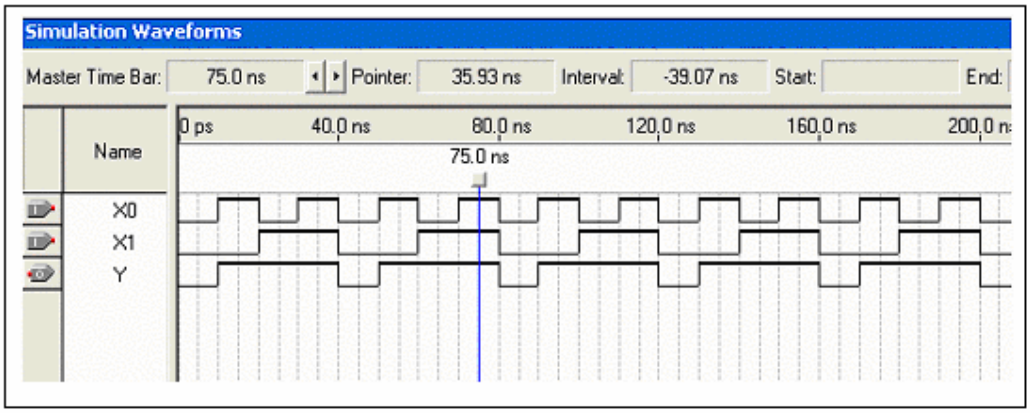

Figura 55 - Diagrama funcional da instrução 2NAOU

- **A instrução 2NFE**: essa instrução executa a lógica *AND* entre dois contatos normalmente fechados em série. A instrução está associada a dois dispositivos de entrada X0 e X1, conforme apresentado na figura 56 e a tabela verdade da tabela 10.

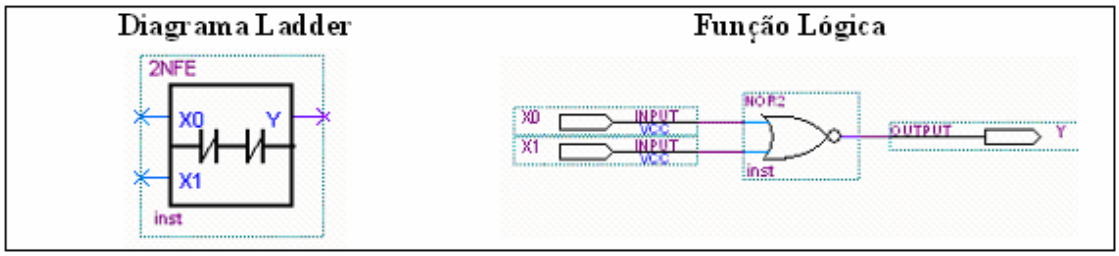

Figura 56 - Símbolo gráfico e função lógica da instrução 2NFE

A figura 56 apresenta o símbolo gráfico *Ladder* da instrução 2NFE e a sua função lógica correspondente.

TABELA 9 - Tabela Verdade da Instrução 2NFE

X0	X1	Y(Continuidade Lógica)
Desligado	**Desligado**	**Sim**
Desligado	**Ligado**	**Não**
Ligado	**Desligado**	**Não**
Ligado	**Ligado**	**Não**

A figura 57 mostra o diagrama funcional da instrução 2NFE, pode-se observar que, se os contatos forem desligados, então existe continuidade lógica. Se pelo menos um dos contatos for ligado, interrompe-se a continuidade lógica da associação em série.

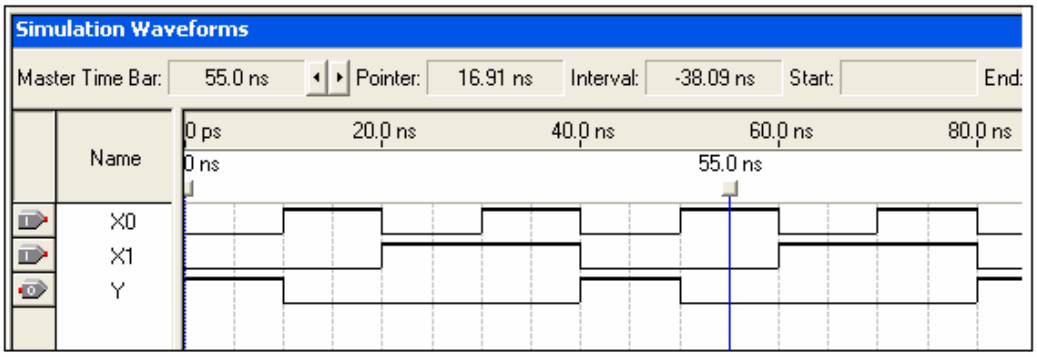

Figura 57 - Diagrama funcional da instrução 2NFE

- **Instrução 2NFOU**: essa instrução executa a lógica *OR* entre dois Contatos Normalmente Fechados em paralelo. A instrução está associada a dois dispositivos de entrada X0 e X1, conforme apresentado na figura 58 e tabela verdade da tabela 11.

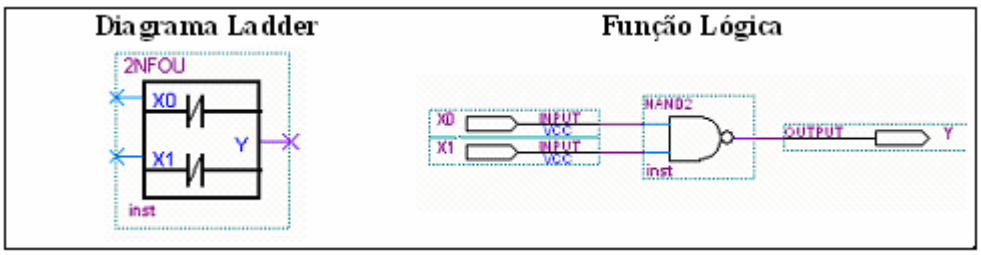

Figura 58 - Símbolo gráfico e função lógica da instrução 2NFOU

A figura 58 apresenta o símbolo gráfico *Ladder* da instrução 2NFOU e a sua função lógica correspondente.

TABELA 10 - Tabela Verdade da Instrução 2NFOU

X0	X1	Y(Continuidade Lógica)
Desligado	**Desligado**	**Sim**
Desligado	**Ligado**	**Sim**
Ligado	**Desligado**	**Sim**
Ligado	**Ligado**	**Não**

A figura 59 mostra o diagrama funcional da instrução 2NFOU. Pode-se observar que, se pelo menos um de seus contatos for desligado, então existe continuidade lógica. Se os dois contatos forem ligados, a continuidade lógica da associação em paralelo é interrompida.

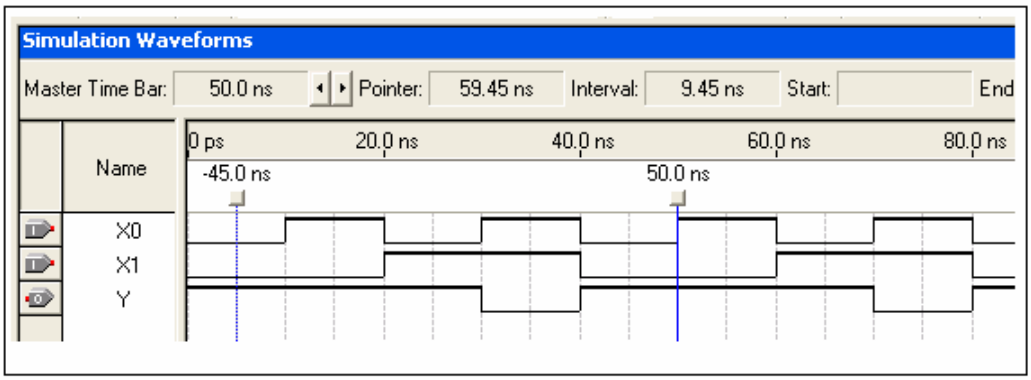

Figura 59 - Diagrama funcional da instrução 2NFOU

- **Instrução 2NAENF**: essa instrução executa a lógica *AND* entre um Contato Normalmente Aberto em série a outro Contato Normalmente Fechado. A instrução está associada a dois dispositivos de entrada X0 e X1, conforme apresentado na figura 60 e tabela verdade da tabela 12.

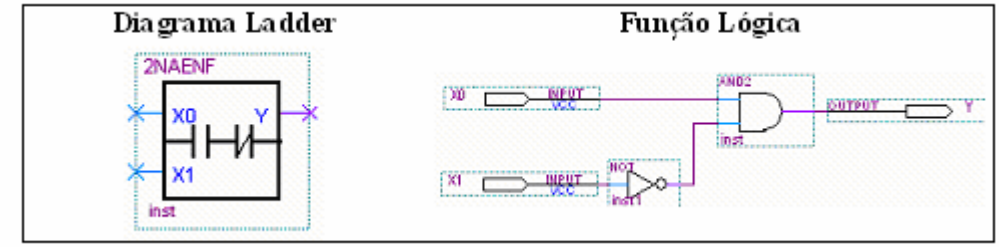

Figura 60 - Símbolo gráfico e função lógica da instrução 2NAENF

A figura 60 apresenta o símbolo gráfico *Ladder* da instrução 2NAENF e a sua função lógica correspondente.

TABELA 11 - Tabela Verdade da Instrução 2NAENF

X0	X1	Y(Continuidade Lógica)
Desligado	**Desligado**	**Não**
Desligado	**Ligado**	**Não**
Ligado	**Desligado**	**Sim**
Ligado	**Ligado**	**Não**

A figura 61 mostra o diagrama funcional da instrução 2NAENF. Pode-se observar que, se o contato NA for ligado e o contato NF desligado, então existe continuidade lógica. Se o contato NA for desligado ou o contato NF for ligado, interrompe-se a continuidade lógica da associação em série.

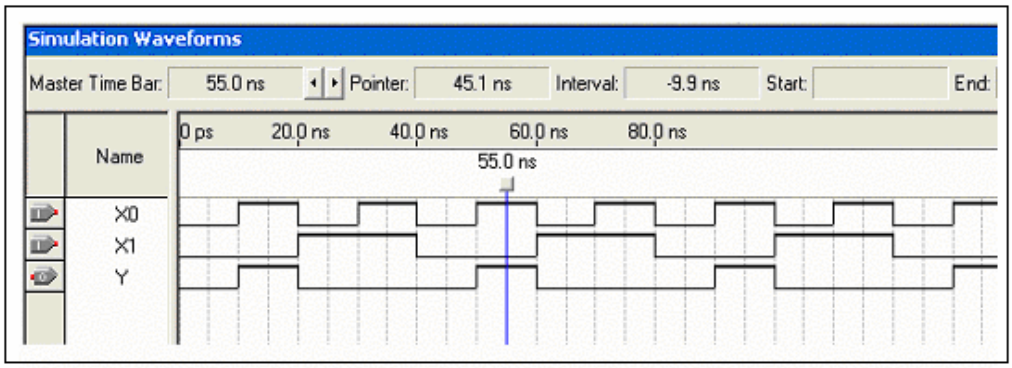

Figura 61 - Diagrama funcional da instrução 2NAENF

- **Instrução 2NAOUNF**: essa instrução executa a lógica *OR* entre um Contato Normalmente Aberto em paralelo a outro Contato Normalmente Fechado. A instrução está associada a dois

dispositivos de entrada X0 e X1, conforme apresentado na figura 62 e tabela verdade da tabela 13.

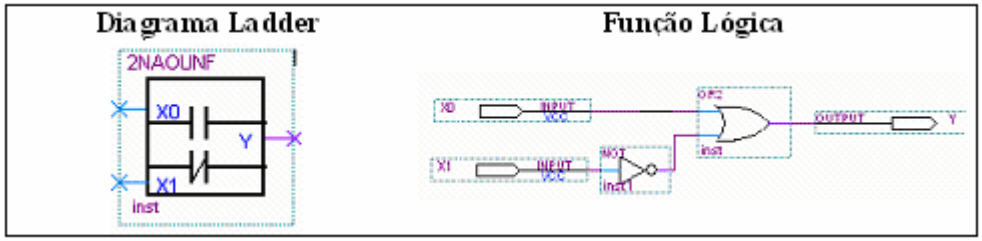

Figura 62 - Símbolo gráfico da instrução 2NAOUNF

A figura 62 apresenta o símbolo gráfico *Ladder* da instrução 2NAOUNF e a sua função lógica correspondente.

TABELA 12 - Tabela Verdade da Instrução 2NAOUNF

X0	X1	Y(Continuidade Lógica)
Desligado	**Desligado**	**Sim**
Desligado	**Ligado**	**Não**
Ligado	**Desligado**	**Sim**
Ligado	**Ligado**	**Sim**

A figura 63 mostra o diagrama funcional da instrução 2NAOUNF. Pode-se observar que, se o contato NA for ligado ou o contato NF desligado, então existe continuidade lógica. Se o contato NA for desligado e o contato NF for ligado, interrompe-se a continuidade lógica da associação em paralelo.

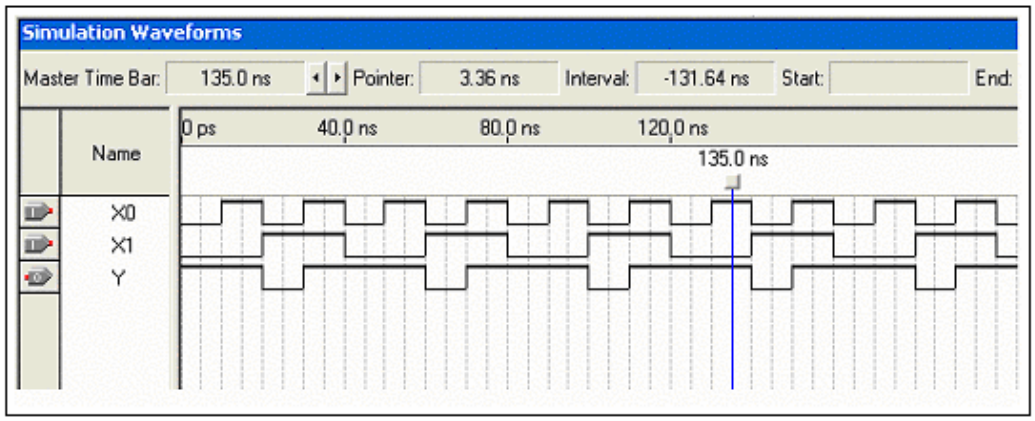

Figura 63 - Diagrama funcional da instrução 2NAOUNF

3.2.4.2 Instruções de Temporização e Contagem

As instruções de temporização e contagem são instruções de saída com funções idênticas às fornecidas pelos temporizadores e contadores construídos mecânica ou eletronicamente. São geralmente utilizadas para ativar ou desativar dispositivos elétricos ao fim de determinado tempo ou contagem. O princípio de funcionamento de uma instrução de contagem e temporização são semelhantes, pois ambas podem ser consideradas contadores. Um temporizador conta um número de intervalos de tempo fixos, necessários para atingir a duração pretendida, enquanto que um contador registra o número de ocorrências de um determinado evento (FERREIRA, 1994).

3.2.4.2.1 Mega Funções *Lpm_Counter*, *Lpm_Compare* e *Lpm_Constant*

A empresa Altera, fabricante do *software* QUARTUS II, que é utilizado neste trabalho, recomenda à utilização das mega funções *lpm_counter*,

lpm_compare e *lpm_constant* para implementação de contadores e temporizadores, em detrimento de qualquer outro tipo de contador binário, porque as mega funções são otimizadas para aumentar o desempenho da arquitetura dos circuitos integrados de tecnologia Altera Corp. (ALTERA, 2004). Por exemplo, a mega função *lpm_counter* pode implementar um contador binário básico e oferecer os seguintes recursos:

- Contador crescente (*up*), decrescente (*down*) e crescente/ decrescente (up/down).
- Suporta contagem de 2 até 256 bits.
- Capacidade de transferência de dados (*load*) síncrona e assíncrona.
- Permite portas de entrada para *preset*, *set*, *clear*, *enable*, *carry-in*, *carry-out*, dados, etc.

A figura 64 apresenta o símbolo gráfico da mega função *lpm_counter* com as portas e parâmetros opcionais (ALTERA, 2004).

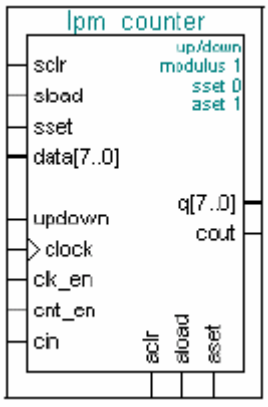

Figura 64 - Símbolo da mega função *lpm_counter*

3.2.4.2.2 Instrução Contador Crescente - CTU

A instrução intitulada CTU, incrementa uma unidade no valor acumulado a cada vez que ocorre um evento em sua entrada (CLK). Estão disponíveis na biblioteca PLCPROJECT dez (10) contadores CTU, numerados seqüencialmente de CTU60 a CTU69. Esses contadores podem ser configurados independentemente. Se um programa de aplicação necessitar de mais contadores do que os existentes, esses contadores deverão ser criados como novas macroinstruções independentes, salvas na biblioteca PLCPROJECT seguindo a numeração existente, a partir do CTU69. A figura 65 apresenta o símbolo gráfico *Ladder* da instrução CTU.

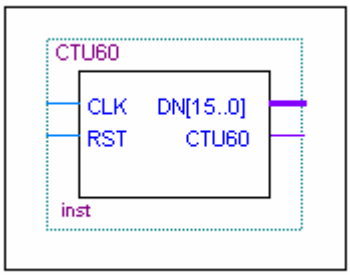

Figura 65 - Símbolo gráfico da instrução CTU

A função lógica da instrução CTU é apresentada na figura 66, sendo constituída pelas seguintes mega funções:

- Mega função *lpm_counter*: responsável pela contagem crescente (*up counter*) dos eventos de entrada e armazenamento do valor acumulado (saída DN de 16 bits). Pode ser programada para 16

bits de contagem, a qual permite contagem de 65.536 eventos (2^{16} eventos).

- Mega função *lpm_compare*: responsável pela comparação entre o valor acumulado *data(a)* (saída DN) proveniente da mega função *lpm_counter* e o valor pré-determinado *data(b)* proveniente da mega função *lpm_constant*. Quando o valor acumulado é igual ou maior que o valor pré-determinado a saída (CTU) será nível alto.
- Mega função *lpm_constant*: responsável por armazenar o valor de contagem pré-determinado. Determina a contagem a ser atingida.

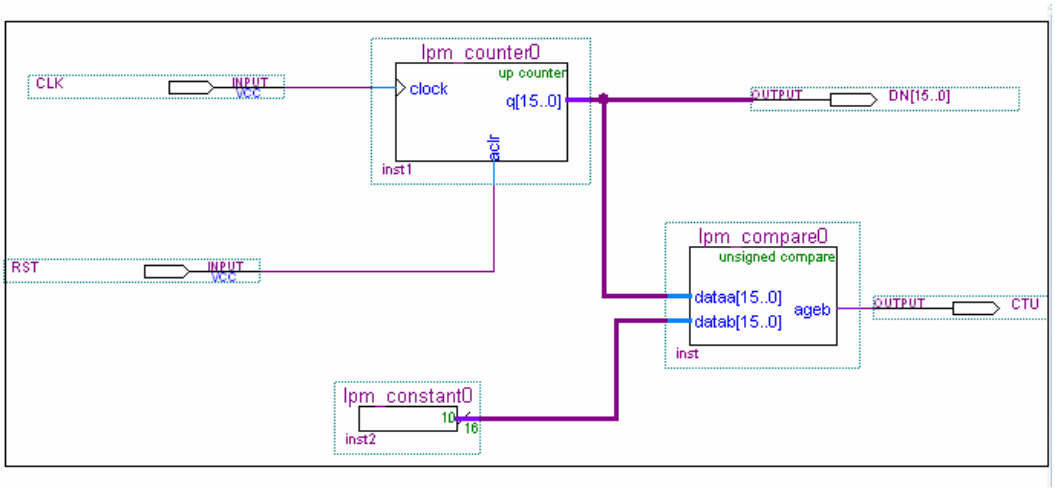

Figura 66 - Função lógica da instrução CTU

A figura 67 mostra o diagrama funcional da instrução CTU, nesse pode-se observar que a cada borda positiva do pulso de entrada (CLK), a instrução CTU

incrementa de uma unidade o seu valor acumulado (DN). A saída (CTU) será energizada, quando o valor de contagem pré-determinado for atingido. A instrução CTU continua contando, mesmo depois de ter alcançado o valor pré-determinado e a sua saída (CTU) permanecerá em nível alto. A instrução será desativada, saída (CTU) nível baixo, quando a entrada (RST) for acionada (nível alto). No exemplo da figura 67 o valor pré-determinado foi programado para cinco eventos.

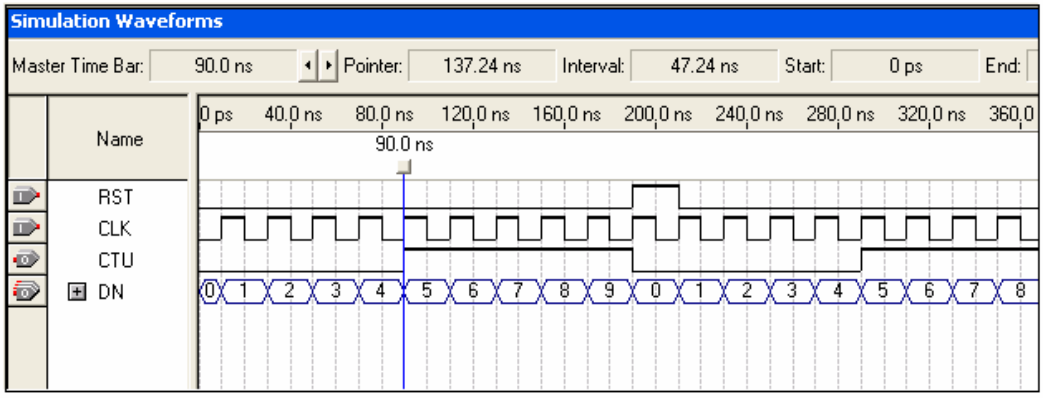

Figura 67 - Diagrama funcional da instrução CTU

3.2.4.2.3 Instrução Contador Decrescente - CTD

O princípio de funcionamento da instrução intitulada CTD é semelhante ao da instrução contador CTU, apresentada no item anterior, porém, nessa instrução o processo de contagem é decrescente desde o valor pré-determinado até o valor zero. Estão disponíveis na biblioteca PLCPROJECT dez (10) contadores CTD, numerados seqüencialmente de CTD100 a CTD109. Esses contadores podem ser configurados independentemente. Se um programa de aplicação necessitar de mais contadores do que os existentes, esses contadores deverão ser criados como

novas macroinstruções independentes, salvas na biblioteca PLCPROJECT seguindo a numeração existente, a partir do CTD109. A figura 68 apresenta o símbolo gráfico *Ladder* da instrução CTD.

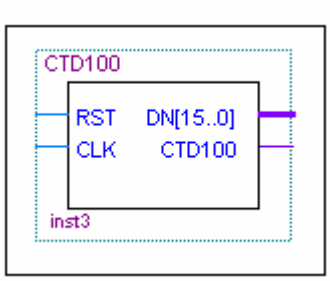

Figura 68 - Símbolo gráfico da instrução CTD

A função lógica da instrução CTD é apresentada na figura 69, sendo constituída pelas seguintes mega funções e elementos lógicos:

- Mega função *lpm_counter*: responsável pela contagem decrescente (*down counter*) dos eventos de entrada e armazenamento do valor pré-determinado de contagem (*data*), proveniente da mega função *lpm_constant*. Essa função pode ser programada para 16 bits de contagem, o que permite contagem de 65.536 eventos (2^{16} eventos).
- Mega função *lpm_constant*: responsável por armazenar o valor de contagem pré-determinado, a qual será transferida para a mega função *lpm_counter*. Essa função determina o valor a partir do qual será iniciada a contagem.

- Mega função *lpm_compare*: responsável pela comparação entre o valor pré-determinado proveniente da mega função *lpm_counter* e o valor zero. A sua saída (aeb) será ativada, nível alto, quando o valor atual (DN) da mega função *lpm_counter* atingir o valor zero.
- Elemento lógico *Flip-flop RS*[42]: responsável pela memorização do estado da saída da mega função *lpm_compare* (aeb). A sua saída (CTD) será ativada quando a contagem atingir o valor zero e será desativada (*Reset*), somente quando o sinal (RST) for ativado, início da nova contagem.

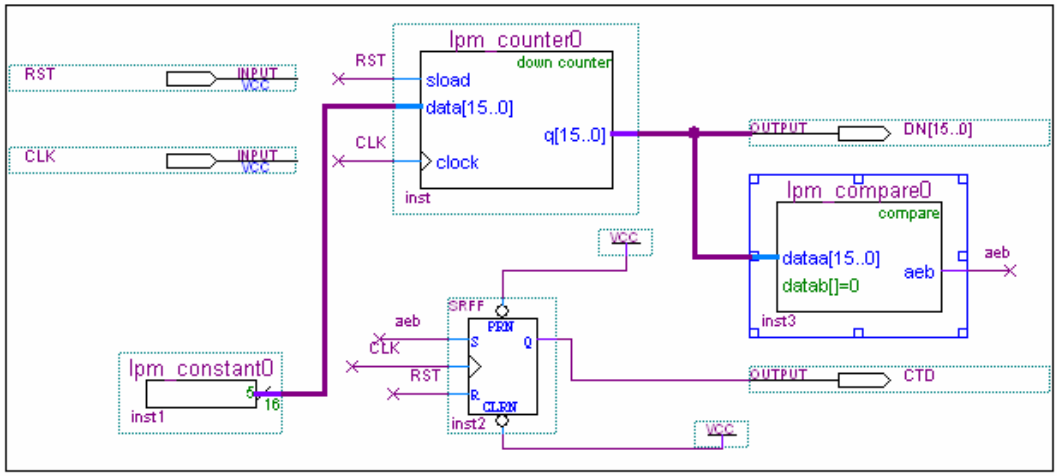

Figura 69 - Função lógica da instrução CTD

A figura 70 mostra o diagrama funcional da instrução CTD, nesse pode-se observar que inicialmente o sinal (RST), quando nível alto, carrega o valor de contagem pré-determinado (cinco) no contador. A cada borda positiva do pulso de

[42] Circuito ou dispositivo que contém elementos ativos com dois possíveis estados estáveis, podendo estar em um único estado em cada instante. Sinônimo de circuito biestável.

entrada (CLK), a instrução CTD diminui de uma unidade o valor pré-determinado**.**

Quando o valor pré-determinado atingir o valor zero, a saída (CTD) será energizada, nível alto. A instrução será desativada, saída (CTD) nível baixo, quando a entrada (RST) for acionada e o valor pré-determinado (cinco) for novamente carregado no contador.

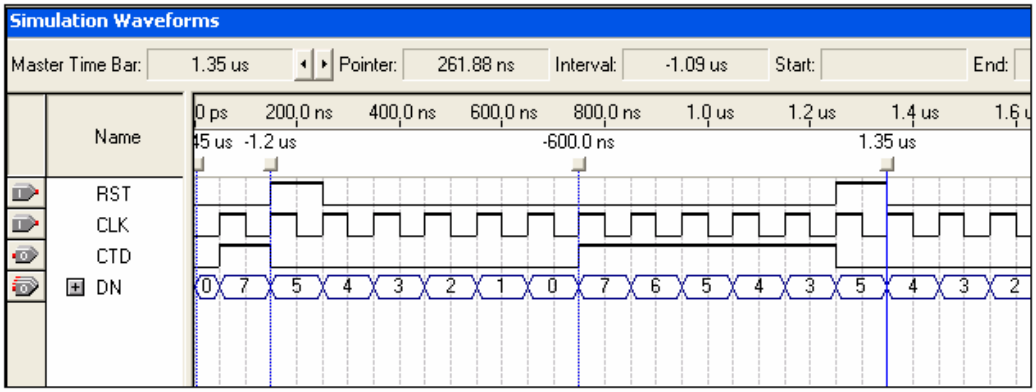

Figura 70 - Diagrama funcional da instrução CTD

No decorrer dos testes funcionais das instruções CTU e CTD realizado no CLP protótipo, observou-se à necessidade de implementação de circuitos eliminadores de ruído na entrada de pulso das instruções de contagem. Tal fato decorreu em virtude das chaves utilizadas para simularem os pulsos de entrada serem mecânicas e gerarem o mencionado ruído. Os circuitos eliminadores de ruído foram implementados por meio de uma macro instrução chamada *antibouncing*, cujo símbolo gráfico *Ladder* é apresentado na figura 71.

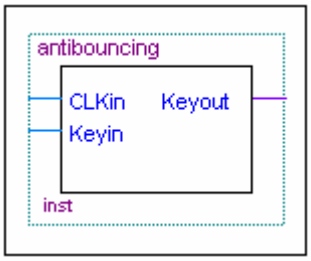

Figura 71 - Símbolo gráfico da instrução *antiboucing*

A figura 72 mostra a função lógica da instrução intitulada *antiboucing*, que é constituída por dois blocos lógicos. O primeiro bloco lógico (*div1M*) tem a função de dividir a freqüência de *clock* geral do sistema de 10 MHz, utilizada no desenvolvimento deste trabalho, por 10^6, para obtenção de uma freqüência de 10Hz, que será utilizada na sincronização do segundo bloco lógico intitulado *disbounce*. Nesse bloco a entrada (Keyin) é ligada a um dispositivo externo, por exemplo, uma chave de pulso mecânica (*pushbottom*) e a saída (Keyout) é ligada a entrada de *clock* da instrução CTU ou CTD.

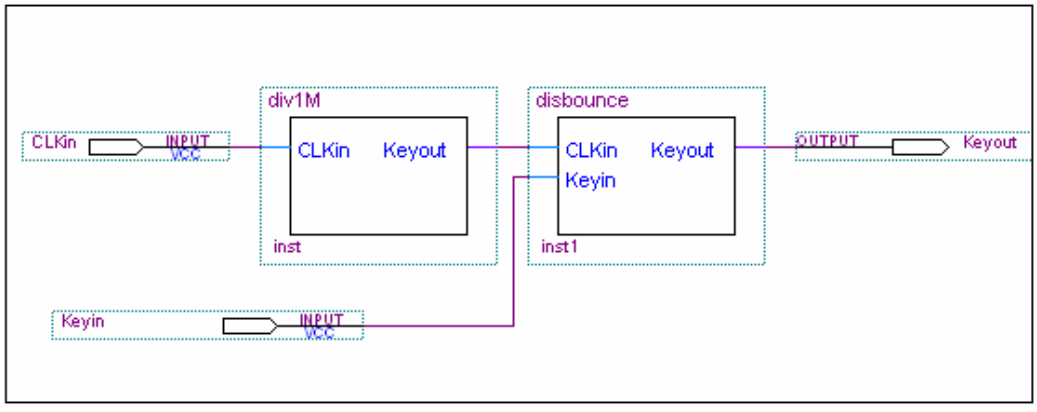

Figura 72 - Função lógica da instrução *antiboucing*

A figura 73 apresenta o diagrama funcional do segundo bloco da instrução *antibouncing*. Pode-se observar que o sinal que representa o ruído gerado na entrada (Keyin), não é reproduzido na saída (Keyout).

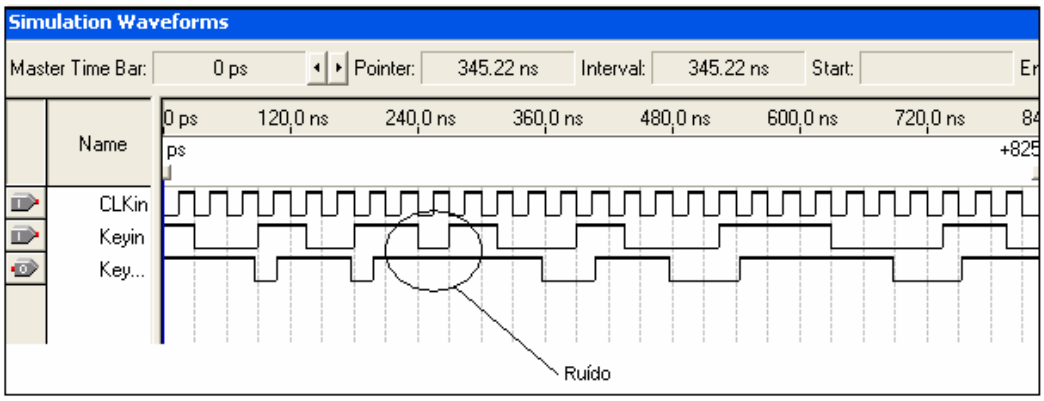

Figura 73 - Diagrama Funcional da instrução *antibouncing*

3.2.4.2.4 Instrução Temporizador - TON

As instruções TON são utilizadas para temporização de condições ou eventos controlados pelo programa de aplicação (GEORGINI, 2003). A biblioteca PLCPROJECT desenvolvida para este trabalho, dispõe basicamente de dois tipos de temporizadores: o TOND e o TONC. A diferença entre os temporizadores refere-se apenas à base de tempo. No temporizador TOND a base de tempo é de 0,1 segundo (um décimo de segundo), ou seja, incrementa a cada décimo de segundo (10Hz). No temporizador TONC a base de tempo é de 0,01 segundo (um centésimo de segundo), ou seja, incrementa a cada centésimo de segundo (100Hz). Os temporizadores apresentam as seguintes especificações:

- **Temporizador TOND**: Possui duas entradas. Uma entrada (CLKin) responsável pela geração da base de tempo de 0,1 segundo (10Hz) e outra entrada (ENABLE) responsável pela habilitação da instrução. Estão disponíveis na biblioteca PLCPROJECT dez (10) temporizadores TOND, numerados seqüencialmente de TOND50 a TOND59. Esses temporizadores podem ser configurados independentemente. Se um programa de aplicação necessitar de mais temporizadores do que os existentes, esses temporizadores deverão ser criados como novas macroinstruções independentes, salvas na biblioteca PLCPROJECT seguindo a numeração existente, a partir do TOND59. A figura 74 apresenta o símbolo gráfico *Ladder* criado da instrução TOND.

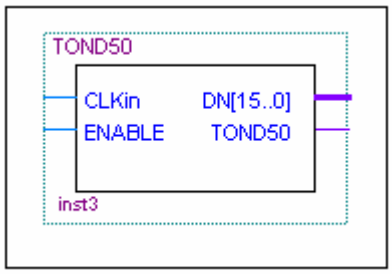

Figura 74 - Símbolo gráfico da instrução TOND

A figura 75 apresenta a função lógica da instrução TOND. Essa é basicamente similar a função lógica da instrução contador CTU, apresentada no item 3.2.4.2 deste trabalho, com acréscimo do bloco (*CLKgen*) gerador da base de

tempo de 0,1 segundo (10Hz). O tempo pré-determinado é armazenado na mega função *lpm_constant*. Quando a entrada (ENABLE) torna-se verdadeira, o valor acumulado (DN) na mega função *lpm_counter* começa a incrementar, a cada 0,1 segundo (10Hz), até alcançar o valor do tempo pré-determinado, após o que a saída (TON) torna-se ativa (nível alto). Se a entrada (ENABLE) permanecer verdadeira, o valor acumulado (DN) continuará incrementando independente do valor pré-determinado.

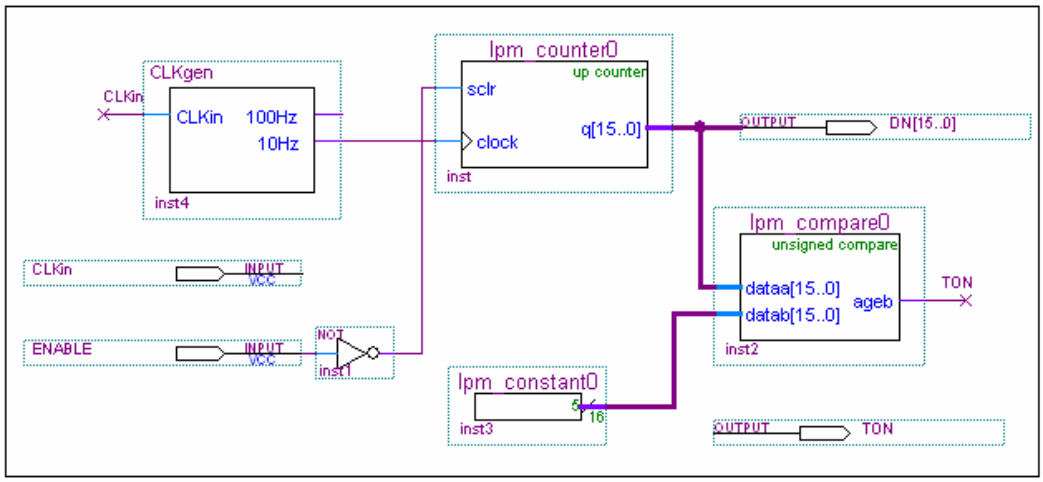

Figura 75 - Função lógica da instrução TOND

A figura 76 apresenta o diagrama funcional da instrução TOND. Uma vez acionada a entrada (ENABLE), a instrução TOND terá o seu valor atual incrementado a cada pulso de entrada (0,1 segundo), até que a mesma atinja o valor de tempo pré-determinado (cinco pulsos), quando então sua saída (TON) será ativada, nível alto. Porém, se a entrada (ENABLE) permanecer ativada, a saída (TON) também permanecerá ativada até a entrada (ENABLE) ser falsa.

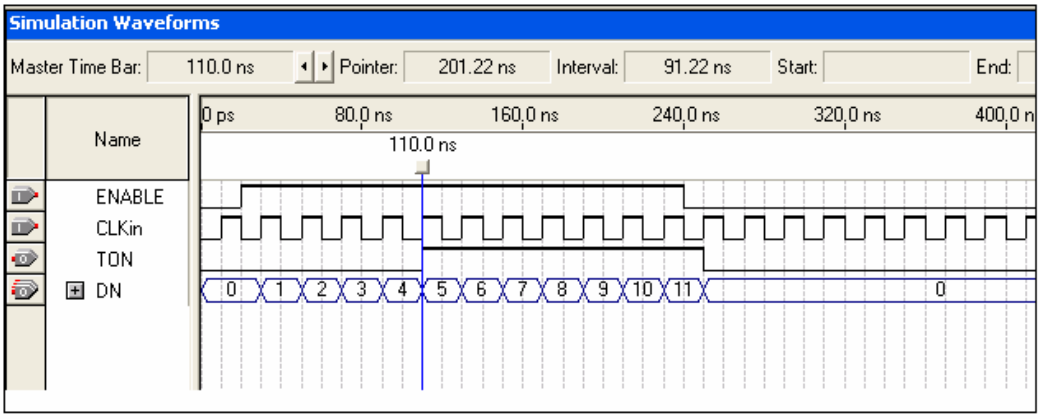

Figura 76 - Diagrama funcional da instrução TOND

- **Temporizador TONC**: O princípio de funcionamento da instrução TONC é similar ao da instrução TOND, apresentada no item anterior. Possui duas entradas, uma entrada (CLKin) responsável pela geração da base de tempo de 0,01 segundo (100Hz) e outra entrada (ENABLE) responsável pela habilitação da instrução. Estão disponíveis na biblioteca PLCPROJECT dez (10) temporizadores TONC, numerados seqüencialmente de TONC100 a TONC109. Esses temporizadores podem ser configurados independentemente. Se um programa de aplicação necessitar de mais temporizadores do que os existentes, esses temporizadores deverão ser criados como novas macroinstruções independentes, salvas na biblioteca PLCPROJECT seguindo a numeração existente, a partir do TONC109. A figura 77 apresenta o símbolo gráfico *Ladder* desenvolvido para a instrução TONC.

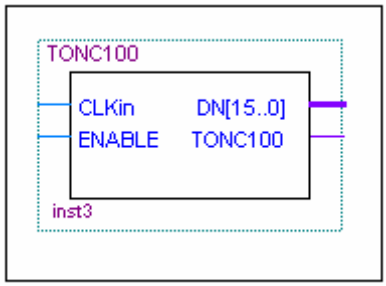

Figura 77 - Símbolo gráfico da instrução TONC

A figura 78 apresenta a função lógica da instrução TONC. Essa é semelhante à função lógica da instrução TOND, apresentada no item anterior, com a diferença principal contida no bloco (*CLKgen*), o qual gera uma base de tempo de 0,01 segundos (100Hz).

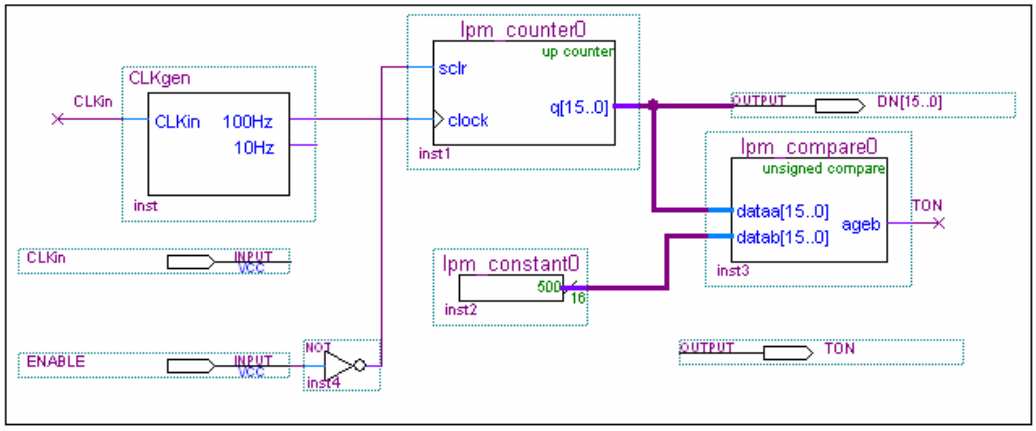

Figura 78 - Função lógica da instrução TONC

CAPÍTULO 4 - ENSAIOS PRÁTICOS

Este capítulo apresenta os ensaios realizados com o CLP proposto neste trabalho, baseados em Lógica Programável Estruturada. Os ensaios práticos foram realizados em duas fases. Na primeira fase foram utilizados o *hardware* do kit de desenvolvimento FPT1 CPLD /FPGA, descrito no item 3.1.2.1.3, deste trabalho, em conjunto com o *software* QUARTUS II e a biblioteca PLCPROJECT elaborada (macroinstruções do tipo relés). Na segunda fase dos ensaios práticos foram utilizados o *hardware* do protótipo de CLP, o *software* QUARTUS II e a biblioteca PLCPROJECT (macroinstruções do tipo relés, temporização e contagem), para simulação e validação da metodologia adotada neste trabalho.

4.1 Exemplo de Aplicação 1

Nesta seção, pretende-se mostrar a viabilidade da metodologia adotada, testando o funcionamento do CLP proposto e a biblioteca PLCPROJECT (macroinstruções do tipo relés).

Para esse, fim utilizou-se como exemplo um sistema de controle para transportador automático de peças. O objetivo é analisar o comportamento de toda a nova arquitetura e a biblioteca de macroinstruções gráficas PLCPROJECT, para validar a arquitetura proposta.

4.1.1 Sistema de Controle para Transportador Automático de Peças

O transporte de peças em uma esteira e a sua retirada deve ser realizado automaticamente, sem a intervenção do operador, a partir do acionamento de uma botoeira (BL1). A figura 79 ilustra o sistema de transporte de peças automático.

Acoplado mecanicamente à esteira, um motor elétrico (MT1) realiza a sua movimentação. Um cilindro pneumático de duas vias comandado por dois solenóides, S1 (avança pistão) e S2 (recua pistão), são os responsáveis pela retirada das peças da esteira. O sistema automatizado deve proporcionar o seguinte comportamento:

- No acionamento da botoeira (BL1), automaticamente inicia-se a movimentação da esteira com a ligação do motor do transportador (MT1), desde que a botoeira desliga motor (BL0), botoeira emergência desliga sistema (BE1) e o relé térmico de proteção do motor (RT1) não estejam acionadas.
- A peça será transportada pela esteira até ser detectada pelo sensor de fim de curso (FC1). Estando o sensor de fim de curso (FC1) acionado, o motor do transportador (MT1) será desligado, parando a esteira e o solenóide de avanço do pistão (S1) será ligado, empurrando a peça para fora da esteira.
- Quando a peça for retirada da esteira, um segundo sensor (FC2) deverá detectá-la. Estando o sensor FC2 acionado, o pistão deve ser recuado por meio do desligamento do solenóide de avanço (S1) e acionamento do solenóide de recuo (S2).

- Ao desativar o sensor (FC2), o motor do transportador (MT1) deve ser ligado, iniciando o transporte de uma nova peça;
- Essa seqüência deve se repetir indefinidamente.
- Em qualquer momento em que for pressionada a botoeira desliga motor (BL0), o motor deve ser desligado.
- Em qualquer momento em que for pressionada a botoeira desliga sistema (BE1) ou acionado o relé térmico (RT1) do motor, todas as saídas devem ser desligadas.

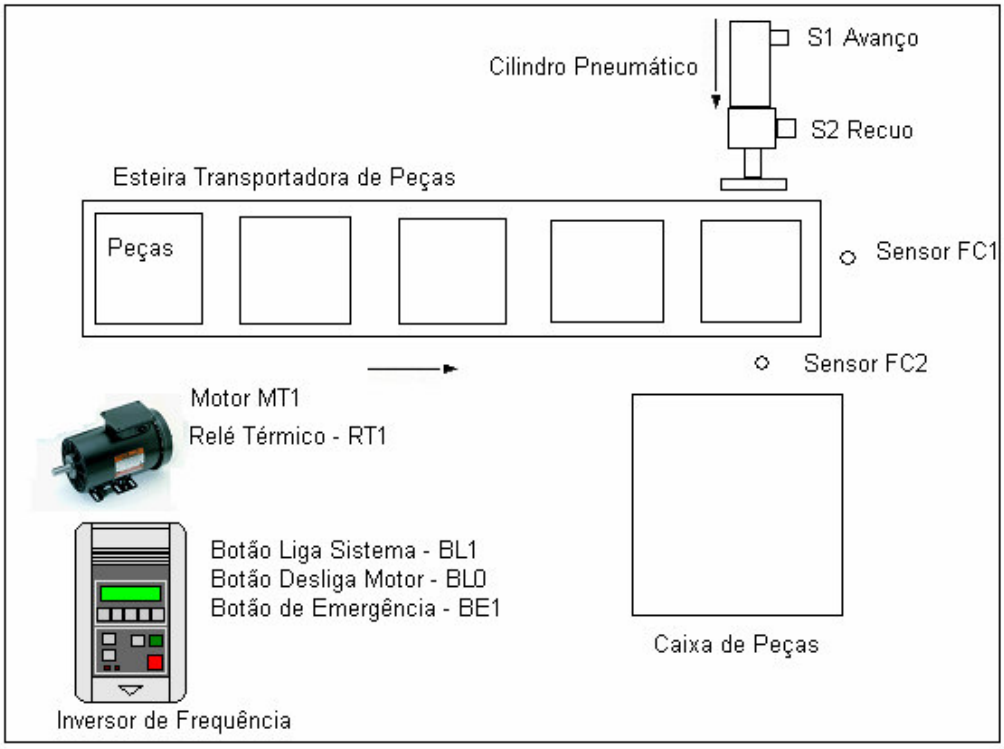

Figura 79 - Diagrama do transportador automático de peças

As tabelas 14 e 15 apresentam as configurações de entradas e saídas do CLP em estudo, para implementação do sistema de controle para transportador automático de peças.

TABELA 13 - Configuração de entradas do CLP

Entradas Discretas	Descrição
BL1 (Pino 46)	Botoeira de Liga Sistema
BL0 (Pino 47)	Botoeira de Desliga Motor
BE1 (Pino 48)	Botoeira de Emergência do Sistema
RT1 (Pino 49)	Relé Térmico de Proteção do Motor
FC1 (Pino 51)	Sensor Fim de Curso 1 (Peça Pronta para ser Empurrada da Esteira)
FC2 (Pino 59)	Sensor Fim de Curso 2 (Peça Fora da Esteira)

TABELA 14 - Configuração de saídas do CLP (I)

Saídas Discretas	Descrição
MT1 (Pino 7)	Motor do Transportador da Esteira
S1 (Pino 8)	Solenóide Avança Pistão
S2 (Pino 9)	Solenóide Recua Pistão
L1 (Pino 11)	Sinalização Motor Desligado
L2 (Pino 12)	Sinalização Sensor FC1 Acionado
L3 (Pino 13)	Sinalização Sensor FC2 Acionado
L4 (Pino 14)	Sinalização Relé Térmico RT1 Acionado
L5 (Pino 10)	Sinalização Botoeira de Emergência Acionada

A figura 80 mostra o diagrama de ligação implementado no protótipo do CLP proposto, o qual foi utilizado no ensaio prático do sistema de controle do transportador automático de peças. Esse tem a função de comandar o motor (MT1), os solenóides das válvulas de avanço (S1) e recuo (S2) do cilindro, bem como monitorar os sensores (FC1) e (FC2), e sinalizar a operação através das lâmpadas de sinalização (L1, L2, L3, L4, L5 e L6). A descrição do fluxo de funcionamento será apresentada no item 4.1.1.1 deste trabalho.

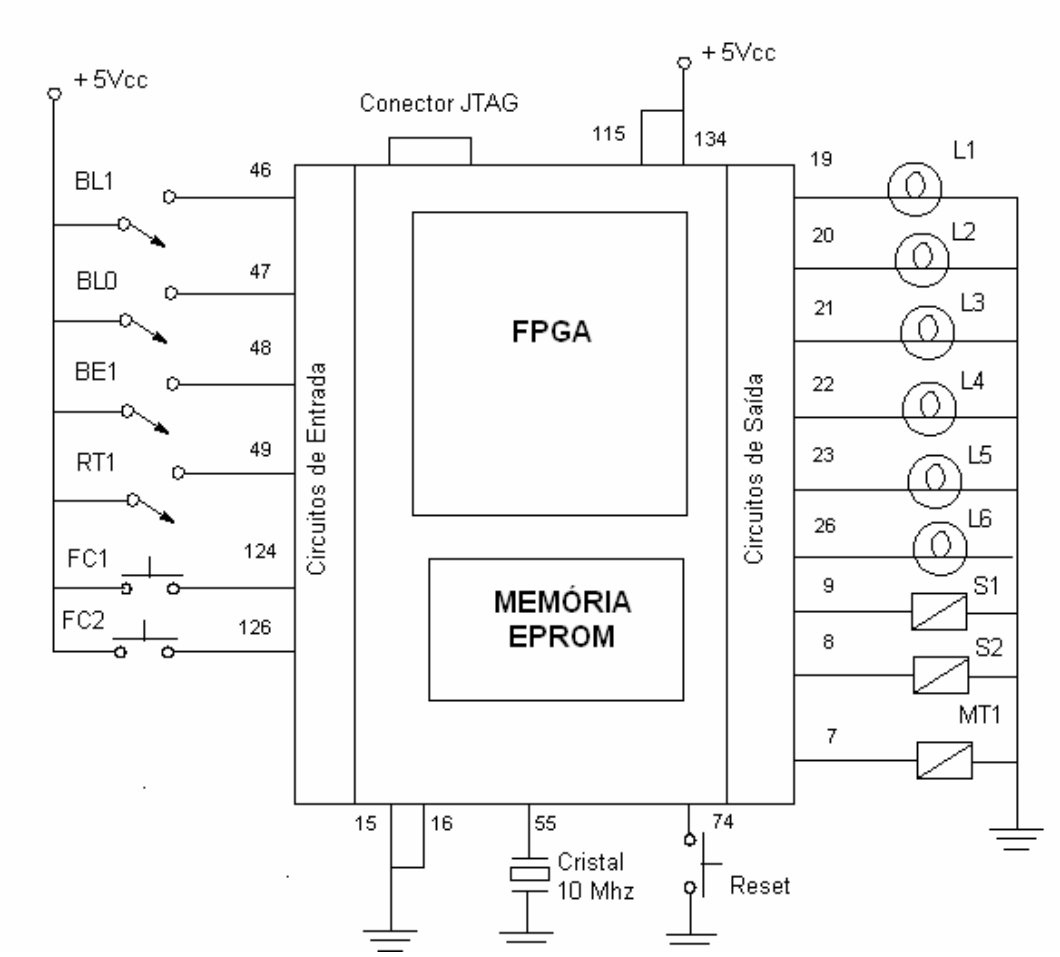

Figura 80 - Diagrama de ligações do CLP protótipo

4.1.1.1 Fluxograma Analítico do Sistema

A seguir, é apresentada uma possibilidade de descrição para o sistema de controle para transportador automático de peças, na forma de fluxograma analítico do sistema. O fluxograma da figura 81 mostra o acionamento da botoeira BL1, atuação do relé térmico RT1, botoeira de emergência BE1 e o acionamento do motor do transportador MT1.

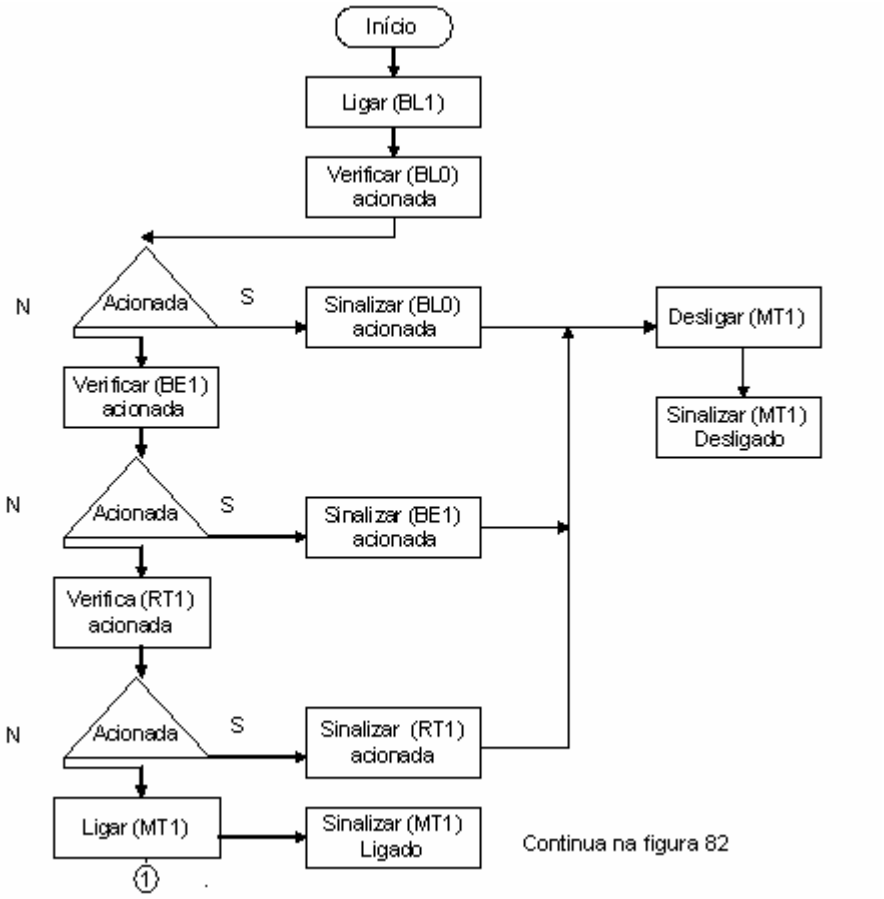

Figura 81 - Acionamento do transportador MT1

O fluxograma da figura 82 (a) mostra as ações para a deteção da peça no final da esteira pelo sensor (FC1), a parada do transportador (MT1) e a expulsão da peça realizada pelo avanço do pistão. O fluxograma da figura 82 (b) mostra as ações da peça fora da esteira, a sua detecção pelo sensor FC2, o acionamento do motor do transportador MT1 e o retorno do pistão a sua posição inicial.

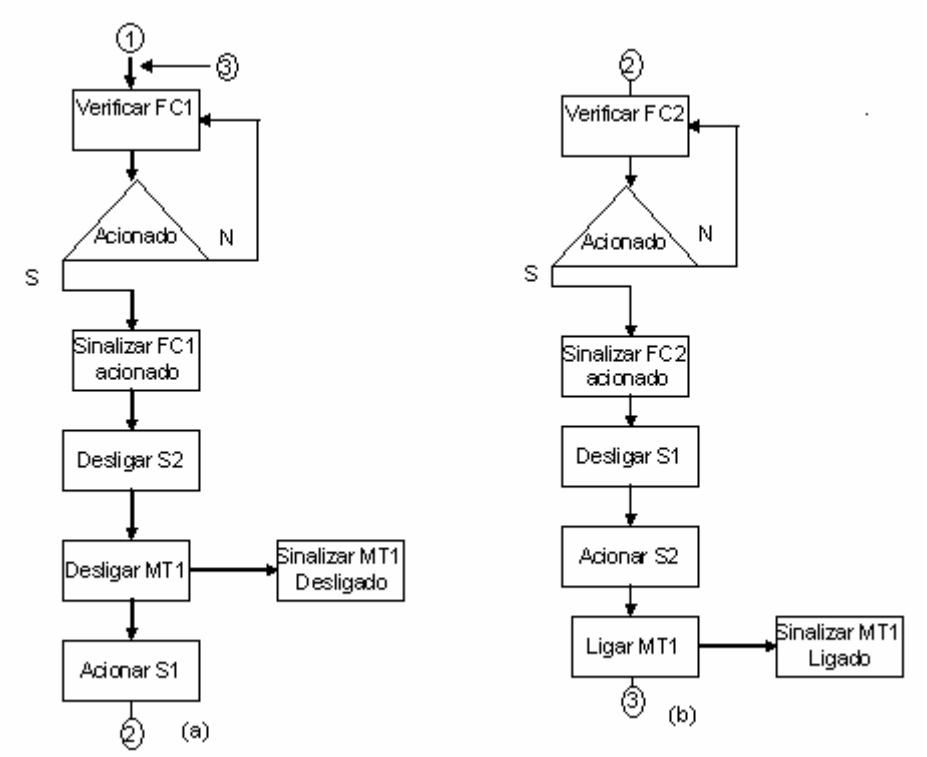

Figura 82 - Atuação dos sensores FC1 e FC2

O programa utilizado no ensaio do protótipo do CLP foi elaborado para atender as tarefas do sistema de controle para o transportador automático de peças, as suas instruções estão no apêndice 1. O programa foi desenvolvido no *software* QUARTUS II, modo Editor Gráfico, a partir da biblioteca de símbolos gráficos PLCPROJECT.

4.2 Exemplo de Aplicação 2

Nesta seção mostra-se a viabilidade da metodologia adotada, apresentando a seqüência de testes de funcionamento do *hardware* do CLP protótipo e a biblioteca PLCPROJECT (macroinstruções do tipo relés, macroinstruções de temporização e contagem).

Para esse fim utilizou-se como exemplo um sistema de controle para semáforo. O principal objetivo dessa fase é analisar o comportamento do *hardware* adotado para a nova arquitetura e a biblioteca de macroinstruções gráficas PLCPROJECT, instruções de temporização e contagem, a fim de validar a arquitetura proposta.

4.2.1 Sistema de Controle para Semáforo

Um sistema de controle para semáforo destina-se ao controle automático do cruzamento de duas vias: uma avenida principal e uma rua secundária, conforme indicado na figura 83. O controle será realizado por meio da temporização de dois semáforos de três fases para veículos.

O sistema de controle deve proporcionar o seguinte comportamento:

- O ciclo de funcionamento tem início automático com a passagem livre de veículos pela avenida principal e pedestres pela rua secundária, após trinta segundos o ciclo se inverterá, passando

antes pela sinalização de atenção (sinalização amarela) por quatro segundos.

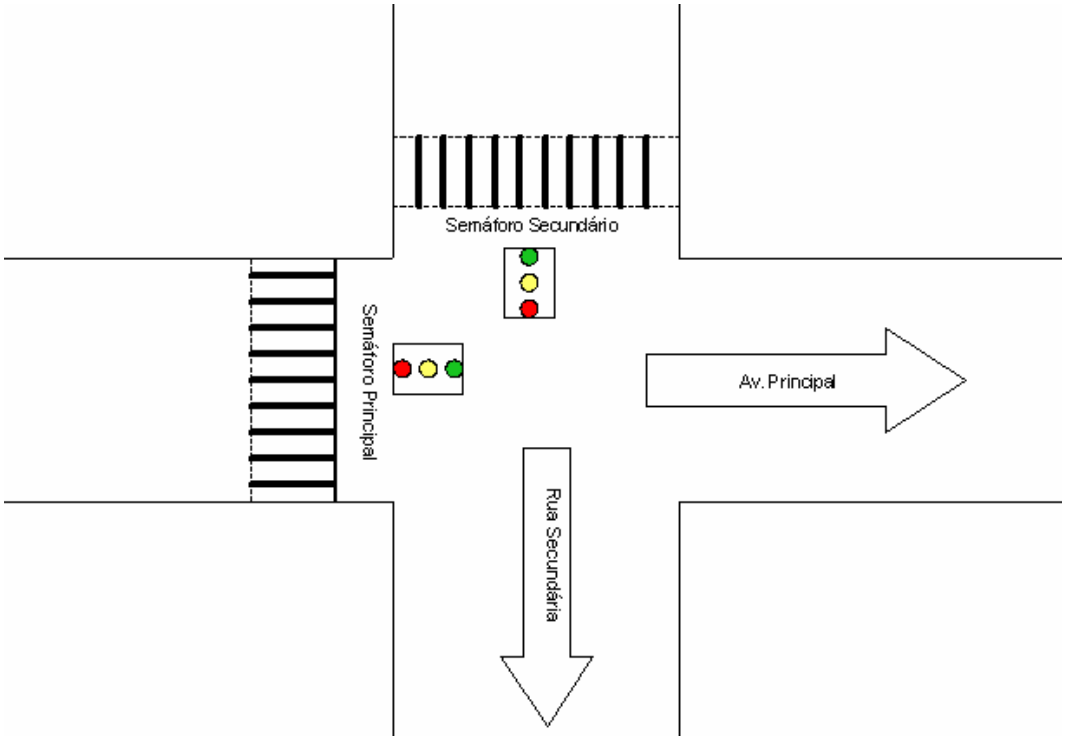

Figura 83 - Sistema de controle de semáforos

- O novo ciclo de funcionamento tem início automático com a passagem livre de veículos pela rua secundária e pedestres pela avenida principal, após 30 segundos o ciclo se inverterá, passando antes pela sinalização de atenção (sinalização amarela) por quatro segundos.

A figura 84 apresenta o diagrama de ligação implementado no protótipo do CLP proposto, o qual foi utilizado na segunda fase dos ensaios práticos do sistema de controle para semáforos. Pode-se observar que o controle do semáforo não utilizou nenhuma das oito entradas do protótipo do CLP, pois a lógica de acionamento dos semáforos foi controlada apenas pelos tempos determinados para cada fase, sem a intervenção de entradas externas.

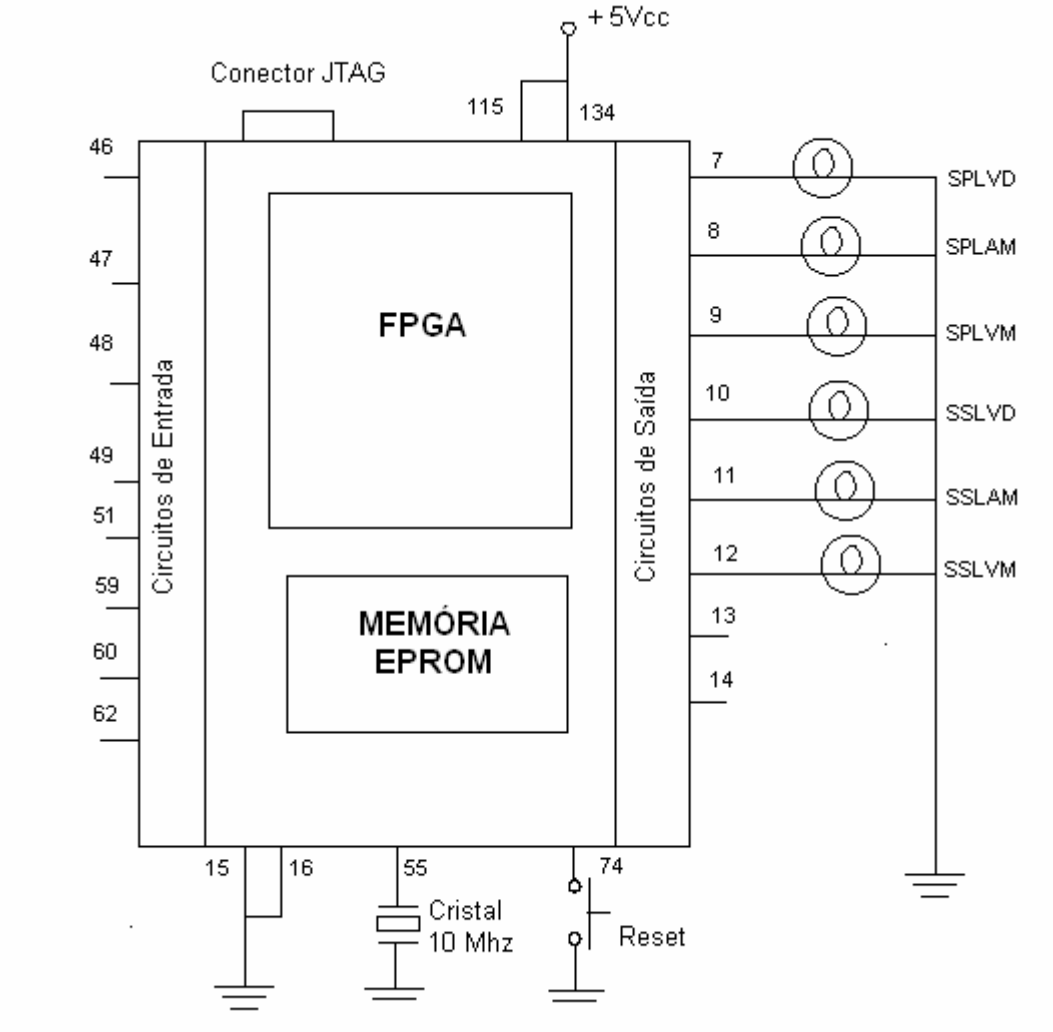

Figura 84 - Diagrama de ligações para sistema de semáforos

A tabela 16 apresenta as configurações de saídas do protótipo do CLP em estudo, para implementação do sistema de controle para semáforo.

TABELA 15 - Configuração de saídas do CLP (II)

Saídas Discretas	Descrição
SPLVD (Pino 7)	Semáforo Principal – Lâmpada Verde
SPLAM (Pino 8)	Semáforo Principal – Lâmpada Amarela
SPLVM (Pino 9)	Semáforo Principal – Lâmpada Vermelha
SSLVD (Pino 10)	Semáforo Secundário – Lâmpada Verde
SSLAM (Pino 11)	Semáforo Secundário – Lâmpada Amarela
SSLVM (Pino 12)	Semáforo Secundário – Lâmpada Vermelha

4.2.1.1 Fluxograma Analítico do Sistema

A seguir é apresentada uma possibilidade de descrição para o sistema de controle de tráfego, na forma de fluxograma analítico do sistema. O fluxograma da figura 85 mostra as ações iniciais do primeiro ciclo de funcionamento do semáforo de carros da avenida principal e da rua secundária.

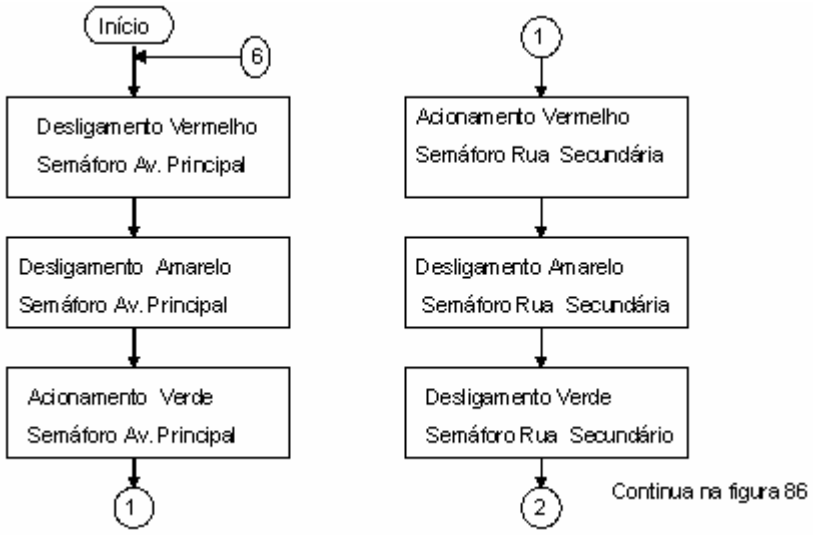

Figura 85 - Ações do primeiro ciclo do semáforo da Av. principal

O fluxograma da figura 86 apresenta as ações ocorridas durante o segundo ciclo de funcionamento dos semáforos de carros da avenida principal.

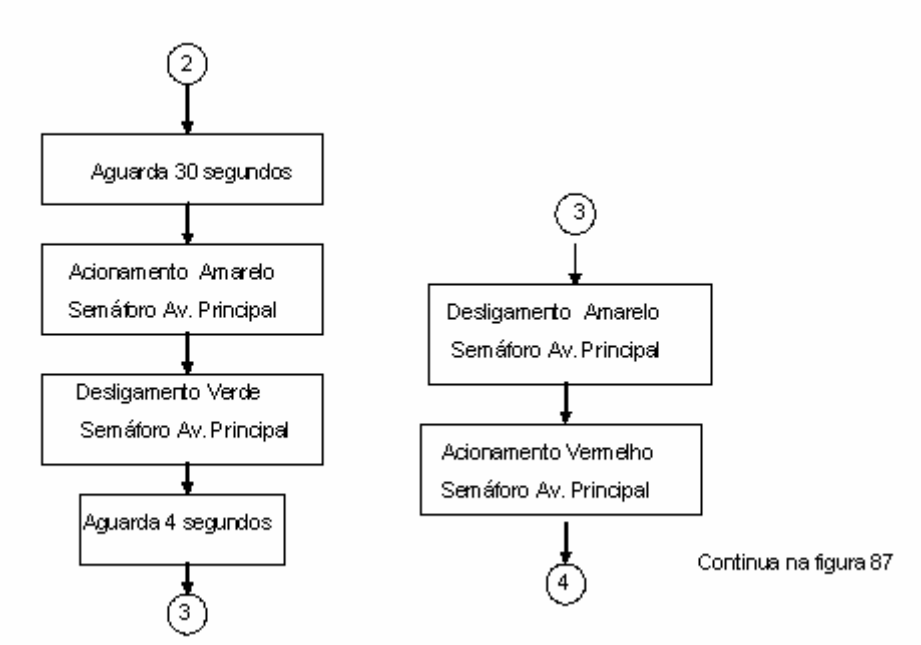

Figura 86 - Ações do segundo ciclo do semáforo da Av. principal

O fluxograma da figura 87 apresenta as ações ocorridas durante o segundo ciclo de funcionamento dos semáforos de carros da rua secundária.

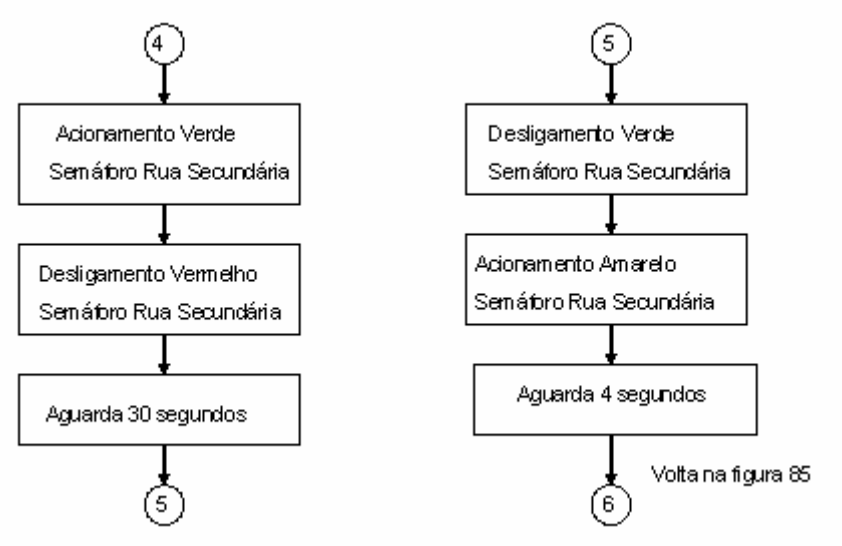

Figura 87 - Ações do segundo ciclo do semáforo da rua secundária

O programa utilizado no ensaio do protótipo do CLP proposto, elaborado para atender as tarefas do sistema de controle para semáforos, está no apêndice 1 deste trabalho. O programa foi desenvolvido no *software* QUARTUS II, modo editor gráfico, a partir da biblioteca de símbolos gráficos PLCPROJECT (instruções tipo relé, temporização e contagem).

4.3 Resultados Obtidos

Foram realizadas diversas simulações funcionais utilizando o *kit* de desenvolvimento FPT1 e o protótipo de CLP, em conjunto com a biblioteca PLCPROJECT elaborada. Os blocos de macroinstruções do tipo relés, temporização e contagem implementados estão funcionando perfeitamente (parte operativa, parte controle, etc.). Como exemplo é mostrado na tabela 17 um comparativo entre os tempos de processamento do protótipo de CLP elaborado e um CLP tradicional FP0, fabricado pela empresa Matsushita, executando os programas aplicativos para controle de transportador automático de peças, descrito no item 4.1 deste trabalho, e controle de semáforos, apresentado no item 4.2. Pode-se observar na tabela que o tempo de execução total dos programas é da ordem de 13,4 ηs (nanossegundos) no protótipo de CLP proposto, enquanto que para o CLP tradicional FP0 obteve-se o tempo médio de 30μs (microssegundos) para executar os mesmos programas de aplicação.

A listagem dos programas em Linguagem Ladder para o CLP FP0 encontra-se no apêndice 1 deste trabalho, juntamente com a listagem dos programas desenvolvidos no *software* QUARTUS II, a partir da biblioteca de símbolos gráficos PLCPROJECT.

TABELA 16 - Tempos de processamento entre CLPs (II).

CLP Tradicional	**CLP Proposto**
Tempo de Scan = 30 μs	Tempo de execução = 12,9 ηs
Menor tempo de Scan = 20 μs	Menor tempo de execução = 12,9 ηs
Maior tempo de Scan = 30 μs	Maior tempo de execução = 13,4 ηs
Tempo médio de execução por instrução = 1,5 μs	

CAPÍTULO 5 - CONCLUSÕES

Os resultados dos ensaios práticos efetuados com o protótipo desenvolvido mostram que os objetivos propostos foram alcançados em toda a sua plenitude, principalmente no que concerne à implementação de um sistema que equivale a um Controlador Lógico Programável, o qual foi possível integrar em um único dispositivo de Lógica Programável Estruturada, as funções do microcontrolador e dos seus circuitos integrados de aplicações específicas, **reduzindo as dimensões do sistema**.

No ensaio do protótipo, utilizando Lógica Programável Estruturada, nesse caso específico um dispositivo FPGA, pode-se observar, por meio da simulação funcional e temporal do programa, a minimização dos tempos de processamento. Na aplicação apresentada neste trabalho obteve-se um t**empo de execução total dos programas estudados no protótipo de 13,4 ηs (nanossegundos),** enquanto que para um CLP tradicional obteve-se um tempo de **30 μs (microssegundos) para executar os mesmos programas.**

No protótipo, a substituição da CPU (microcontrolador, sistema de memória e circuitos de controle) pelo bloco FPGA possibilitou mudanças na execução do programa de aplicação que permitiram a eliminação dos ciclos de busca de instruções em memória e a eliminação dos ciclos de varredura (*scan time)* do CLP tradicional, aumentando a velocidade de processamento do sistema.

A utilização de Lógica Programável Estruturada na arquitetura do protótipo possibilitou a configuração e reconfiguração do seu *hardware* pelo usuário final. Outra vantagem obtida foi a redução do número de componentes ASICs, diminuindo o consumo de energia elétrica do sistema.

Um dos limites observados no desenvolvimento do trabalho está relacionado com a dependência de *hardware* adequado (FPGAs de nova geração de maior capacidade de elementos lógicos), para implementação de *softwares* complexos (macroinstruções gráficas de manipulação de dados), característica que também pode ser observada nos CLPs tradicionais.

Como sugestões para trabalhos futuros, já que o enfoque principal deste trabalho é o desenvolvimento de Controladores baseado em Lógica Programável Estruturada, são: i) o desenvolvimento de um compilador da Linguagem *Ladder* para VHDL, que permita programar o CLP proposto segundo a norma IEC 61131-3; ii) o desenvolvimento de uma interface de comunicação padrão Ethernet que permita ao CLP proposto comunicar-se em rede local; iii) o desenvolvimento de macroinstruções e circuitos de entrada e saída para conversão analógica/ digital e vice-versa.

REFERÊNCIAS BIBLIOGRÁFICAS

AFONSO, R., CAMPOS, C. (2002). **Síntese de Alto Nível para Componentes Programáveis.** Trabalho sobre CPLD/FPGA - Instituto Superior de Engenharia do Porto, Porto, Portugal. Disponível em <http:// www.ipp.pt/conteudos/89/isep_eeec.pdf >. Acesso em: 12/05/2004).

ALTERA Corp. (1998). Configuration EPROMs for FLEX Devices. Data Sheet. Ver .9.

ALTERA Corp. (1998). **FLEX 10K Embedded Programmable Logic Family**. Data Sheet. Ver.3.1.1.

ALTERA Corp. (1995). CPLDs vs FPGAs – Comparing High Capacity Programmable Logic. Product Information Bulletin 18, ver.1.

ALTERA Corp. (1995). **Implementing RAM Functions in FLEX 10K Devices.** Applications Note 52,ver1.

ALTERA Corp. (2004). Lpm_Counter Megafunction – User Guide. Ver1. Disponível por www em http// www.altera.com. Acesso em 03/01/2005.

ALLEN BRADLEY Corp. (2000). Digital Servo Drive Power and Flexibility. Data sheet. Disponível em <http://www.ab.com/motion/controllers/2098-BR001A-EN-P-AUG00.pdf.>. Acesso em 03/01/2005.

ALLEN BRADLEY Corp. (1996). Micro Mentor – Entendendo e Utilizando os Microcontroladores. Publicação Técnica.São Paulo, Brasil.

ARAGÃO, A.C.O.S. (1998). Uma Arquitetura Sistólica para solução de Sistemas Lineares Implementada com Circuitos FPGAs. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

ASHENDEN, J. P. (1990). The VHDL Cookbook. First Edition**.** Depto Computer Science, University of Adelaide South, Austrália.

AROMAT Corporation. (2000). Programming Tool *Software* – Control FPWIN GR: Operational Guide BooK. Osaka, Japan.

AROMAT Corporation. (2002). Small PLC - FP0, FP1, FP-M (C16T), FP-M (C20R/C20T/C32T). Data sheet. Disponível em <http://www.metaltex.com.br >. Acesso em 03/01/2005.

BABB, M. (2004). FPGA – based control: a smashing new technology trend. Control Engineering Europe.October edition.Editorial.

BROW, S.; VRANESIC, Z. (2000). Fundamentals of Digital Logic with VHDL Design. Mc Graw-Hill Series in Computer Enginneering.

BROW, S. (1992). Routing Algorithms and Architectures for Field Programmable Gate Arrays. Ph.D. Thesis, Department of Electrical Engineering, University of Toronto.Canada.

CASILLO, A. L. (2003). VHDL – VHSIC *Hardware* Description Language. Apostila, Disciplina Organização e Arquitetura de Computadores I, Universidade Federal do Rio Grande do Norte. Disponível em <http//: www.dimap.ufrn.br/~ivan/orgl/vhdl.pdf >. Acesso em 12/05/2004.

CHAN, P. K. (1994). Digital Design Using Field Programmable Gate Arrays. New Jersey: Prentice Hall.

COFFMAN, K. (1999). Real World FPGA design with Verilog. New Jersey: Prentice Hall.

COMPTON, K. (1999). Programming Architectures for Run Time Reconfigurable Systems. Master's Thesis. Dept. of ECE. Northwestern University Evanston. IL. U.S.A. Disponível em <http:// www.ee.washington.edu/faculty/hauck/publications-/katiThesis.pdf >. Acesso em 16/12/2003.

CORETTI, J.A. (1998). Manual de Treinamento Básico de Controlador Lógico Programável. Centro de Treinamento SMAR, Sertãozinho, SP.

COSTA, C. (1990). Automação do Chão de Fábrica – Um Passo Fundamental Rumo ao CIM. XXIII Congresso Internacional de Informática e II Congresso Internacional de Informática. SUCESSU – Sociedade dos Usuários de Computadores e Equipamentos Subsidiários. Rio de Janeiro.

CSIRO Manufacturing & Infrastructure Technology (2004). Machine Vision: CLP – Configurable Logic Processor VME Module.Press release. Disponível em <http://vision.cmit.csiro.au/expertise/clp >. Acesso em 15/12/2004.

ELLIPTIC Semicinductor Inc.(2004). CLP-01 Pseudo Random Number Generator Core. Preliminary Data sheet**.** Disponível em <http://www.ellipticsemi.com/product_information.html#internalidentifier>. Acesso em 15/12/2004.

FELIX, P. (2002). Organização para a Arquitetura ARM. Notas de aula para a Disciplina de Arquitetura de Computadores, Instituto Superior de Engenharia de Lisboa, Lisboa, Portugal. Disponível em http://www.cc.isel.ipl.pt/pessoais/pedrofelix/ac/slides/organizaçoesparaarquitetura arm.pdf. Acesso em 09/01/2004.

FERNANDES, L. M. J. (1994). Redes de Petri e VHDL na Especificação de Controladores Paralelos. Dissertação (Mestrado) – Departamento de Informática, Universidade do Minho, Braga, Protugal.

FERREIRA, R. S.; CHIEPPE, U.; FREITAS, G. C.; BIANCARDI. , C. (2003). *Software* Livre no Ensino de Sistemas Digitais e Arquitetura de Computadores. Universidade Federal de Viçosa, Departamento de Ciência da Computação, Viçosa, MG. Disponível em <http://www.weci.spc.org.pe/weci2003/download/009-RFerreira.pdf>. Acesso em 09/01/2004.

GE FANUC Automation, Inc. (2004). Programmable Automation Controllers (PACs) – Flexibility, Openness and Performance in the Next Generation of Control. GE Fanuc Automation Information Centers. Disponível em <http://www.gefanuc.com/PAC>. Acesso em 03/01/2005.

GEORGINE, M. (2000). Automação Aplicada Descrição e Implementação de Sistemas Seqüenciais com PLCs. São Paulo. Editora Érica.

GONSALES, A.; CARRO, L.; LUBASZEWSKI, M.; SUZIN, A. (2001). Projeto de um PLD com Características de Testabilidade. GME – Grupo de Microeletrônica, UFRG – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS. Disponível em <http://www.iberchip.org/VII/cdnav/pdf/60.pdf>. Acesso em 14/02/2004.

HENNESSY, J.L; PATTERSON, D.A; (1997). Computer Organization and Design: The *Hardware/Software* Interface. Second Edition. San Francisco: Morgan Kaufmann Publishers, Inc.

.

HERVELLA, C. (1998). Projeto e Desenvolvimento de um Sistema de Controlador Programável Flexível para Manipuladores e Robôs Industriais. Dissertação (Mestrado), Universidade de Campinas, Campinas.

JASINK, R.P. (2001). Introdução a Linguagem VHDL. Relatório Técnico, Laboratório de Microeletônica, CPDTT, CEFET-PR, Curitiba.

KUGLER, M.; JUNIOR, T. J.; LOPES, S. H. (2003). Desenvolvimento de uma Rede Neural LVQ em Linguagem VHDL para Aplicação em Tempo Real. VI Congresso Brasileiro de Redes Neurais, pp 103-108, Centro Universitário da FEI, São Paulo.

LEDGARD, H. (1981). Ada – An Introduction. Editora Campus. Rio de Janeiro. Brasil.

LEAP ELECTRONIC CO.(2002). CPLD Logic Design and Practices. Instruction Book. First dition. Taiwan, China.

LEUTRON Vision AG.(2002). Products PicPort- Pro-CL-PMC, PicProdigy – Color and PicProdigy – Color - CLP. Informations about News products**.** Disponível em <http://www.leutron.com/english/product/pprocl_pmc_i.htm>. Acesso em 15/12/2004.

LOPES, A., ALMEIDA, F., NEUMANN, J., PINHEIRO, V. (1999). Dispositivos Lógicos Programáveis- PLD. Trabalho Disciplina Circuitos Integrados, CEFET-RJ, Rio de Janeiro. Disponível em <http//: www.cefetrio.hpg.ig.com.br/ciencia_e_educacao/8/CI/pld2/default.htm>. Acesso em 16/12/2003.

MELO, F .L.; LIMA, E. R. C.; ROSARIO, M. J. (2004). Robô Móvel com Sistema de Arquitetura Aberta Utilizado na realização de Tarefas de Inspeção das Paredes Internas e Externas de Dutos. Anais do IV Congresso Internacional de Automação, Sistemas e Instrumentação. ISA – Instrumentation Systems, and Automation Society, District 4 (South América). Novembro, São Paulo, Brasil.

MEN Mikro Elektronik. (2004). Flexible and Independent With FPGA Technology. FPGA Technology. Disponível em <http://www.men.de /menrail/02_fpga.asp?lid= >. Acesso em 16/12/2004).

MESQUITA, D.G. (2002). Contribuições para Reconfigurações Parcial, Remota e Dinâmica de FPGAs. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Pós Graduação em Ciência da Computação, Porto Alegre.

MESQUITA, D.G; MORAES, F. (2000). Implementação de Sistemas Digitais Reconfiguráveis Parcial, Remota e Dinamicamente. Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre. Disponível em <http://www.inf.pucrs.br/~dmesquita/interest/itens/pep.ppt>. Acesso em 07/05/2004.

MESQUITA, D.; MORAES, F.; MOLLER, L.; CALAZANS,N. (2000); Reconfiguração Parcial e Remota de Dispositivos FPGA da Família Virtex. Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre.

MILANI, G.C.A. (1998). SPmm1 Uma Máquina Paralela Reconfigurável. Dissertação (Mestrado), Universidade Federal de São Carlos, São Carlos.

MORAES, F.G.; CALAZANS, N.L.V.; FERREIRA, E.H.; LIEDKE, D.C. (2000). Implementação Eficiente de uma Arquitetura Load/Store em VHDL. Faculdade de Informática. Pontifícia Universidade Católica do Rio Grande do Sul.

MORAES, C.C.; CASTRUCCI, P.L. (2001). Engenharia de Automação Industrial. Rio de Janeiro.LTC.

MOREIRA, P.J.L.; JOELSON, E., CHACON, . (1999). HDL - *Hardware* Description Language. Trabalho Disciplina Circuitos Integrados, CEFET-RJ, Rio de Janeiro. Disponível em <http:// www.cefeterio.hpg.ig.com.br/cienciaeeducacao/8/CI/trabalhos992/hdl/hdl.htm>. Acesso em o7/05/2004.

NATIONAL INSTRUMENTS Corp. (2004). NI CompactRIO – Reconfigurable Control And Acquisition System. Application Notes and Tutorials. Austin Texas. Disponível em <htpp://www.ni.com/compactrio>. Acesso em 16/12/2004.

NATIONAL INSTRUMENTS Corp. (2003). Developing Measurement and Control Applications with the LabView FPGA Pionneer System. White papers. Disponível em <htpp://www.ni.com/info>. Acesso em 16/12/2004.

NATIONAL INSTRUMENTS Corp. (2004). LabVIEW FPGA Module. Bulletim technical. Austin Texas**.** Disponível em <htpp://www.ni.com/pdf/products/us/labview_fpga_module.pdf>. Acesso em 16/12/2004.

OLIVEIRA, R.A.N.; RUBENS, G.A.;(1983). Microprocessador Z-80 – *Hardware*. A e N Consultoria, Projetos e Publicações. Rio de Janeiro, Brasil.

PERRY, D.L. (1998). VHDL. 3.a Edição. New York : Mcgraw-Hill.

PUPO, S.M. (2002). Interface Homem Máquina para Supervisão de um CLP em Controle de Processo. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos.

ROSE, J.; GAMAL, A. E.; SANGIOVANNI, A. (1993). Architecture of Field Programmable Gate Arrays. In proceedings of the IEEE; vol. 81, no. 7, pp. 1013-1029.

TEIXEIRA, M. A. (2002). Técnicas de Reconfigurabilidade dos FPGAs da Família APEX 20K Altera. Dissertação (Mestrado), Universidade de São Paulo, São Carlos. Disponível em <http//:

www.teses.usp.br/teses/disponiveis/55/55134/tde-11092002-164901/publico/texto.pdf >. Acesso em 09/01/2004.

TOOD, W. (2004). PAC: The Next – Generation PLC. News Releases. Disponível em <htpp:// www.ni.com>. Acesso em 16/12/2004.

WAKERLY, J. F. (2000). Digital Design – Principles & Practices. 3th Edition. New Jersey : Prentice Hall.

WARNOCK, I. G. (1997). Programmable Controllers Operational and Application. Europe: Prentice Hall.

VOORHORST, R. C. (2004). How to overcome the limitations of PLCs by using Programmable Automation Controllers. News Releases**.** Disponível em <http://www.ni.com/netherlands>. Acesso em 16/12/2004.

ZAGHETTO, A.; PRADO, A. C.; TAVARES, A. (2001). Trabalho sobre Dispositivos Lógicos programáveis – FPGA. Departamento de Engenharia Eletrônica e de Computação, Universidade Federal do Rio de Janeiro. Disponível em <http//:www.gta.ufrj.br/grad/01_1/pld/hcpld.htm>. Acesso em 07/05/2004.

APÊNDICE 1 – LISTAGEM DOS PROGRAMAS ESTUDADOS

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo