Universidade Estadual de Campinas - UNICAMP
Faculdade de Engenharia Elétrica e de Computação - FEEC
Departamento de Sistemas de Controle e Energia - DSCE
Laboratório de Sistemas Modulares Robóticos - LSMR

Sistema para controle de máquinas robotizadas utilizando Dispositivos Lógicos Programáveis

Luiz Eduardo Guardia Filho

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

Banca Examinadora:
Prof. Dr. Marconi Kolm Madrid (Orientador) - DSCE/FEEC/UNICAMP
Prof. Dr. Alexandre César Rodrigues da Silva - FEIS/UNESP
Prof. Dr. José Raimundo de Oliveira - FEEC/UNICAMP
Dr. Edson Adriano Vendrusculo (Professor Associado) - FEEC/UNICAMP

Campinas, julho de 2005

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Guardia Filho, Luiz Eduardo
G931s Sistema para controle de máquinas robotizadas utilizando dispositivos lógicos programáveis / Luiz Eduardo Guardia Filho. --Campinas, SP: [s.n.], 2005.

Orientador: Marconi Kolm Madrid
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Robótica. 2. Diapositivos Lógicos programáveis. 3. Hardware. 4. Eletrônica digital. 5. Circuitos lógicos. 6. VHDL (Linguagem descritiva de hardware). Processamento paralelo (Computadores). I. Madrid, Marconi Kolm. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: System to control of robotic machines using programmable logic devices
Palavras-chave em Inglês: Robotics, Programmable logic devices, Hardware, Digital electronic, Logic circuits, VHDL (Very hardware description language e Parallel processing (Computers).
Área de concentração: Automação e Controle
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Alexandre César Rodrigues da Silva, José Raimundo de Oliveira e Edson Adriano Vendrusculo.
Data da defesa: 06/07/2005

Agradecimentos

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

- Ao meu orientador, professor Marconi Kolm Madrid, pela excelente orientação, amizade, companheirismo e pela crença na minha capacidade intelectual.
- Aos professores da graduação, Alexandre, Nobuo e Edvaldo, pelo incentivo na continuação da carreira acadêmica.
- Ao professor José Raimundo de Oliveira pela disponibilização do programa Quartus e pelos valiosos momentos de discussão para que esse trabalho pudesse ser realizado.
- Ao amigo Fabrício Nicolato pelos incentivos e companheirismo em todos os momentos desse trabalho.
- Ao amigo Filipe Ieda Fazzanaro pelos constantes apoios na escrita desse trabalho.
- Aos amigos Héder, Mário, Eberval, e Bonani, pelo companheirismo dentro do LSMR e cujas contribuições foram importantes para a finalização deste trabalho.
- Aos meus tios Luiz Carlos, Angélica e Abigail pelo acolhimento, carinho, conselhos e apoios para a realização de mais esse sonho.
- A toda minha família e amigos pelo apoio e incentivo nos momentos difíceis.
- Ao convênio CAPES pelo apoio financeiro.

Resumo

Este trabalho de mestrado teve o propósito de projetar e construir um sistema de *hardware* capaz de realizar o controle de máquinas robotizadas em tempo real. Foi dada uma abordagem usando técnicas de processamento paralelo e eletrônica reconfigurável com o uso de dispositivos lógicos programáveis. Mostrou-se em função dos resultados das implementações que o sistema proposto é eficiente para ser utilizado no controle de robôs baseado em modelos matemáticos complexos como cinemático direto/inverso, dinâmico e de visão artificial. Esse mesmo sistema prevê sua utilização para os quatro níveis hierárquicos envolvidos em plantas que se utilizam de controle automático: supervisão, tarefas, trajetória e servomecanismos. O sistema possui interfaces de comunicação *USB* e *RS-232*, conversores A/D e D/A, sistema de processamento de imagens (entradas e saídas de sinais de vídeo analógico), portas E/S, chaves e *leds* para propósito geral. A eficiência foi comprovada através de experimentações práticas utilizando sistemas robóticos reais como: sistema de um pêndulo acionado, robô redundante de 4GDL denominado Cobra, e solução em *hardware* de funções importantes no sentido da resolução dos modelos matemáticos em tempo real como funções transcendentais.

Palavras-chave: robótica, dispositivos lógicos programáveis, computação paralela, *hardware* para computação paralela.

Abstract

This work had as purpose the project and build of a hardware system with abilities to accomplish the real time control of robotic machines. It was given an approach using techniques of parallel processing and programmable electronics configuration with programmable logic devices. According to the implementation results, it was shown that this proposed system is efficient to be used for controlling robots based on complex mathematical models, like direct/inverse kinematics, dynamics and artificial vision. This system foresees its use for the four hierarchical levels involved in industrial plants that use automatic control: supervision, tasks, trajectory/path and servomechanisms. The system has USB and RS-232 communication interfaces, A/D and D/A converters, image processing capabilities (with input/output for analog video signals), I/O ports, and switches and leds for general purpose. Its efficiency is demonstrated through practical experimentations using real robotic systems as: a driven pendulum system, a redundant 4 DOF robot called "Cobra", and a hardware solution for important functions in the sense of real time mathematical models computing, like the transcendental functions.

Keywords: robotics, programmable logical devices, parallel computing, hardware for parallel computation.

A minha mãe, irmã, avó, avô (in memorian) e minha namorada Priscila

Sumário

Lista de Abreviaturas

A/D Analógico/Digital;

AS Active Serial;

ASIC Application Specific Integrated Circuit;

BGA Ball Grid Array;

CD Compact Disc;

CPU Central Processing Unit;

D/A Digital/Analógico;

DOS Disk Operating System;

DSP Digital Signal Processing;

EDA Eletronic Design Automation;

EEPROM Electrically-Erasable Programmable Read-Only Memory;

EPROM Erasable Programmable Read-Only Memory;

FPAA Field Programmable Analog Arrays;

FPGA Field Programmable Gate Arrays;

GDL Graus de liberdade;

HDL Hardware Description Language;

I/O Input/Output;

IOE I/O Element;

ISA Industry Standard Architecture;

JTAG Join Test Action Group;

LAB Logic Array Blocks;

LE Logic Elements;

LED Light Emitting Diodes;

LPM Library of Parameterized Modules;

LSMR Laboratório de Sistemas Modulares Robóticos;

LUT Look-up Tables;

MUX Multiplex;

OTP One-Time Programming;

PAL Programmable Array Logic;

PC Personal Computer;

PCI Peripheral Component Interconnect;

PLA Programmable Logic Arrays;

PLD Programmable Logic Devices;

PLL Phase-Locked Loop;

PROM Programmable Read-Only Memory;

PWM Pulse Width Modulation;

RAM Random Access Memory;

RISC Reduced Instruction Set Computer;

RTC Run-Time Configuration;

RTL Register Transfer Level;

SDRAM Synchronous Dynamic Random Access Memory;

SRAM Static Random Access Memory;

ULA Unidade Lógica Aritmética;

USB Universal Serial Bus;

VHDL VHSIC Hardware Description Language;

VHSIC Very High Speed Integrated Circuit.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução, Motivação e Organização

1.1 Introdução

A necessidade atual de se realizar implementações em hardware tem-se intensificado acentuadamente, principalmente devido à crescente demanda pela produção em larga escala de produtos e serviços mais elaborados e de grande confiabilidade. A isso está associado o invento, ou aprimoramento do funcionamento de máquinas automáticas que necessitam de sistemas de controle mais sofisticados, que empregam modelos matemáticos auxiliares de grande complexidade, para que as muitas tarefas que são atribuídas a elas sejam realizadas satisfatoriamente. Estes modelos, geralmente, envolvem o cálculo de equações que possuem enormes quantidades de operadores e funções transcendentais com características não-lineares. Além disso, esses cálculos, devem ser realizados aos milhares em tempos da ordem dos milissegundos, como no caso dos sistemas de controle de máquinas robotizadas. Isso se deve à complexidade dos problemas abordados que requerem grande esforço e recursos computacionais (alto custo) podendo chegar, em alguns casos, a serem intratáveis por sistemas computacionais baseados em processamento seqüencial.

Os sistemas robóticos são constituídos por sub-sistemas (níveis), ver Figura 1.1, que podem operar paralelamente. O paralelismo está presente em diversos níveis, tornando tais sistemas candidatos a serem beneficiados pelas estratégias de computação paralela.

Na segunda metade da década de 80 surgiram os primeiros dispositivos lógicos programáveis (*Programmable Logic Devices - PLD*), como resultado do aprimoramento dos precursores *PLA*(*Programmable Logic Arrays*). Tais dispositivos permitem o projeto de circuitos digitais com auxílio por programação, e também durante as etapas de desenvolvimento que os projetos sejam testados e modificados quantas vezes forem necessárias, e com acréscimo muito reduzido de custo. Isso dá aos *PLD* uma grande versatilidade, abrindo possibilidades de emprego num vasto campo de aplicações, tais como: comunicações (Denning *et al.*, 2004), processamento

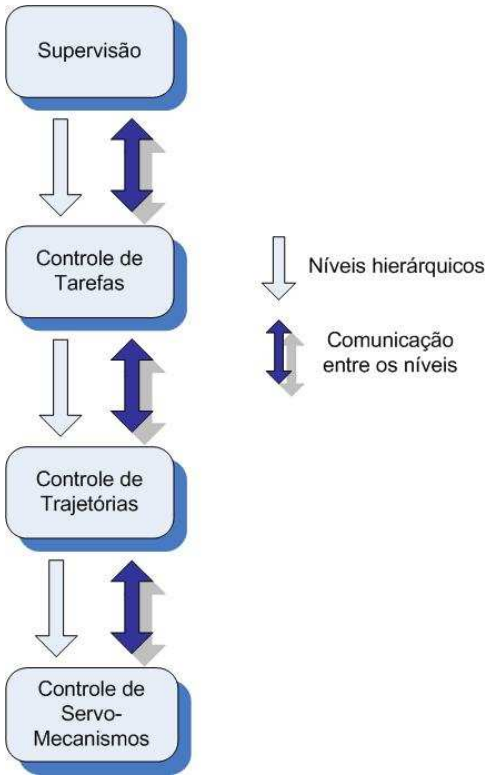

Figura 1.1: Níveis robóticos de um sistema típico.

digital de áudio (VoB *et al.*, 2004), processamento de imagens (Boluda *et al.*, 1999) (Kolodko e Vlacic, 2003), inteligência artificial (Martinez-Alvarez *et al.*, 2002) (Tang e Yip, 2004), sistemas robóticos (Hoshi *et al.*, 2004) (Bailak *et al.*, 2004) (He *et al.*, 2004) etc.

O desenvolvimento de dispositivos de lógica programável possibilita a criação de sistemas digitais complexos em um único componente, utilizando um único ambiente de desenvolvimento. Possibilita também grande integração entre *hardware* e *software*, permitindo que o projetista escolha quais funções devem ser realizadas em *hardware* e quais devem ser realizadas em *software*, alterando assim o conceito tradicional que se tinha sobre o desenvolvimento de *hardware* e *software* onde cada parte era geralmente planejada e implementada por equipes diferentes (Vahid, 2003).

Com o ambiente de desenvolvimento *Quartus II* (QuartusII, 2005) e o *SOPC Builder* (SOPC, 2005), pode-se programar o processador NIOS (NIOSI, 2005) (NIOSII, 2005), que pode ter a precisão (número de *bits*) escolhida de acordo com cada problema em questão. Através dele pode-se realizar, por exemplo, o controle de periféricos como acesso a memórias, dispositivos de entrada/saída e fluxo de dados além de poder-se implementar arquiteturas de controle tanto monoprocessadas como multiprocessadas (de Aguiar Dias, 1991).

Embora os *PLD* atuais ainda operem com freqüências de relógio (*clock*) inferiores às

dos microprocessadores, eles são geralmente superiores em eficiência devido à característica inerentemente paralela de suas arquiteturas. O projetista pode construir o próprio conjunto de instruções, que seja o mais adequado a uma determinada tarefa, e, posteriormente, substituí-lo por outro sempre que se fizer necessário, ou desejável, tornando os sistemas muito flexíveis para alterações e expansões.

Nos últimos anos, a tecnologia dos PLD tem evoluído significativamente, alcançando elevados níveis de densidade, altos índices de desempenho e menores custos de fabricação. Esta evolução tem tornado cada vez menor a distância das capacidades computacionais entre *PLD* e circuitos integrados de aplicação específica (*Application Specific Integrated Circuit - ASIC*). Além dos avanços em capacidade, desempenho e custos, os fabricantes de *PLD* têm introduzido cada vez mais recursos de reconfigurabilidade.

1.1.1 Histórico LSMR

O Laboratório de Sistemas Modulares Robóticos (LSMR) foi criado cerca de 20 anos e, desde então, a construção e o estudo de estruturas robóticas vem sendo pesquisadas. Neste laboratório foram projetados e construídos 4 (quatro) protótipos de sistemas robóticos.

O primeiro, com 4 (quatro) graus de liberdade, denominado de **JECA I** (Figura 1.2), todo o sistema de controle foi implementado utilizando-se microprocessadores de 8 *bits* por razões tecnológicas. Fez-se uso de uma arquitetura monoprocessada para o controle e acionamento das juntas (Madrid, 1988). Já o segundo, um robô com 5 (cinco) graus de liberdade mais garra de 2 (dois) dedos, nomeado como **JECA II** (Figura 1.2) teve todo seu sistema de controle projetado e implementado utilizando uma arquitetura paralela multiprocessada (mestre/escravos). Nesse trabalho, foram utilizados microprocessadores de 8 *bits* onde cada junta tinha seu próprio controle e acionamento (Madrid, 1994).

O terceiro, um pêndulo acionado, consiste de um sistema mecânico simples composto de dois eixos e engrenagens do tipo polias e correias sincronizadas (Figura 1.3). Nesse trabalho, foi proposta uma técnica que emprega algoritmos genéticos e teoria dos conjuntos nebulosos integrados, visando o desenvolvimento automático de controladores de alta performance para servomecanismos tipo elo-acionado, ou módulo de junta robótica. Todos os algoritmos foram implementados em linguagem C. A aquisição da leitura do *encoder* e geração do sinal de *PWM* foram realizados por um *hardware* externo e enviados a um PC através de uma placa de comunicação de dados *ISA*, construída no LSMR (de Souza, 2000). Com esse mesmo protótipo foi proposta uma técnica que emprega redes neurais diferentemente daquela que emprega teoria dos conjuntos nebulosos (Jungbeck, 2001).

Foi realizado um trabalho na área de visão computacional, onde foi construído um sistema de *hardware* capaz de extrair características em imagens de vídeo. Foi dada uma

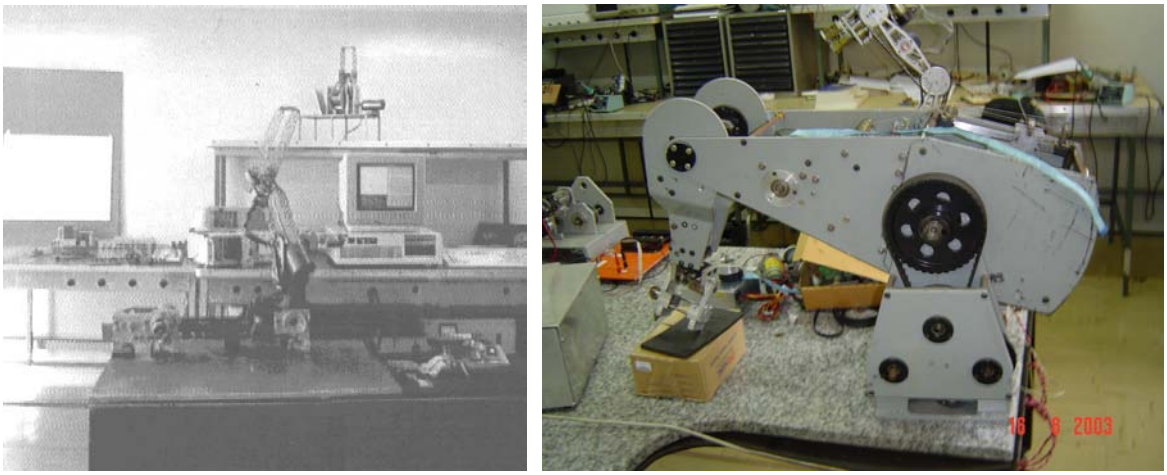

Figura 1.2: Estrutura do robô JECA I (à esquerda) e estrutura do robô JECA II.

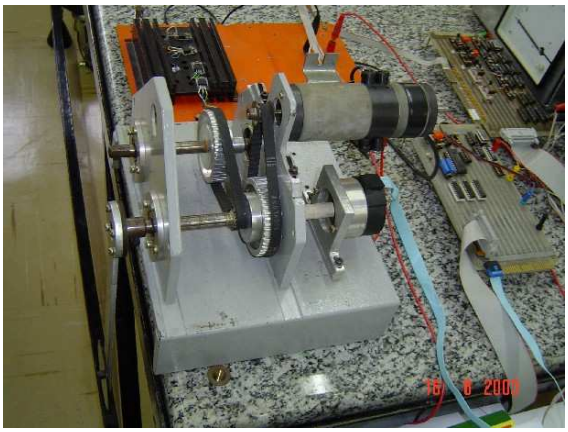

Figura 1.3: Bancada experimental com a montagem do pêndulo acionado.

abordagem usando técnicas de processamento paralelo e eletrônica reconfigurável com uso de dispositivos lógicos programáveis. Para realização desse trabalho, foi adquirido um *kit* de desenvolvimento *Excalibur*, ver Figura 1.4, da Altera (Nicolato, 2002).

O quarto e mais recente robô projetado e construído no LSMR consiste de um robô planar redundante com 4 (quatro) graus de liberdade, nomeado Cobra. Nesse trabalho está sendo empregado método de busca heurística para controle e geração de trajetórias. A implementação do algoritmo está sendo feita utilizando o *xPC Target* do *Matlab/Simulink*. A implementação conta com auxílio de um *hardware* que gerencia os dados que vêem de uma placa *ISA* para aquisição dos sinais dos *encoders*, chaves fim de curso e envio dos sinais de *PWM* (Nicolato, 2005). Esta estrutura robótica está mostrada na Figura 1.5.

Está em desenvolvimento também um trabalho de pesquisa para controladores de tempo real usando um robô Rhino XR3 (Figura 1.6) como paradigma e técnica de inteligência artificial.

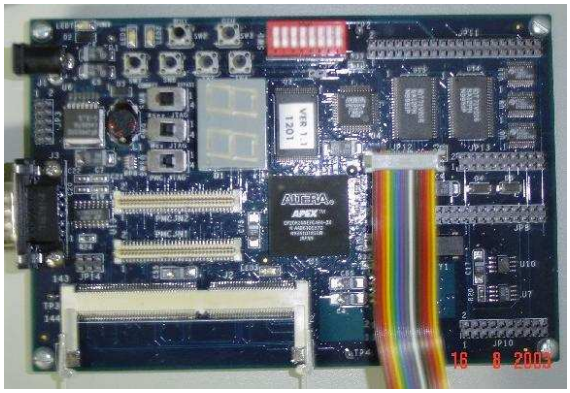

Figura 1.4: Kit de desenvolvimento Altera.

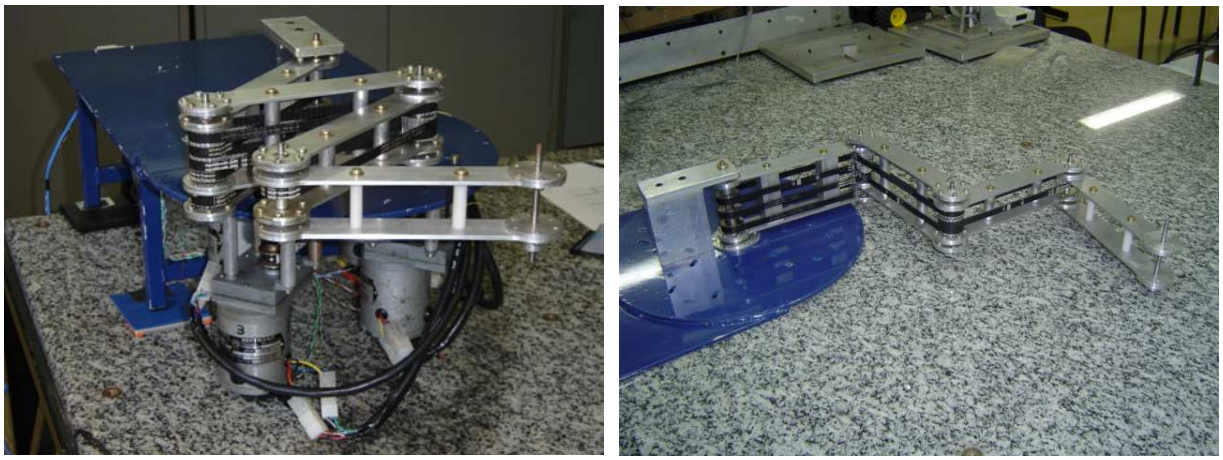

Figura 1.5: Robô COBRA desenvolvido no LSMR.

A técnica está sendo implementada utilizando o *Real Time* do Linux para garantia de tempo real. O sistema está concebido para utilizar uma placa contendo um *PLD* para aquisição dos sinais de *encoder* e chaves fim de curso para que sejam enviados a um PC através de uma placa de aquisição de dados *PCI* da *National Instruments* (Jungbeck, 2005).

Além desses, existem outros protótipos experimentais e trabalhos que servem de apoio para o desenvolvimento de trabalhos tanto de iniciação científica quanto de pesquisas de pós-graduação.

1.2 Motivação

Durante esses 20 anos do LSMR, muitas estruturas robóticas foram construídas e outras se encontram em fase de conclusão e vários sistemas de controle foram projetados e implemen-

Figura 1.6: Robô Rhino XR3 utilizado no trabalho.

tados sendo que todos se utilizaram das melhores tecnologias disponíveis em cada época.

Os sistemas robóticos comerciais apresentam custos muito elevados comparados com a realidade dos laboratórios de pesquisa nacionais, e também com o orçamento de pequenas e médias empresas. Uma célula robótica, por exemplo, chega a custar 120.000 dólares além de serem sistemas fechados onde o usuário não tem acesso as partes internas (pinos, sinais etc). Um outro contratempo é quando esses sistemas necessitam de manutenção onde nesse caso o usuário fica totalmente dependente dos fabricantes para dar-lhes manutenção, o que também é um processo com custos bem elevados (geralmente precisa trazer técnicos do exterior para dar manutenção).

Assim, neste trabalho projetou-se e construiu-se um sistema para controle de máquinas robotizadas que possa atender as necessidades atuais e futuras do LSMR apontadas durante esses 20 anos de existência, que possa atuar em todos os níveis apresentados na Figura 1.1 e, ao mesmo tempo, trazer uma tecnologia totalmente nacional sem que haja necessidade de importação toda vez que for desenvolvida uma nova proposta para sistema de controle.

1.3 Objetivos

Os principais objetivos desse trabalho foram:

- Trazer para o LSMR mais tecnologia de desenvolvimento de *hardware* com emprego de *PLD*;
- Ampliar e aprimorar os conhecimentos na área de arquitetura de computadores;

- Ampliar e aprimorar os conhecimentos na tecnologia da implementação de circuitos lógicos em dispositivos lógicos programáveis;
- Construir um sistema para controle de máquinas robotizadas que seja capaz de realizar todos os níveis hierárquicos de controle para sistemas robóticos;
- Construir um sistema que seja modular, com alta capacidade de processamento, expansível e de baixo custo.

1.4 Organização

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

O capítulo 2 apresenta características importantes dos Dispositivos Lógicos Programáveis, introdução a ferramenta *xPC Target*, bem como uma introdução ao conceito de Placa de Circuito Impresso (PCI) para as proposições feitas nesse trabalho.

O capítulo 3 apresenta o *hardware* projetado com a descrição detalhada de cada bloco.

O capítulo 4 apresenta algumas aplicações realizadas dentro do LSMR para mostrar e comprovar o funcionamento do sistema de controle desenvolvido.

O capítulo 5 apresenta as conclusões do trabalho, as perspectivas, assim como sugestões para trabalhos futuros.

Capítulo 2

Bases Técnicas e Tecnológicas

Este capítulo apresenta características importantes dos Dispositivos Lógicos Programáveis, introdução a ferramenta xPC Target, bem como uma breve introdução sobre Placas de Circuito Impresso (PCI).

2.1 Introdução

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

O primeiro tipo de componente programável pelo usuário e apto a implementar funções lógicas foi a *Programmable Read-Only Memory* (*PROM*), na qual as linhas de endereço podem ser usadas como entradas do circuito lógico, e as linhas de dados como saídas. Como funções lógicas raramente requerem termos-produto, e uma *PROM* contém um decodificador completo para os seus endereços de entrada, ela é um tipo de arquitetura ineficiente para a implementação de circuitos lógicos e, portanto, dificilmente usada na prática para este propósito. Posteriormente, outros dispositivos desenvolvidos para implementar circuitos lógicos foram os *Programmable Logic Arrays* (*PLA*) e os *Programmable Array Logic* (*PAL*), que são dispositivos com dimensões físicas relativamente pequenas constituídos por dois níveis de lógica, um plano *E* (*AND*) seguido por um plano *OU* (*OR*). Os *PLA*, *PAL* e vários pequenos *PLD* estão agrupados em uma categoria denominada de *PLD* simples (*SPLD*), cujas características mais importantes são o baixo custo e o alto desempenho em velocidade de processamento. Com o avanço da tecnologia, tornou-se possível construir dispositivos com capacidades mais elevadas

que os *SPLD*. A dificuldade de se aumentar a capacidade de uma arquitetura *SPLD* rígida é devido à estrutura dos planos lógicos programáveis crescer muito rapidamente em função do aumento do número de entradas (Brown e Rose, 1996). O único modo factível de se prover uma maior capacidade para os dispositivos baseados na arquitetura *SPLD* é através da integração de múltiplos *SPLD* em um único componente, de modo a conectar seus blocos via programação. Atualmente, existem vários tipos de *PLD* no mercado baseados nesse tipo de arquitetura (Atmel, 2001; Lattice, 2002; Altera-Max, 2001; Xilinx, 2002). Eles são coletivamente referenciados como *PLD* complexos (*CPLD*).

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

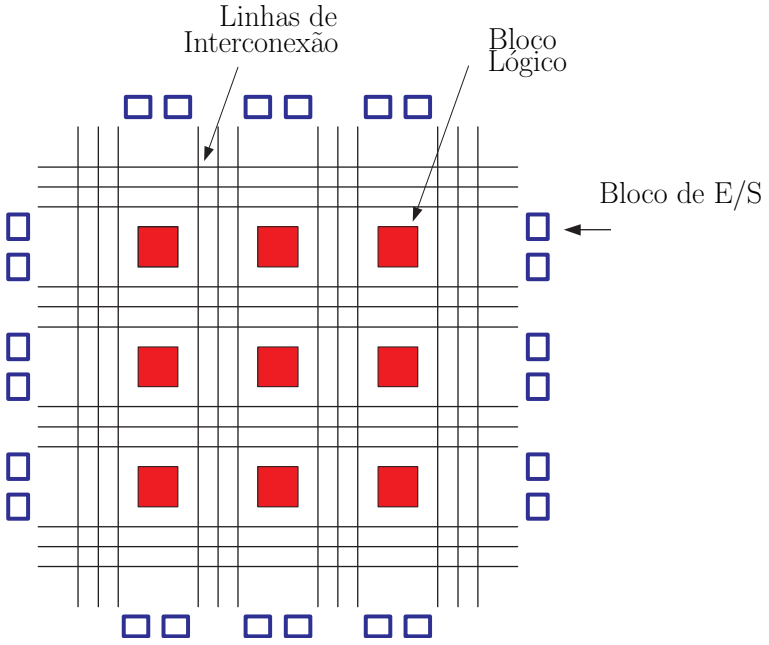

Figura 2.1: Estrutura básica de um *FPGA*.

2.1.1 Métodos de Configuração

Dependendo de como são construídos, os *PLD* dividem-se em duas categorias: os programáveis apenas uma vez (*one-time programming* - *OTP*) e os reprogramáveis. Esses últimos,

por sua vez, dividem-se em: reconfiguráveis estaticamente e reconfiguráveis dinamicamente. Os reconfiguráveis estaticamente são configurados antes ou depois da realização de uma tarefa, e não permitem configuração parcial. Na reconfiguração dinâmica, também chamada de *run-time reconfiguration* (RTR), *on-the-fly reconfiguration* ou *in-circuit reconfiguration*, as mudanças estruturais na configuração do PLD podem ser feitas durante a realização de uma tarefa, isto é, o sistema digital é dividido em sub-circuitos que são configurados no componente enquanto em operação.

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

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

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

2.1.2 Projeto com Dispositivos Lógicos Programáveis

O projeto de sistemas digitais destinados a implementação em dispositivos lógicos programáveis é realizado por ferramentas denominadas *Eletronic Design Automation* - EDA). Uma ferramenta de EDA para PLD típica inclui programas para as seguintes tarefas: entrada inicial

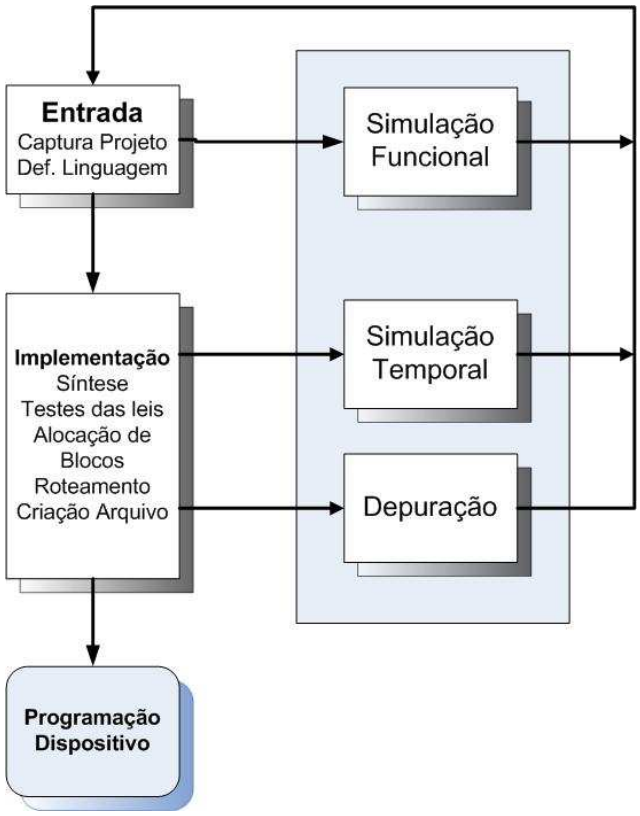

Figura 2.2: Fluxo de projeto em lógica programável.

para o projeto, otimização, adequação ao dispositivo, simulação e configuração, Figura 2.2.

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

Tradicionalmente, as ferramentas baseadas em diagramas esquemáticos propiciam maior controle sobre o particionamento e posicionamento da lógica no dispositivo. Porém, tal benefício vem em detrimento do tempo gasto na realização do projeto, uma vez que o mesmo é executado no nível de portas lógicas. Já, as ferramentas baseadas em linguagens de descrição permitem que o projeto seja realizado num nível de abstração mais elevado (por ex. descrição comportamental em $VHDL$), possibilitando maior rapidez no processo de desenvolvimento. As desvantagens comuns das HDL são a diminuição do desempenho e a densidade lógica necessária (quantidade de lógica implementada por unidade de área do componente) (OptiMagic, 2000).

Atualmente, os principais fabricantes de PLD disponibilizam bibliotecas de funções (em alguns casos, parametrizadas) básicas (Altera-LPM, 1996); tais como somadores, multiplica-

dores, multiplexadores, memórias etc, que podem ser instanciadas a partir de editores de esquemáticos ou por meio de *HDL*. Adicionalmente, há uma ampla pesquisa sobre tradutores que fazem a conversão de programas escritos em linguagens de alto nível (Sankaran e Haggard, 2001; Rinker *et al.*, 2001).

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

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

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

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

É importante salientar que, seja qual for a tecnologia, ou componentes escolhidos para implementar o projeto, o conhecimento da arquitetura dos componentes, bem como das ferramentas de projeto ajuda bastante na obtenção de melhores resultados.

2.2 Síntese Utilizando VHDL

Um forte atrativo das linguagens de descrição de *hardware*, e mais especificamente a *VHDL*, é a possibilidade de empregá-las na síntese de dispositivos *ASIC* e *PLD*. A síntese é um método automático de conversão de uma descrição de alto nível para outra de mais baixo

nível de abstração. Há várias ferramentas de síntese disponíveis atualmente, entre as quais pode-se citar: *Leonardo Spectrun*, *FPGA Express*, *Symplify*, *Synopsys* etc. Essas ferramentas convertem descrições apresentadas em formato *Register Transfer Level* (RTL) para *netlists* no nível de portas lógicas. Estes *netlists* consistem de macro-células inter-conectadas, e os modelos para as células no nível de portas lógicas ficam contidos em bibliotecas específicas para cada tecnologia utilizada ou suportada. Um *netlist*, geralmente, pode ser otimizado por área e por velocidade. As entradas para o processo de síntese são as descrições RTL em $VHDL$, as restrições e atributos do circuito, e a(s) biblioteca(s) da tecnologia alvo.

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

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

A tradução da descrição RTL para a descrição booleana equivalente, usualmente, não é controlada pelo usuário. Por outro lado, a descrição intermediária gerada, geralmente, tem um formato otimizado por ferramentas particulares e pode ficar transparente para o usuário. Todas as estruturas IF, $CASE$ e $LOOP$, associações condicionais de sinais, e associações de sinais selecionados, são convertidas para suas formas booleanas equivalentes durante o processo intermediário. Os *flip-flops* e *latches* também poder ser instanciados ou inferidos; em ambos os casos mantêm-se os mesmos *flip-flops* ou *latches* na descrição intermediária.

A seguir são apresentados algumas características de uma família de dispositivos lógicos programáveis denominada *Cyclone*.

2.3 Arquitetura *Cyclone*

Freqüentemente, muitos projetistas de sistemas usam múltiplos PLD num mesmo circuito, pois existem arquiteturas mais adequadas para cada tipo de aplicação. Os dispositivos da família *Cyclone* combinam características de quatro famílias de PLD diferentes. Isso per-

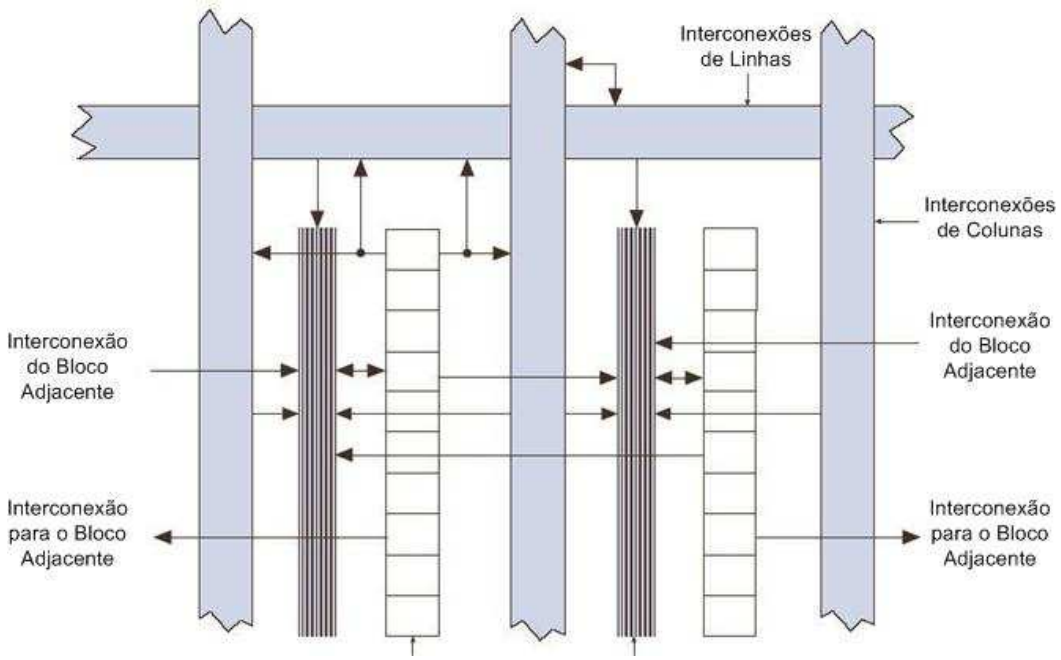

Figura 2.3: Estrutura de uma *LAB*.

mite realizar a implementação do projeto completo em um único componente. A característica mais importante dos dispositivos *Cyclone* é seu excelente custo/benefício em relação as demais famílias da Altera (Cyclone, 2005).

A arquitetura *MultiCore* da família *Cyclone* é constituída por conjuntos de blocos lógicos (*logic array blocks - LAB*) que, por sua vez, são constituídos por elementos lógicos (*logic elements - LE*). Cada *LAB* consiste de 10 *LE*, sinais de controle das *LAB*, ligações locais, conjunto de tabelas de procura (*look-up tables - LUT*) e registradores. As conexões locais transferem os sinais entre os *LEs* na mesma *LAB*. Já as conexões *LUT* transferem a saída de um *LE* para um *LE* adjacente. A Figura 2.3 mostra a estrutura de uma *LAB*.

Na arquitetura *Cyclone*, as conexões entre *LE*, blocos de memória *M4K* e pinos de *E/S* são feitas com o uso de interconexões do tipo *MultiTrack*. Esses tipos de interconexões consistem de conjuntos de canais rápidos de interconexão dispostos na forma de linhas e colunas. Qualquer *LE*, *IOE* ou *ESB* pode ser comandado por outro *LE*, *E/S* ou blocos de memória *M4K* através destas interconexões em linhas e colunas.

O elemento lógico (*LE*) é a menor unidade de lógica presente numa arquitetura *Cyclone*, Figura 2.4. Cada *LE* contém uma *LUT* de quatro entradas, que pode implementar funções de até quatro variáveis. Além disso, cada *LE* contém um registrador programável, com conexões de alta velocidade (*carry e cascade chains*).

Cada registrador programável do *LE* pode ser configurado para operar como um *flip-*

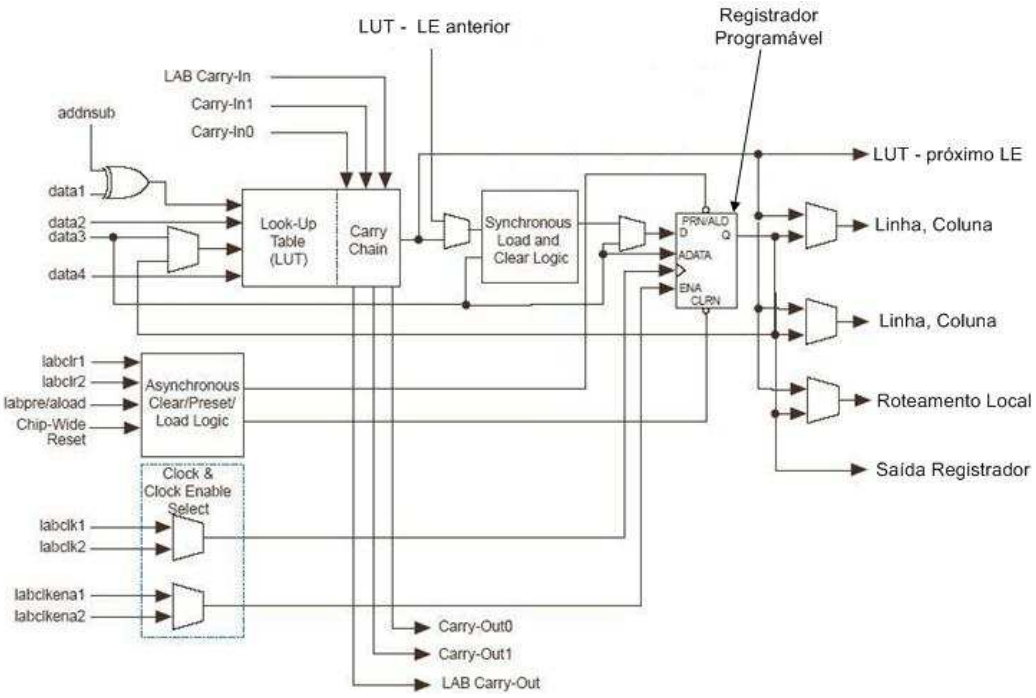

Figura 2.4: Estrutura de um LE.

flop tipo: D, T, JK, ou SR, sendo que seu relógio e seu sinal de controle *clear* podem ser comandados por sinais globais, por pinos de entrada e saída (E/S) de propósito geral, ou por qualquer outro tipo de lógica interna existente no componente. Para funções combinacionais, o registrador é ignorado e a saída da LUT comanda as saídas do LE. Qualquer pino de E/S pode ser roteado para assumir as funções de *reset*, *preset*, *clock enable* e para uso geral.

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

2.4 NIOS

Os processadores NIOS formam uma família de processadores embutidos que possuem um núcleo *RISC* de propósito geral com as seguintes características:

- Conjunto de instruções, barramento de dados e espaço de endereço de 32 *bits;*

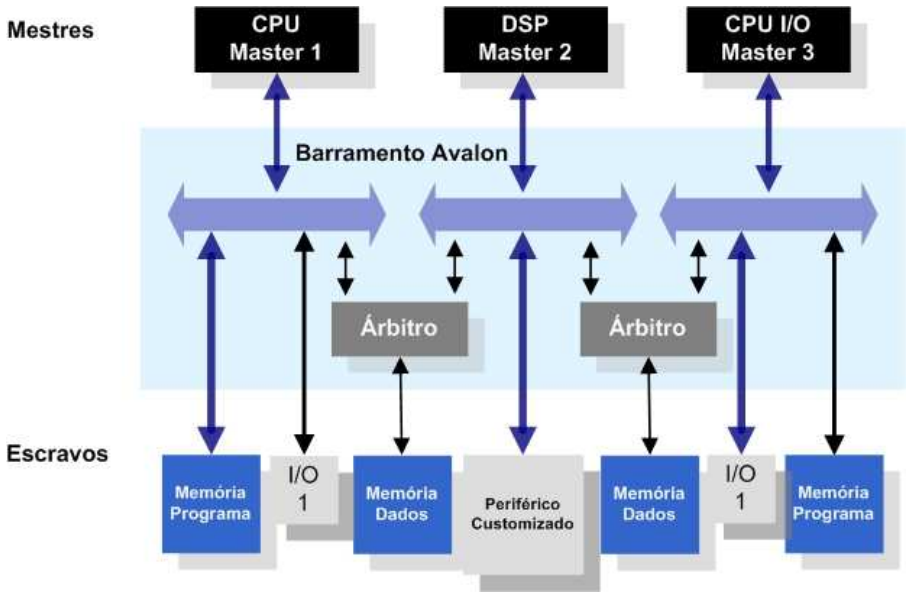

Figura 2.5: Exemplo de sistema baseado no processador NIOS.

- 32 registradores de propósito geral;
- 32 fontes de interrupção externas;
- Multiplicação e divisão de 32 *bits* em instruções únicas;
- Instruções dedicadas para cálculo de multiplicação que produzem resultados de 64 a 128 *bits*;
- Deslocamento em uma única instrução;
- Acesso a uma gama de periféricos *on-chip* e interfaces para memórias e periféricos *off-chips*;
- Performance de 100 *DMIPS*.

O sistema NIOS é equivalente a um microcontrolador, ou seja, inclui uma *CPU* (*Central Processing Unit - Unidade Central de Processamento*) e mais uma combinação de periféricos e memória em um único *chip*. A Figura 2.5 apresenta um sistema básico baseado no processador NIOS.

O NIOS é um processador configurável *soft-core*. Neste contexto, configurável refere-se ao fato das características poderem ser adicionadas ou removidas de um sistema afim de alcançar o desempenho e/ou custo desejados. *Soft-core* refere-se ao fato da *CPU* apresentar-se na forma de um projeto *soft* (i.e, não fixo em silício), de modo a permitir que o mesmo seja implementado em diversas famílias de *PLD* da Altera. Ou seja, o fabricante Altera não

comercializa *chips* NIOS, e sim *PLD*. É o usuário que deverá configurar o processador NIOS juntamente com os periféricos que se ajustem as suas especificações, e em seguida programar o sistema dentro de um *PLD*.

A alta configurabilidade não significa que o projetista deva criar uma nova configuração do processador para cada aplicação, ou seja, o fabricante disponibiliza projetos que podem ser utilizados. O conjunto de periféricos flexíveis é uma das características mais notáveis de sistemas desse tipo, pois o projeto pode ser feito sob medida para as exigências da aplicação. Já o mapa de endereços flexível torna genérico o acesso a memória de periféricos. Os periféricos podem ser classificados em duas grandes classe: periféricos padrão e personalizados.

Os periféricos padrão são fornecidos pela Altera que são comumente usados em microcontroladores tais como, *timers*, interface de comunicação serial, E/S de propósito geral, controladores de memória etc. Os periféricos personalizados são criados pelo próprio usuário e integrados ao sistema. Para sistemas com desempenho crítico que gastam muitos ciclos de relógio de *CPU* executando um seção de código específica, uma técnica comum é criar um periférico personalizado que implemente a mesma função em *hardware*. Essa abordagem oferece um duplo benefício de desempenho: a implementação em *hardware* é mais rápida que em *software*; e o processador fica livre para executar outras funções em paralelo enquanto o periférico customizado processa seus dados.

Da mesma forma que para periféricos, pode-se criar instruções personalizadas para aumentar o desempenho do sistema. A natureza *soft-core* do processador NIOS permite ao projetista integrar lógica personalizada na unidade lógica aritmética (ULA). Da mesma forma que as instruções nativas, a nova instrução lógica pode utilizar como entrada até dois valores de registradores e, opcionalmente, escrever o resultado em um registrador de destino.

Através da utilização de instruções personalizadas o projetista pode fazer um ajuste fino no *hardware* do sistema para maximizar seu desempenho. Como o processador é implementado em lógica programável, engenheiros de *software* e *hardware* podem trabalhar de modo integrado para otimizar interativamente o *hardware* e testar os resultados de execução do *software* em *hardware* real (implementação em um *PLD*).

Sob a perspectiva de *software*, instruções personalizadas aparecem como macros em *assembly* geradas pela máquina ou como funções C, o que elimina a necessidade do programador conhecer *assembly* para usar as instruções personalizadas.

Com o ambiente de desenvolvimento *QuartusII* (QuartusII, 2005) e o SOPC *Builder* (SOPC, 2005) o processador NIOS pode ser programado para dentro do *PLD*.

2.5 Ambiente de Desenvolvimento

O projeto de sistemas digitais utilizando a família de *PLD* Cyclone e também de outras famílias da *Altera*, é realizado com auxílio de um ambiente de desenvolvimento chamado *QUARTUSII*, Figura 2.6. O processo de projeto usando o programa *QUARTUSII* passa por quatro fases: entrada, compilação, verificação e programação (configuração).

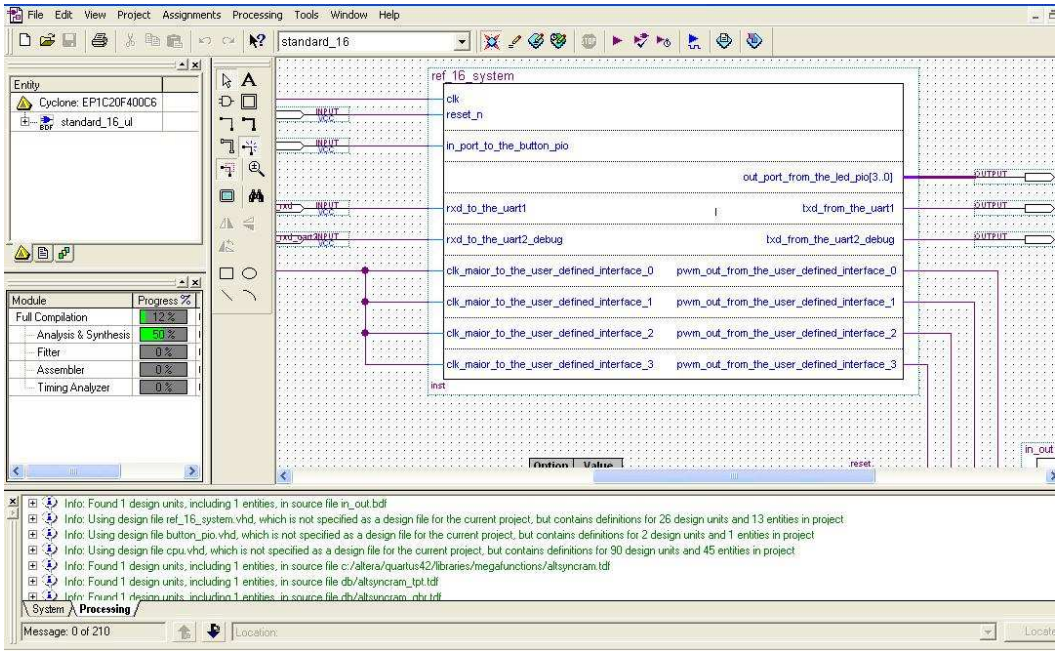

Figura 2.6: Interface gráfica do programa *QUARTUS II*.

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

O *QUARTUSII* também possui um editor de textos para a entrada de arquivos de projeto escritos em *HDL* (*VHDL*, *Verilog*, *Assembly*, *LinguagemC* ou *AHDL*), sendo que seu compilador sintetiza lógica a partir de qualquer uma dessas *HDL* e, também, mapear esta lógica para algumas famílias de *PLD*. Ele também permite que funções *LPM* (funções parametrizáveis disponíveis na biblioteca de componentes da Altera - Quartus II) sejam instanciadas em *HDL*

e seu editor de *floorplan* simplifica o processo de associação de lógica a pinos e células lógicas do componente. O projetista pode associar pinos e células lógicas antes de compilar um projeto e também verificar e modificar os resultados pós compilação.

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

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

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

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

O *QUARTUSII* também possui um analisador lógico denominado *Signal Tap*. Com ele é possível visualizar qualquer sinal (ponto) do circuito projetado, fazendo com que a depuração do projeto possa ser feita de forma simples e rápida. Para sua utilização é necessário que o sistema que contenha o *PLD* tenha a possiblidade de configuração via interface *JTAG* (*Joint Test Action Group*).

2.6 *xPC Target*

O *toolbox xPC Target* é uma ferramenta importante no projeto, teste e desenvolvimento de sistemas em tempo real que façam uso do *hardware* padrão de computadores pessoais, os conhecidos *Personal Computers* (*PC*). O *xPC Target* é um ambiente de desenvolvimento que utiliza dois *PC* distintos sendo o primeiro denominado *host* e o segundo, *target* e, possibilitando assim a realização de aplicações em tempo real.

O princípio de funcionamento do *xPC Target* consiste em utilizar um *desktop* comum, funcionando como o computador *host*, com o *software* MATLAB e o seu pacote SIMULINK previamente instalados, permitindo a criação de modelos de blocos para a aplicação em estudo. Criado o modelo, é possível simulá-lo, em tempo real, no computador *target*.

O *xPC Target* permite que o usuário adicione blocos de Entrada/Saída (E/S) ao modelo desenvolvido e utilize o *PC host* com outro *toolbox*, o *Real-Time Workshop*[1], além de um compilador $C/C++$ pré-instalado para criar o código executável. Este código, gerado no *host PC*, é carregado para o *target PC* onde já está sendo executado o *xPC Target real-time kernel*. Após o carregamento do código executável, pode-se acionar e testar a aplicação desenvolvida em um ambiente de tempo real.

Os principais requisitos e características do *xPC Target* podem ser listadas da seguinte maneira (xPC Target, 2005):

Requisitos de *hardware*: o *toolbox xPC Target*, como comentado, necessita de um *host PC*, um *target PC* e, para a aquisição de dados, o *target PC* necessita possuir placas de E/S. Tipicamente, o *target PC* pode ser um *desktop* ou um computador industrial;

Requisitos de *software*: é necessário um compilador $C/C++$ pré-instalado (Borland $C++$ Builder, Borland $C/C++$, WATCOM $C/C++$, Microsoft Visual $C/C++$[2]) além do *toolbox Real-Time Workshop*;

Documentação e manuais de ajuda: em (Mathworks, 2005), encontra-se uma grande quantidade de documentação relacionada à este *toolbox* além de muitos outros que compõem o *software* MATLAB.

De uma maneira bem simplificada, o *xPC Target* pode ser estudado a partir do esquema apresentado na Figura 2.7. Como pode ser observado, a comunicação entre os dois computadores é realizada através da entrada serial de ambos (*RS-232*), a mesma metodologia escolhida neste trabalho.

[1]Uma grande quantidade de documentação sobre este *toolbox* pode ser encontrado em http://www.mathworks.com/access/helpdesk/help/toolbox/rtw/.

[2]Neste trabalho, utilizou-se a versão 6.0 deste compilador.

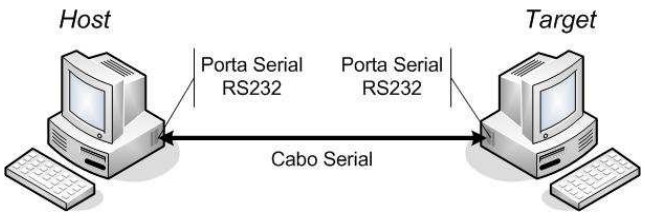

Figura 2.7: Esquema de comunicação entre os computadores *host* e *target* via porta serial.

Objetivando ilustrar as possibilidades envolvidas, outra maneira possível de estabelecer a comunicação entre os computadores *host* e *target* dar-se-á através do protocolo de comunicação *TCP/IP* (Figura 2.8).

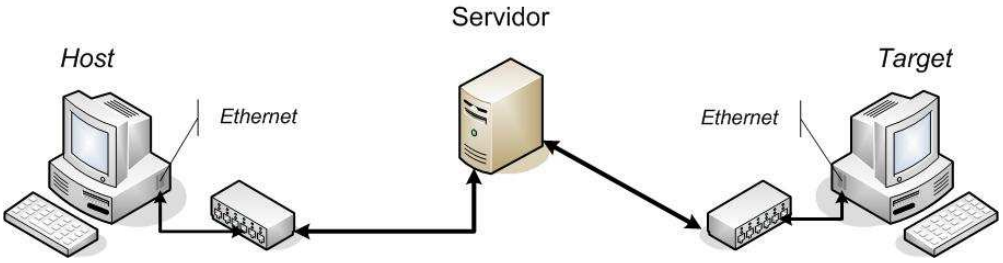

Figura 2.8: Esquema de comunicação entre os computadores *host* e *target* via protocolo de comunicação *TCP/IP*.

Para validar o correto funcionamento do sistema de comunicação, utilizou-se o exemplo, dentre os vários disponibilizados pelo MATLAB, denominado `xpctank` cujo diagrama de blocos é apresentado na Figura 2.9.

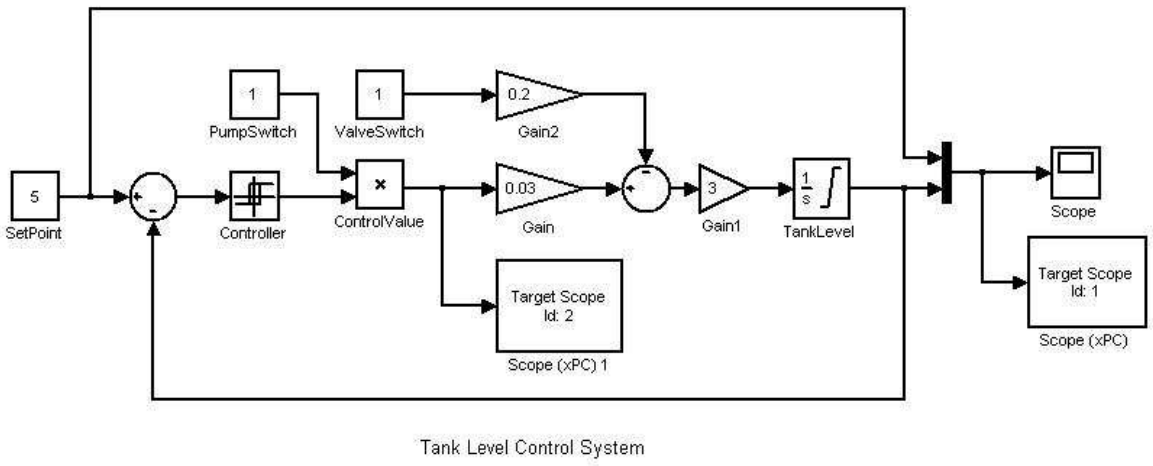

Figura 2.9: Diagrama de blocos do exemplo utilizado.

Na Figura 2.10 tem-se o resultado da simulação desse sistema na tela do *target PC*. O computador *target* utilizado para esta simulação possui um processador Intel® Celeron 400MHz com 128M*bytes* de memória *RAM* instalados. Este microcomputador foi conectado, via porta serial *RS-232*, ao computador *host* cujo processador instalado é um AMD Athlon^{TM} XP 1800+, com 256M*bytes* de memória *RAM*.

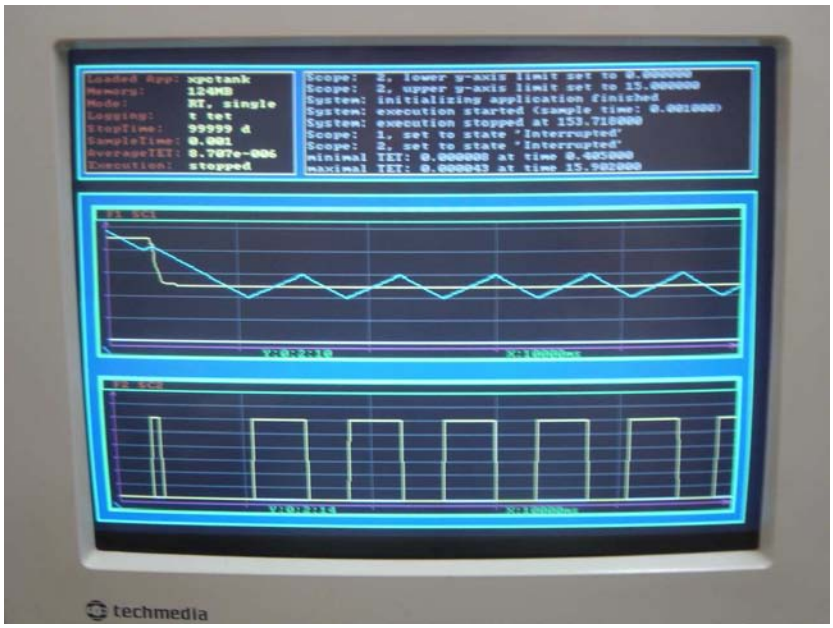

Figura 2.10: Resposta do sistema exemplo apresentado.

2.7 Placa de Circuito Impresso (PCI)

As placas de circuito impresso (PCI) são partes integrantes na maioria dos sistemas eletrônicos atuais e são encontradas na maioria dos produtos eletrônicos, aplicações militares, equipamentos médicos etc. As PCI são consideradas como elemento chave dos equipamentos eletrônicos. A rápida evolução tecnológica tem resultado numa queda dos custos e melhora dos equipamentos fazendo com que as PCI sejam fabricadas mais rapidamente e que tenham uma maior durabilidade. De acordo com estimativas, as PCI constituem em torno de 3% do uso total de um equipamento eletrônico (Li *et al.*, 2004).

Há 3 tipos de PCI: face simples, dupla face e multi-camadas. Elas podem ser rígidas, flexíveis ou ambos. São compostas das seguintes partes: 1) substrato não-condutor; 2) substrato condutor; 3) componentes montados. Dos diversos tipos de substratos usados na fabricação de PCI, os *FR4* (*Flame Retardant*) são os mais utilizados. Um outro tipo bastante utilizado também é o *FR2*. Outros tipos de substratos podem ser encontrados em (Li *et al.*, 2004).

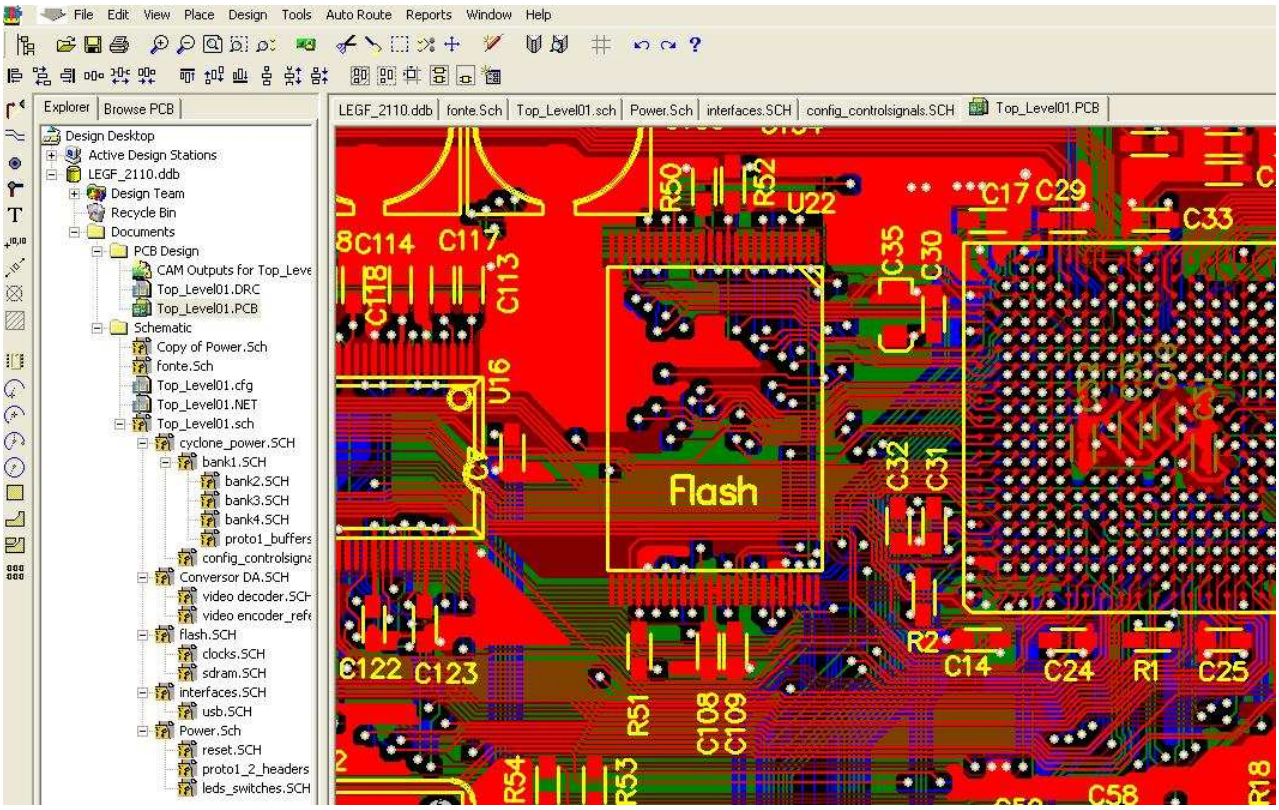

Figura 2.11: Interface gráfica do *software Protel*.

Os componentes geralmente montados nas PCI incluem *chips*, conectores, resistores, capacitores etc. Diferentes componentes, possuem em sua composição materiais diferentes, por exemplo, *Tantalum* é usado em capacitores. As ilhas de solda contém An, Pb e Cd. Semicondutores contém Ga, Si, Se, Ge, e outros elementos em menores quantidades. A composição das PCI é, portanto, bastante complexo e causa grandes problemas para indústria de reciclagem.

A placa de desenvolvimento proposta nesse trabalho foi fabricada utilizando *FR4*. É uma PCI contendo 6 camadas (multi-camadas). Possui tanto plano de terra como de alimentação positiva. Todo o desenvolvimento (diagrama elétrico e roteamento) foi realizado utilizando o *software Protel*, ver Figura 2.11.

Capítulo 3

A Placa de Desenvolvimento

Neste capítulo apresenta-se o sistema desenvolvido com a descrição detalhada de cada bloco.

3.1 *Hardware*

Um modelo geral de arquitetura de *hardware* para sistemas de controle de robôs industriais é mostrado na Figura 3.1 (Sciavicco e Siciliano, 1996). Outros sistemas de controle podem ser encontrados em (Arribas e Macia, 2002), (Roggen *et al.*, 2003), (Fischman e Le, 2004). Em todos os sistemas, a estrutura básica é semelhante a mostrada na Figura 3.1. As diferenças, ou variantes, encontradas estão relacionadas com cada aplicação envolvida.

O sistema proposto nesse trabalho é mostrado na Figura 3.2. Nessa proposta foram considerados os aspectos abordados no Capítulo 1 como por exemplo, necessidade de alto capacidade computacional, paralelismo inerente nos problemas a serem resolvidos, tempo de execução e, buscando utilizar a melhor tecnologia disponível atualmente. Todos os periféricos foram ligados diretamente ao *PLD* para que se tenha um nível de paralelismo elevado (acesso simultâneo a periféricos) para determinada aplicação.

As etapas de desenvolvimento da placa de desenvolvimento foram:

- Definição das características que o sistema de controle deve possuir e requerimentos de capacidade de processamento;
- Estudo e escolha dos componentes a serem utilizados;
- Projeto dos subsistemas englobados no projeto;
- Desenvolvimento dos esquemáticos utilizando o *software Protel*;

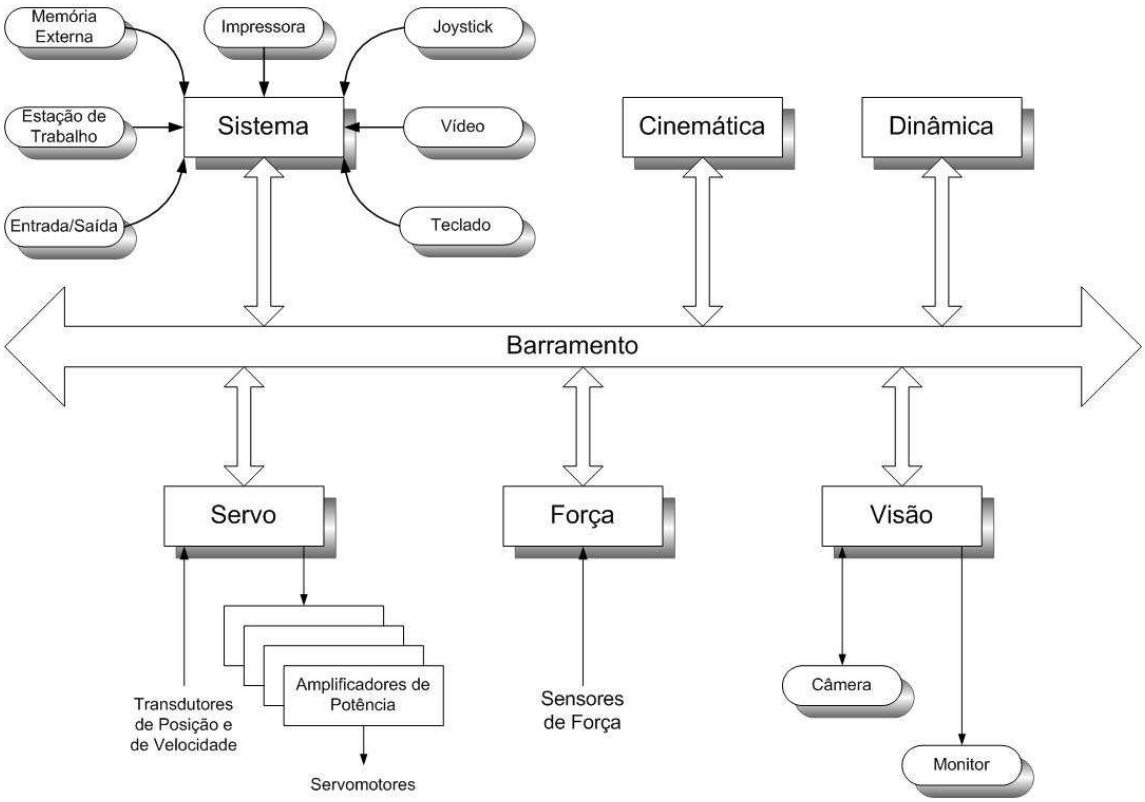

Figura 3.1: Estrutura proposta por Siciliano *et al.* (1996).

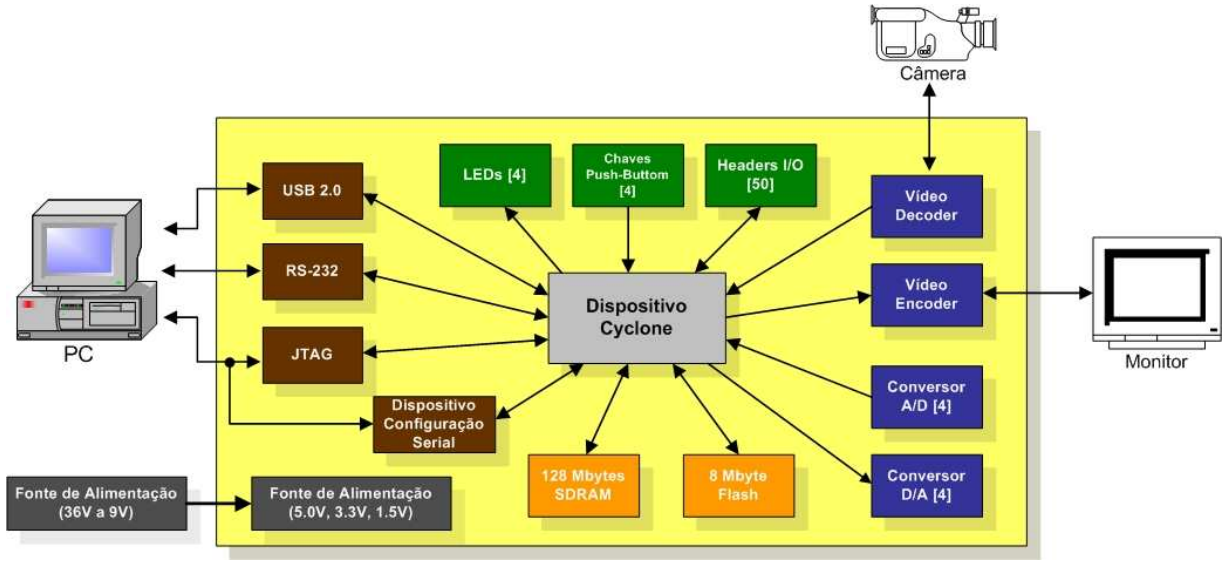

Figura 3.2: Diagrama de blocos proposto.

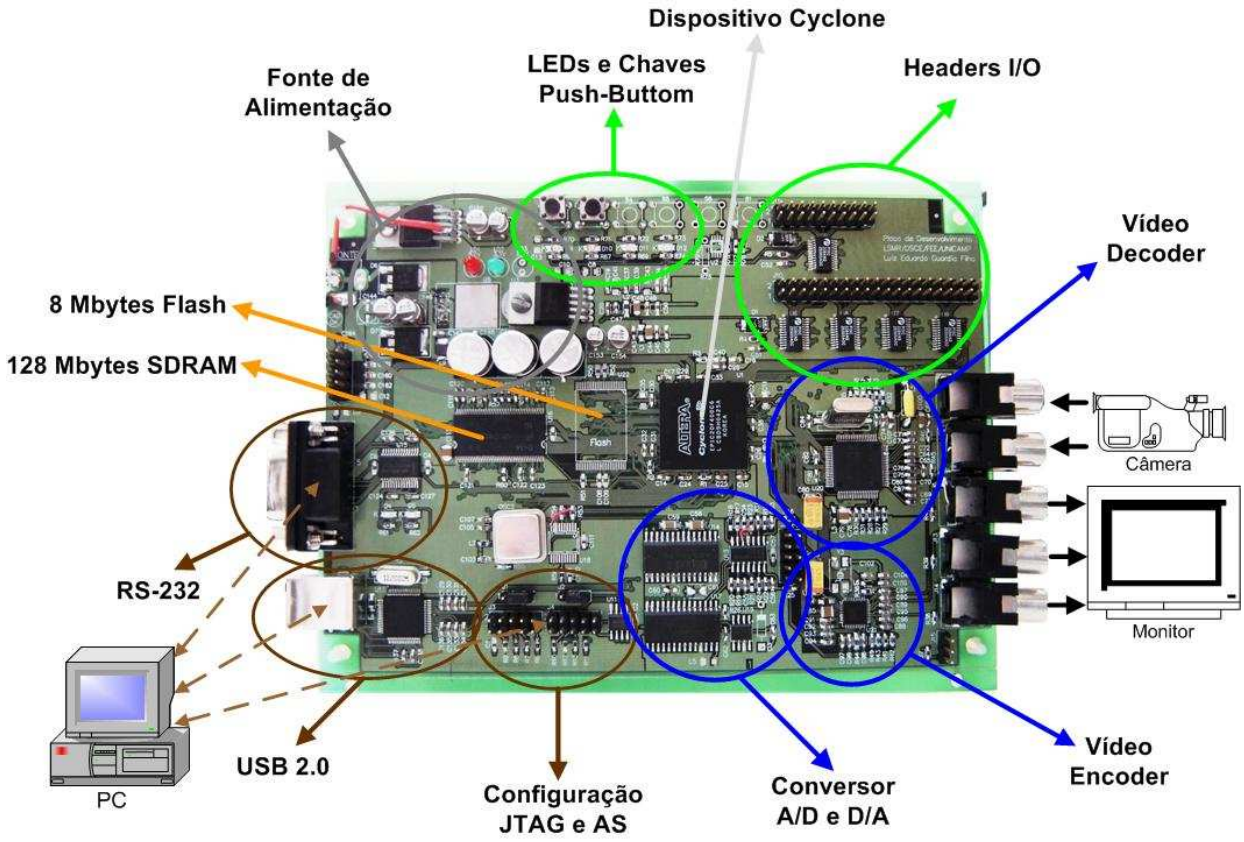

Figura 3.3: Foto da placa de desenvolvimento.

- Roteamento da placa utilizando o *software Protel* (terceirizado);
- Fabricação da Placa de Circuito Impresso e montagem dos componentes (terceirizado);
- Geração da estratégia para a realização dos testes de funcionamento;
- Testes funcionais na placa de desenvolvimento.

A foto da placa bem como a indicação de todos os componentes é mostrada na Figura 3.3. A explicação de cada bloco mostrada na Figura 3.3 é apresentada a seguir.

3.1.1 Alimentação

A alimentação da placa foi projetada para oferecer as seguintes características:

- Tamanho reduzido;
- Pouca dissipação de calor para a placa;

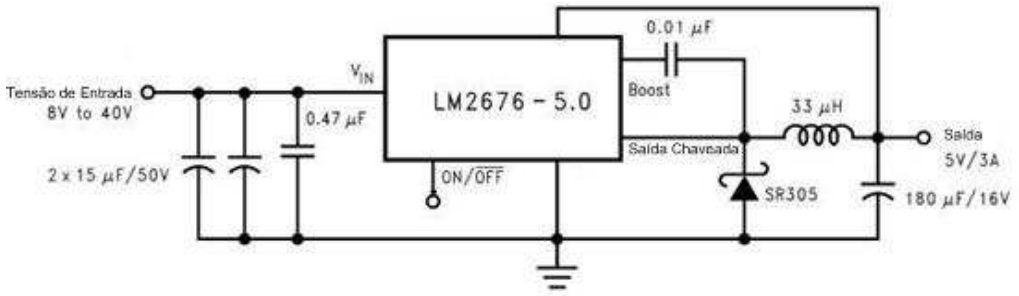

Figura 3.4: Regulador 5V implementado.

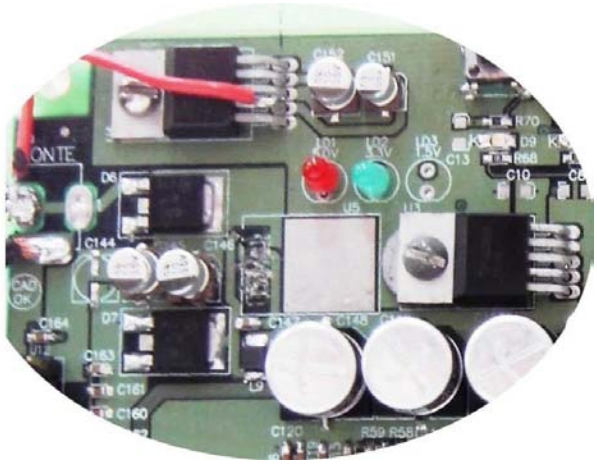

Figura 3.5: Circuito de alimentação da placa.

- Regulação interna de todas tensões necessárias aos componentes;
- Possibilidade de alimentar sensores que são ligados a ela, como por exemplo, *encoders* e chaves fim de curso.

A placa pode ser alimentada com tensão contínua entre 9 e 36 *Volts*. A regulação de entrada, ver Figura 3.4, gera uma tensão de saída estável de 5 *Volts* que é realizado por um regulador *Buck*[1]. Essa configuração, ao contrário de reguladores lineares, faz com que o componente não dissipe muito calor para a placa, dispensando assim o uso de dissipadores.

Os dois reguladores secundários, alimentados pelo principal, são lineares e geram nas suas saídas 3 e 1,5 *Volts*. A potência total da fonte é de 25W, sendo que o consumo para alimentação de todos os componentes é de 6W (sem considerar alimentação de sensores externos). A placa foi projetada também para suportar a alimentação, por exemplo, dos sensores do robô (chaves fim de curso, *encoders* etc). A Figura 3.5 mostra o circuito de alimentação da placa.

[1]Conversor chaveado utilizado para abaixar tensão.

3.1.2 Interfaces de Comunicação

As interfaces de comunicação são responsáveis pela comunicação entre a placa de desenvolvimento e um PC, por exemplo. São através delas que os dados são enviados e recebidos. Foram utilizadas uma interface *USB* e uma *RS-232*.

A interface *USB* utiliza o controlador ISP1581 da *Philips* o qual trabalha a uma freqüência de transferência de dados de 12.5MHz (*USB Revision* 2.0). Essa interface é responsável principalmente para transferência de imagens, tanto capturadas por uma câmera como processadas no *PLD*.

A interface *RS-232* é responsável principalmente para comunicação entre um PC e o processador NIOS (*PLD*). A Figura 3.6 mostra uma foto de cada interface na placa.

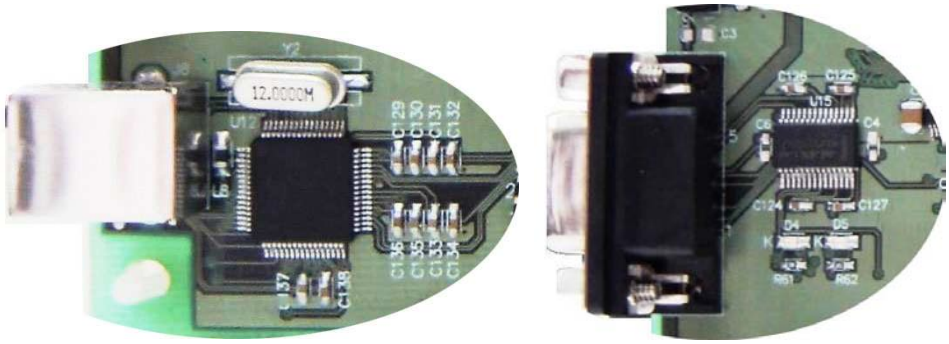

Figura 3.6: Interfaces de comunicação utilizadas na placa. À esquerda, é mostrada a interface *USB*.

3.1.3 Memórias

Foram implementadas dois tipos de memórias: *SDRAM* e *FLASH*. A *SDRAM*, possui 128*Mbytes* de capacidade. Ela é utilizada para auxiliar o *PLD* no processamento de algoritmos. É através dela que armazena-se, por exemplo, blocos de imagens digitais para poder realizar todo o processamento necessário. Por exemplo, utilizando-se uma imagem (matrizes) 640×480 (padrão) no formato YCrCb (tipo de formato disponível no *chip decoder* implementado na placa) 4:2:2, significa que, numa linha, a cada 4 *pixels*[2] tem-se 4 informações (*bytes*) de níveis de cinza (Y)e 4 informações (*bytes*) de cor (2Cb + 2Cr). Assim, como a imagem tem 640 *pixels* em cada linha, tem-se $640 \times (8/4)$ *bytes* por linha. Como se tem 480 linhas, cada imagem ocupa aproximadamente 614,4 *Kbytes* de memória. Todo o controle e acesso da memória foi realizado utilizando-se o processador NIOS.

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

Já a memória FLASH tem o propósito de armazenar programas a serem executados pelo *PLD*. Essa memória não foi montada ainda por dificuldades em adquirir o componente. A Figura 3.7 mostra as memórias utilizadas na placa de desenvolvimento.

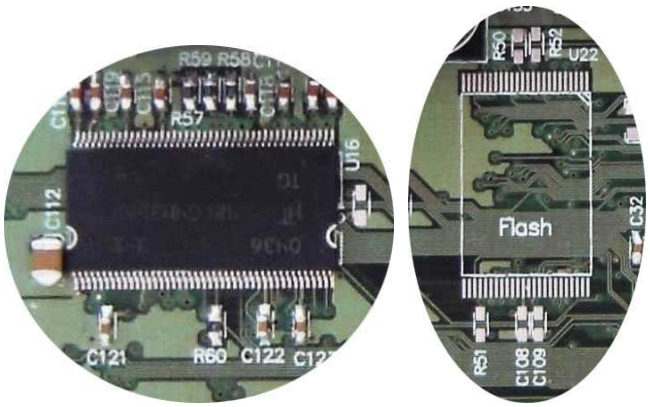

Figura 3.7: Memórias utilizados na placa de desenvolvimento. À esquerda, é mostrada a memória *SDRAM*.

3.1.4 Sistema de Visão

Um sistema de processamento de imagens digitais e visão computacional é uma coleção de componentes de *hardware* e *software* que fazem a aquisição, armazenamento, processamento (melhoramento, extração de características, reconhecimento de padrões etc) e a exibição das imagens digitais, conforme mostrado na Figura 3.8. Embora esses componentes possam ser fisicamente separados, cada um é fundamentalmente necessário para completar o ciclo de processamento de imagens digitais.

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

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

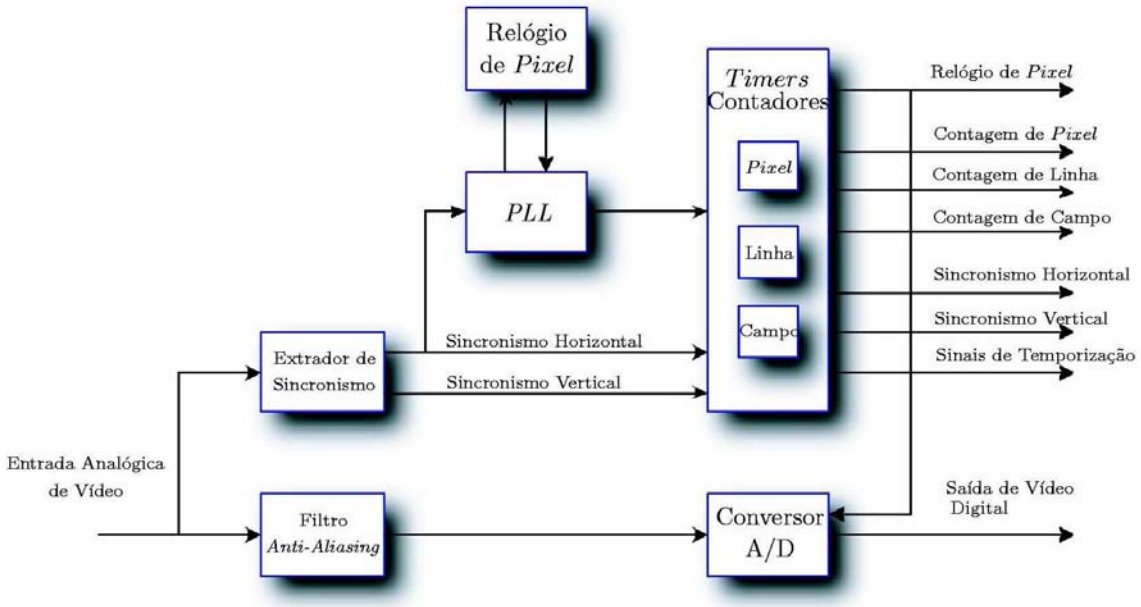

Figura 3.8: Componentes básicos de um sistema de processamento das imagens.

são lidos dela e enviados para dispositivos de exibição, tais como impressoras e monitores de vídeo. Em alguns casos é necessário que os dados digitais sejam convertidos novamente para a forma analógica por meio de um conversor digital-analógico D/A, para então serem exibidos.

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

A placa desenvolvida nesse trabalho possui 2 entradas para sinais de vídeo analógico e 3 saídas para sinais também analógico. O *chip* (*decoder* - TVP5145) responsável pela aquisição do sinal de vídeo realiza internamente todas as funções (blocos) mostradas na Figura 3.8, onde o padrão de entrada utilizado é o *ITU 656* (656, 2005). Neste formato as informações de vídeo são transmitidas a 25 *MWords/s* na seguinte seqüência: Cb, Y, Cr, Y, Cb, Y, Cr etc, onde o Y

se refere a luminância (níveis de cinza) e o CbCr a crominância (cor) (Gonzalez e Woods, 2000). Todo o processamento da imagem é realizado pelo *PLD*.

Já o *chip* que faz a codificação do sinal de vídeo (ADV7179) pode ser utilizado, por exemplo, para visualização do resultado parcial ou final do processamento do sinal.

Esse bloco da placa é utilizado, por exemplo, quando realiza-se controle por visão computacional ou ainda para contemplar-se o nível hierárquico mais alto, o da supervisão. A Figura 3.9 mostra a foto do *encoder* e *decoder* utilizados na placa.

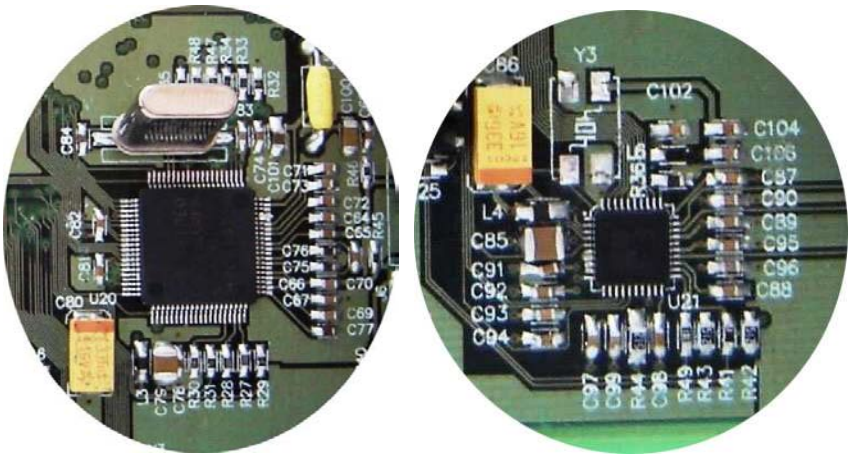

Figura 3.9: Sistema para processamento de imagens utilizado. À esquerda, é mostrada o *chip decoder*.

3.1.5 Conversores A/D e D/A

Funções geradas por blocos funcionais analógicos são muitas vezes processadas por circuitos digitais (por exemplo, um computador). Para processar este sinal usando circuitos digitais, deve-se necessariamente efetuar uma conversão para a forma digital. Tal conversão é efetuada por um conversor analógico/digital (A/D *converter* ou *ADC*). Este sinal processado (ou transformado) deve (na maioria das vezes) atuar, produzindo um efeito sobre o circuito analógico que gerou o sinal original, ou outro similar.

Um sinal na forma digital, para ser processado por um bloco funcional analógico, deve ser previamente convertido (ou reconvertido) para a forma analógica equivalente. Este processo reverso é efetuado por um conversor digital/analógico (D/A *converter* ou *DAC*), ver Figura 3.10.

Os conversores utilizados foram: A/D - ADS7825 e D/A - DAC7625, ambos da Texas Instruments. Um exemplo de aplicação de conversores A/D em robótica seria para aquisição de sinais de sensores analógicos, como os strain gauges (utilizados em garras). As características

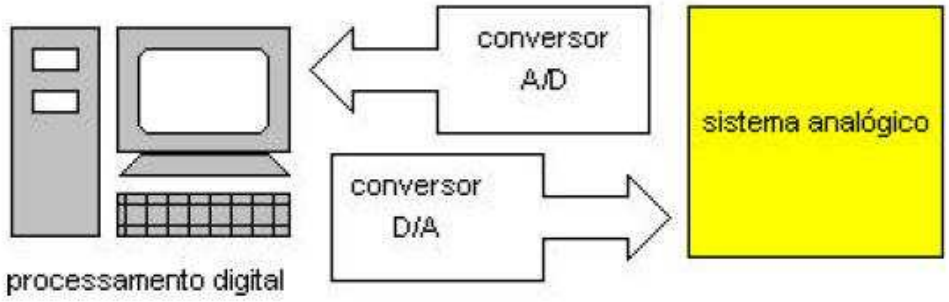

Figura 3.10: Sistema de conversão A/D e D/A.

de cada conversor são mostradas a seguir. A Figura 3.11 mostra a foto dos conversores na placa.

- Conversor A/D
  - Possui 4 entradas de 16 *bits* que são multiplexadas e processadas individualmente;
  - Utilizou-se alimentação simples de +5V;
  - Tempo de amostragem e conversão de 25μs [3];
- Conversor D/A
  - Possui 12 *bits* de resolução;
  - Entrada do dado digital na forma paralela;
  - Possui 4 saídas analógicas;
  - Opera com tensão simples de +5V ;

3.1.6 Pinos de E/S, *Leds* e chaves

Os pinos de E/S disponibilizados na placa de desenvolvimento podem ser utilizados para inserção/retirada de qualquer sinal digital ligado ao *PLD*. No caso de uso da placa em robótica, esses pinos são utilizados como aquisição dos sinais dos *encoders*, chaves fim de curso, comunicação com placas de aquisição etc. Foram disponibilizados 50 pinos de E/S (Figura 3.12).

As chaves e os *leds*, por se tratar de uma placa de desenvolvimento, podem ser utilizados para testes de circuitos implementados no *PLD*. A placa possui 4 chaves e 4 *leds* de propósito geral (Figura 3.12). Foram montadas apenas 2 chaves para testes.

[3]Devido às constantes de tempo envolvidas nos plantas robóticas acionadas eletricamente, geralmente os tempos exigidos para esta finalidade são da ordem das centenas de milissegundos.

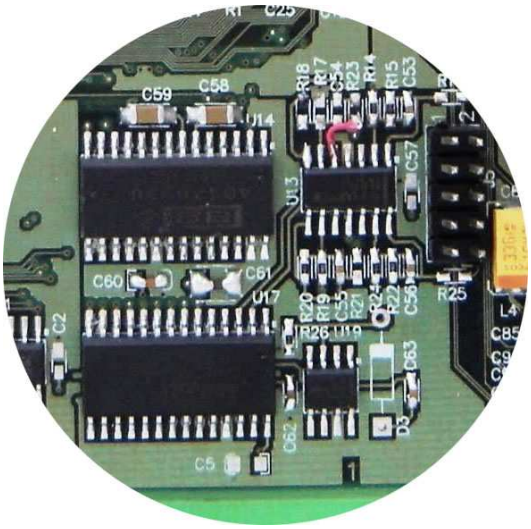

Figura 3.11: Conversores implementados na placa (conversor D/A mais acima).

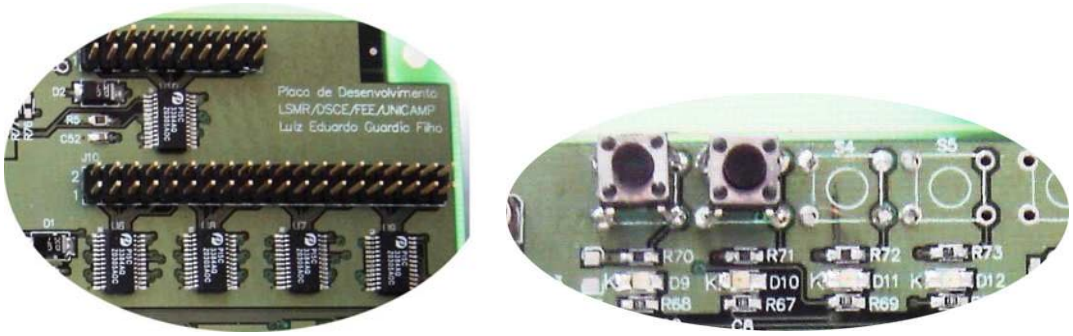

Figura 3.12: Pinos E/S, *leds* e chaves disponíveis na placa.

3.1.7 Configuração

Foram implementados dois modos de configuração: *JTAG* (*Join Test Action Group*) e *AS* (*Active Serial*). As características de cada configuração é mostrada a seguir. A Figura 3.13 mostra os dois modos de configuração utilizados.

- *JTAG*
  - Não necessita de nenhum componente externo para seu funcionamento;
  - Pode ser utilizado em qualquer *PLD* da família Altera;
  - Forma mais simples de configuração;
  - Através dessa interface é possível utilizar ferramentas de depuração de projetos da Altera, como por exemplo, o analisador lógico denominado de *Signal Tap*.

- *AS*
  - Utilizado apenas na família *Cyclone*;
  - Necessita de uma memória serial externa para seu funcionamento;
  - Foi utilizado o dispositivo EPCS4 a qual possui uma capacidade de armazenamento de 4.194.304 *bits*;
  - Durante a configuração o dispositivo cyclone é o mestre e a memória o escravo. Toda vez que a placa é energizada, o *PLD* se configura através dessa memória, não necessitando ser configurada a todo momento.

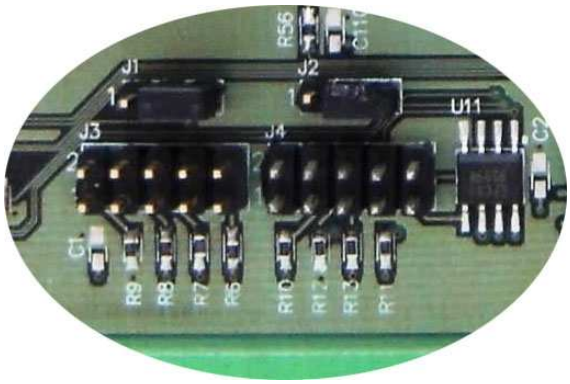

Figura 3.13: Modos de configuração utilizados (configuração *JTAG* à esquerda).

3.1.8 Dispositivo *Cyclone*

O *PLD* utilizado nesse trabalho foi o *EP1C20F400C6* - Figura 3.14, da família *Cyclone* da Altera. Abaixo segue algumas características desse dispositivo.

- Possui 20.060 elementos lógicos, 294.912 *bits* de memória *RAM*, 2 *PLL* e 301 pinos de E/S;
- Velocidade de comunicação entre o *PLD* e periférico: até 640Mbps;
- Arquitetura Multicore: constituído por conjunto de blocos lógicos;
- Encapsulamento *BGA* com 400 pinos.

O encapsulamento desse dispositivo permite a montagem de um dispositivo de menor capacidade e menor custo, EP1C4, sem que seja necessária a alteração do layout da placa. As características desse dispositivo são mostradas a seguir.

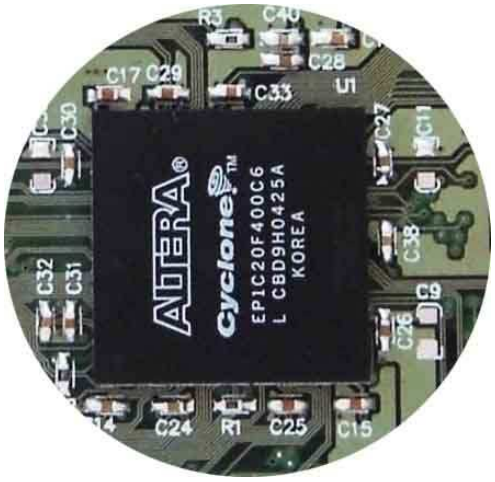

Figura 3.14: *PLD* utilizado nesse trabalho.

- Possui 4.000 elementos lógicos, 78.336 *bits* de memória *RAM*, 2 *PLL* e 301 pinos de E/S;
- Velocidade de comunicação entre o *PLD* e periférico: até 640Mbps;
- Arquitetura Multicore: constituído por conjunto de blocos lógicos;
- Encapsulamento *BGA* com 400 pinos.

Capítulo 4

Aplicações e testes da placa de desenvolvimento

Este capítulo apresenta algumas das implementações realizadas na placa de desenvolvimento para depuração do sistema proposto.

4.1 Implementação de Funções Transcendentais em *Hardware*

Na modelagem matemática e principalmente nas equações de movimento de sistemas robóticos que servem como paradigmas para aplicações das propostas geradas durante o desenvolvimento deste trabalho de mestrado, existe a presença constante de funções transcendentais. Isso se deve ao fato de que tais modelos são baseados na geometria de estruturas mecânicas que possuem movimentos rotacionais e translacionais entre suas partes móveis. É facilmente constatado também que proporcionalmente ao número de graus de liberdade que uma estrutura mecânica tenha, a quantidade dessas funções transcendentais aumenta, e também aumenta o número de operações básicas entre os termos matemáticos envolvidos como adição e multiplicação.

Como o objetivo é conseguir resolver estas equações complexas em tempos muito reduzidos, e sabendo-se que o cálculo de funções transcendentais demanda grande parte da computação geral dos sistemas de equações, procurou-se por uma estratégia de conceber e resolver estas equações em *hardware*.

Não somente no equacionamento dos movimentos estruturais de robôs aparecem termos contendo funções transcendentais, senão também muitas vezes em algoritmos de controle para tais sistemas, principalmente quando utiliza-se estratégias e técnicas numéricas baseadas em

inteligência artificial como redes neurais, heurísticas, lógica nebulosa e computação evolutiva.

4.1.1 Cálculo de Funções Transcendentais

No contexto das equações cinemáticas e dinâmicas de sistemas robóticos, o cálculo das funções transcendentais seno, cosseno e arco-tangente é mais freqüentemente necessário. Assim estão apresentadas na seqüência as propostas para a respectiva realização em *hardware*, sendo que em caso de necessidade de resolução de outras funções transcendentais o método proposto pode ser o mesmo (Guardia Filho *et al.*, 2004).

Para as implementações das funções propostas: seno, cosseno e arco-tangente, foram realizadas a expansões de cada função em série de potência utilizando a Série de *Taylor* através do *software Matlab*. Para cada uma dessas funções também foi realizado um ajuste não-linear nos coeficientes da respectiva série com o método dos mínimos quadrados (Aguirre, 2000). Através do conjunto de blocos de ponto fixo do *Matlab/Simulink* foi simulada a expansão de cada série determinando-se assim o número de *bits* a ser utilizado nas conversões A/D e D/A, e também o erro que cada aproximação gera. O circuito utilizado para determinar o número de *bits* é mostrado na Figura 4.1.

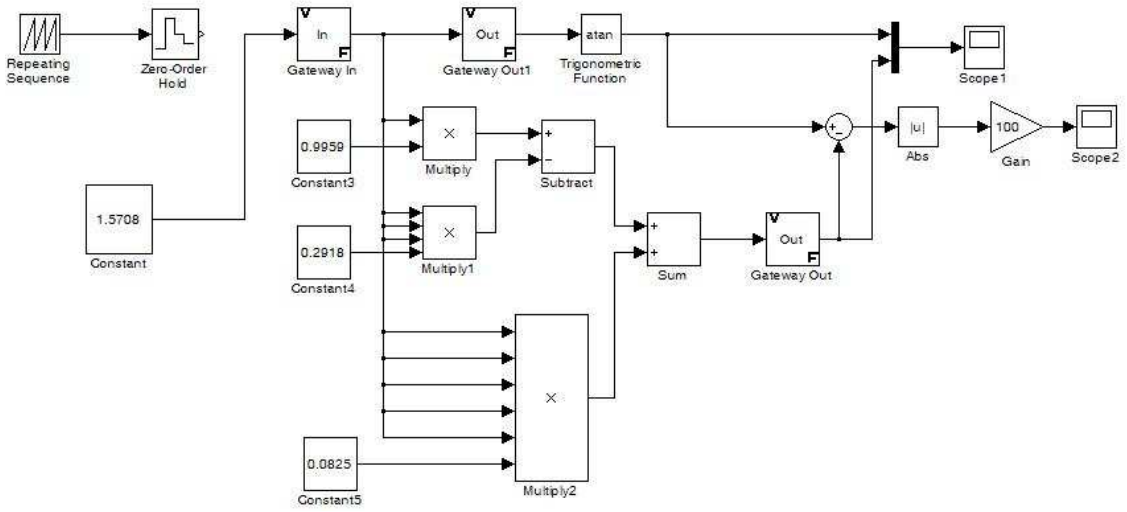

Figura 4.1: Circuito utilizado para determinação do número de *bits*.

Com o ajuste não-linear realizado em cada função foi possível minimizar os erros devido a cada aproximação realizada pela respectiva expansão em Série de *Taylor*, passagem da equação 4.1 para a equação 4.2. A passagem da equação 4.2 para a equação 4.3 foi feita convertendo-se os coeficientes da série de ponto flutuante para ponto fixo utilizando-se para isso uma palavra de 16 *bits*, sendo 4 *bits* para a parte inteira e 12 *bits* para a parte fracionária (número de *bits* determinado pelo circuito da Figura 4.1). Essa palavra pode ser facilmente alterada de

acordo com a necessidade da aplicação, podendo-se assim conseguir maior ou menor precisão, dependendo unicamente do problema em questão. Os cálculos das funções seno e cosseno foram feitos no intervalo [0, $\pi/2$] e do arco-tangente em [0, 1].

$$\sin(x) = x - \frac{1}{6}x^3 + \frac{1}{20}x^5 \tag{4.1}$$

$$\sin(x) = x - 0,1662x^3 + 0,0077x^5 \tag{4.2}$$

$$\sin(x) = x - 681x^3 + 32x^5 \tag{4.3}$$

Pela equação 4.2 pode-se notar que o primeiro coeficiente da série não foi ajustado. Isso se deve ao fato de que fixando-se o primeiro coeficiente, e variando-se apenas os outros dois da série, o erro médio ficou em torno de 0,0051%. Com isso foi possível evitar o uso de um multiplicador na implementação em *hardware*, reduzindo-se assim o custo da função.

As equações 4.4, 4.5 e 4.6 apresentam as aproximações para a função cosseno, onde o erro médio obtido foi de 0,0323%. As equações 4.7, 4.8 e 4.9 apresentam as aproximações para a função arco-tangente onde o erro médio obtido foi de 0,0355%.

$$\cos(x) = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 \tag{4.4}$$

$$\cos(x) = 0,9996 - 0,4964x^2 + 0,0372x^4 \tag{4.5}$$

$$\cos(x) = 4094 - 2033x^2 + 154x^4 \tag{4.6}$$

$$\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 \tag{4.7}$$

$$\arctan(x) = 0,9959x - 0,2918x^3 + 0,0825x^5 \tag{4.8}$$

$$\arctan(x) = 4079x - 1195x^3 + 338x^5 \tag{4.9}$$

As Figuras 4.2, 4.3 e 4.4 mostram as simulações no *Matlab/Simulink* para as equações 4.2, 4.5 e 4.8 juntamente com o erros obtidos em relação à cada função calculada diretamente

no *Matlab*.

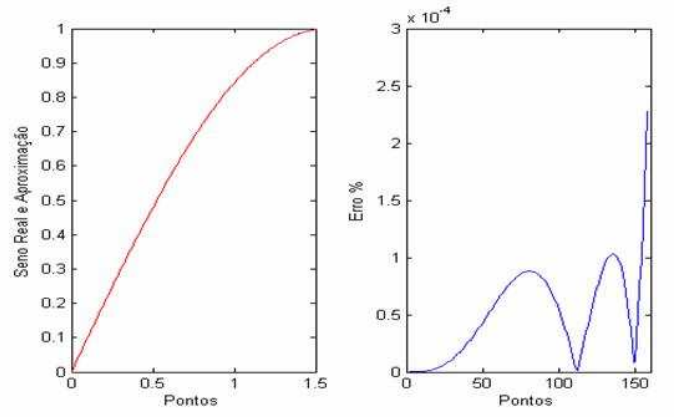

Figura 4.2: Simulação da aproximação da função seno.

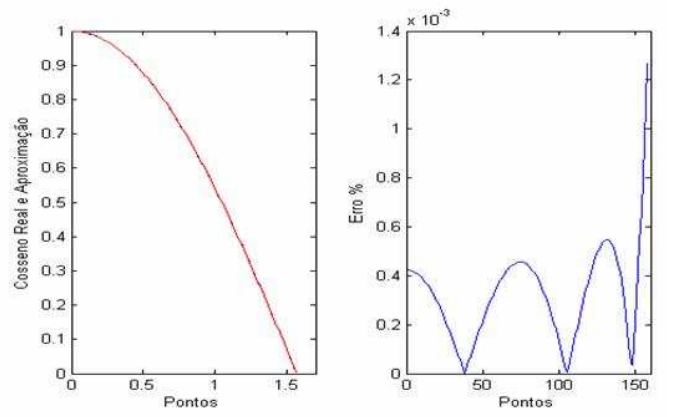

Figura 4.3: Simulação da aproximação da função cosseno.

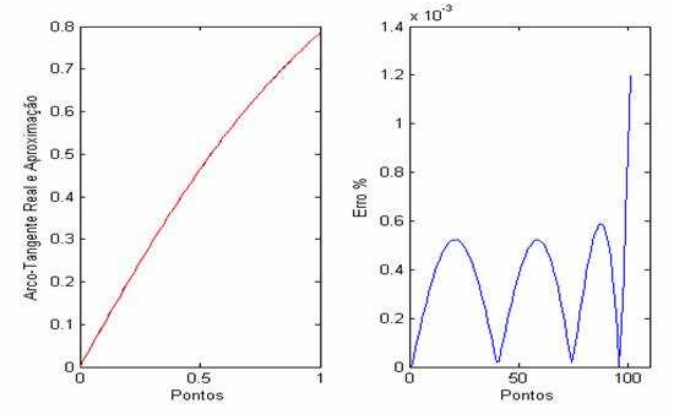

Figura 4.4: Simulação da aproximação da função arco-tangente.

Pode-se observar nos gráficos apresentados nas Figuras 4.2, 4.3 e 4.4, nas curvas dos erros, que para valores próximos de 1 o erro foi maior. Isso ocorre devido aos truncamentos feitos nas respectivas séries.

4.1.2 Arquiteturas do sistema

Para a implementação das equações 4.3, 4.6 e 4.9 em *hardware* foi utilizado o *software* de desenvolvimento *Quartus II*. Para isso foram utilizados multiplicadores, somadores e registradores de deslocamento (*flip-flop*). O diagrama de blocos da arquitetura da função cosseno está apresentado na Figura 4.5. Nele não estão representados os *flip-flop*, por simplicidade.

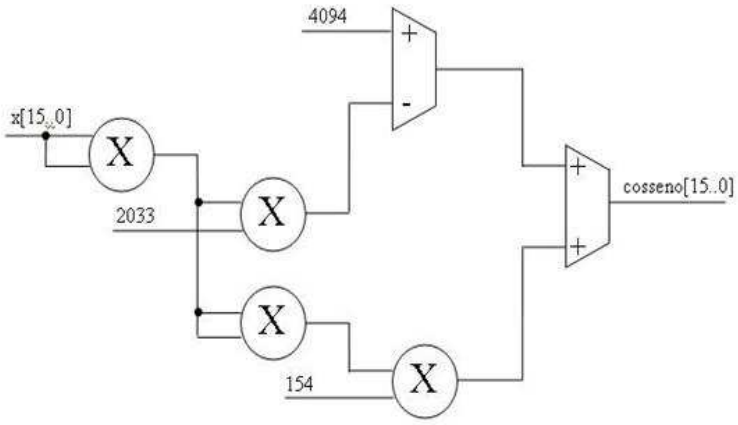

Figura 4.5: Arquitetura para implementação da função cosseno em *hardware*.

O valor do ângulo a ser calculado entra na forma de uma palavra de 16 *bits* em notação de ponto fixo, sendo 4 *bits* para parte inteira e 12 para parte fracionária, que pode variar de 0, a 1,57 radianos para a função seno e cosseno e de 0 a 1 para a função arco-tangente. O valor da saída também é uma palavra de 16 *bits*.

Todas as expansões foram feitas considerando-se apenas o primeiro quadrante, ou seja, ângulos de 0 a $\pi/2$ radianos. Para realizar o cálculo para valores no intervalo de 0 a 2π radianos foi necessário construir um circuito para redução de quadrantes, através do qual todos os valores são deslocados para o primeiro quadrante. O diagrama de blocos desse circuito está mostrado na Figura 4.6.

O circuito redutor de quadrantes funciona da seguinte maneira: compara o ângulo de entrada para determinar se ele é menor ou igual a $\pi/2$, menor ou igual a π, ou menor ou igual a $3\pi/2$. Dependendo das condições, o bloco denominado quadrante aciona o MUX para que este subtraia o valor adequado para que o valor de entrada seja representado no primeiro quadrante. Os dois *flip-flop* do diagrama são utilizados para que haja o sincronismo do relógio, e para que

tudo seja calculado em apenas um ciclo de relógio. O circuito fornece como saída o quadrante em que se encontra o ângulo de entrada e o respectivo valor de saída já ajustado.

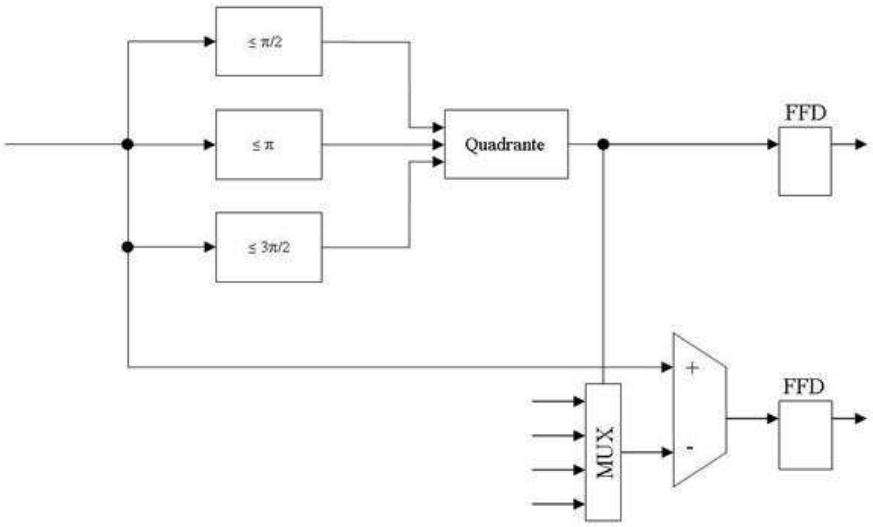

Figura 4.6: Diagrama de blocos do circuito redutor de quadrantes.

4.1.3 Resultados

As implementações das funções foram realizadas utilizando-se o *software* de desenvolvimento *Quartus II* e testadas na placa de desenvolvimento proposta deste trabalho. Para efeito de comparação, as funções foram simuladas também utilizando-se um *PLD* da família Stratix (Stratix, 2005) da Altera. O circuito de redução de quadrantes necessita de 98 *LE* (tanto para o dispositivo *Cyclone* como *Stratix*), a uma freqüência de operação de 80 MHz. A Tabela 4.1 mostra os resultados das compilações de cada função transcendental implementada em *hardware*, como uso de elementos lógicos, freqüência de operação, e número de multiplicadores embutidos utilizados (existentes apenas no componente Stratix). Pode-se notar pela 4.1 que para PLD que possuam em sua arquitetura multiplicadores embutidos, o uso de elementos lógicos ficou bem reduzido além de se conseguir freqüências de operação maiores. A Figura 4.7 mostra o circuito implementado na placa de desenvolvimento para a função cosseno.

A Figura 4.8 apresenta o diagrama de simulação do circuito utilizando-se o *software* *Quartus II*. Os dados de entrada e saída mostrados neste diagrama estão no formato ponto fixo. Pode-se notar através desta figura que para preencher o *pipeline* são necessários 5 ciclos de relógio. Após esses 5 ciclos, o primeiro resultado válido é mostrado (seta em linha contínua). No próximo ciclo já tem-se o valor calculado para a segunda entrada (seta pontilhada), e assim por diante. Com isso consegue-se realizar o cálculo da função cosseno em apenas um ciclo de máquina.

		Seno	Cosseno	Arco Tangente
Cyclone	LE	1382	1186	1625
	Freq	86 MHz	91,11 MHz	87,27 MHz
Stratix	LE	66	66	74
	Multiplicadores Embutidos	12	10	14
	Freq	112 MHz	110 MHz	112 MHz

Tabela 4.1: Resultado das compilações de cada função.

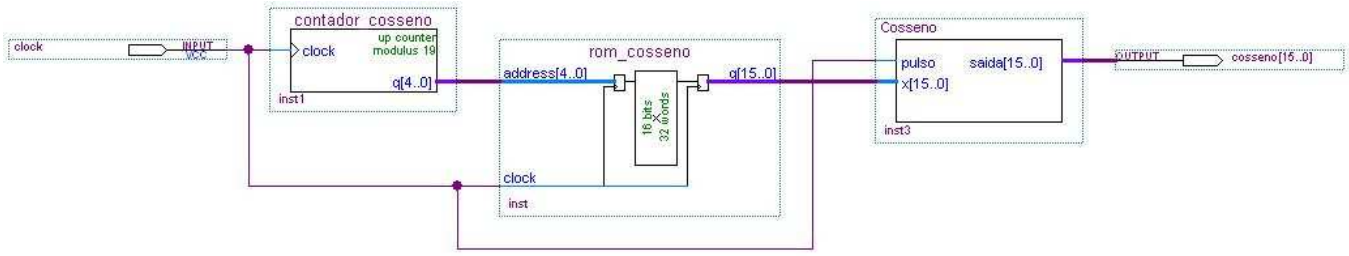

Figura 4.7: Circuito implementado na placa de desenvolvimento para a função cosseno.

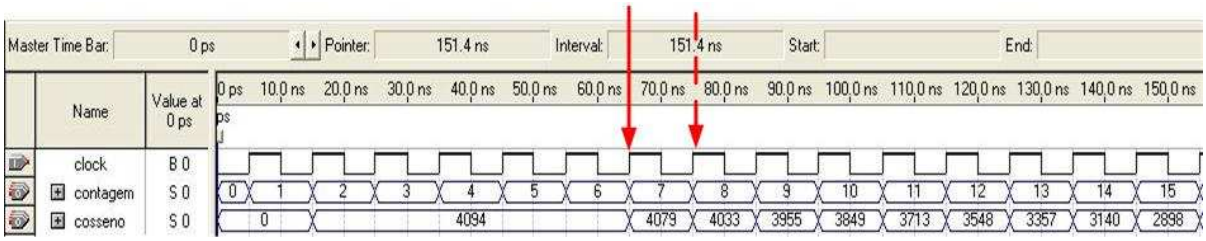

Figura 4.8: Diagrama de tempo para a função cosseno.

A Tabela 4.2 mostra os resultados obtidos nas simulações para alguns valores de entrada para cada função, e também uma comparação com o valor calculado pelo *Matlab* para determinação do erro. Os valores dos erros obtidos para cada valor de entrada e respectiva função estão mostrados na Tabela 4.3.

	Função MATLAB			**Função *Quartus***		
Ângulo	**Seno**	**Cosseno**	**Arco Tangente**	**Seno**	**Cosseno**	**Arco Tangente**
15^o	0,2588	0,9659	0,2561	0,2588	0,9656	0,2554
30^o	0,5000	0,8666	0,4823	0,5000	0,8665	0,4829
45^o	0,7071	0,7071	0,6658	0,7073	0,7075	0,6653
60^o	0,8666	0,5000	0,8084	0,8662	0,5000	0,8118

Tabela 4.2: Resultados das simulações de cada função.

Finalmente, a título de comparação, a capacidade de processamento da arquitetura apre-

	Erro %		
Ângulo	**Seno**	**Cosseno**	**Arco Tangente**
15^o	0	0,031059116	0,27333073
30^o	0	0,057736721	0,124403898
45^o	0,0282845	0,056569085	0,075097627
60^o	0,0230947	0	0,420583869

Tabela 4.3: Erros obtidos nas simulações em relação a valores calculados pelo *Matlab*.

sentada foi comparado com os respectivos cálculos das funções transcendentais realizadas por um microprocessador padrão. Os cálculos realizados por um microprocessador *Athlon XP* 2.1 GHz foram realizados através de código gerado por compilador de linguagem C, com o programa sendo executado no sistema operacional *DOS* 7.1. O teste consiste simplesmente no cálculo das funções seno e cosseno para 1000000 de valores de ângulos entre 0 a 2π radianos. O microprocessador necessitou de 0,1648 s. No caso da implementação no *hardware* proposto, operando em 80 MHz, em 1 ciclo de máquina e com as funções calculadas em paralelo são gastos 1000000[ciclos]x[1/80MHz] = 0,0125 s. Este resultado implica num ganho de velocidade de 13,184 vezes da arquitetura proposta em relação ao microprocessador usado nestes experimentos.

Uma aplicação direta do método proposto é para resolução da dinâmica de robôs industriais em *hardware*. Como exemplo, as equações dinâmicas (4.10) e (4.11) mostram tipicamente o que deve ser solucionado no sentido de se utilizar a dinâmica no controle. Note que apesar de aparentarem um pequeno número de operações matemáticas, existe a presença de termos com funções transcendentais, multiplicações e adições; estas equações estão considerando apenas os dois primeiros graus de liberdade do robô *Rhino XR-3*. No entanto sabe-se que o número de operações matemáticas necessárias para a resolução destes modelos cresce aproximadamente com taxa exponencial em relação ao número de graus de liberdade que um robô possa ter.

O *software* utilizado para auxiliar na obtenção dessas equações foi o *MAPLE* por ser muito eficiente para tratar com análise de equações simbólicas (Fazanaro, 2005).

$$\begin{aligned}\tau_1 = \; & -2m_2 L_{c_2}^2 \sin(\theta_2)\cos(\theta_2)\dot{q}_1\dot{q}_2 + \\ & (I_{yy}^{(2)} - I_{xx}^{(2)})\sin(\theta_1)\cos(\theta_1)(\ddot{q}_2)^2 + \\ & (m_2 L_{c_2}^2 \cos(\theta_2)^2 + I_{zz}^{(1)} + I_{zz}^{(2)})(\ddot{q}_1)^2\end{aligned} \tag{4.10}$$

$$\begin{aligned}\tau_2 = \; & -m_2 g L_{c_2}\cos(\theta_2) + m_2 L_{c_2}^2 \sin(\theta_2)\cos(\theta_2)\dot{q}_1^2 + \\ & 2(I_{xx}^{(2)} - I_{yy}^{(2)})\sin(\theta_1)\cos(\theta_1)\dot{q}_1\dot{q}_2 + \\ & (m_2 L_{c_2}^2 + I_{xx}^{(2)}\sin(\theta_1)^2 + I_{yy}^{(2)}\cos(\theta_1)^2)\ddot{q}_2\end{aligned} \tag{4.11}$$

Com os resultados de freqüência apresentados na Tabela 4.1 e sabendo-se que para

solução de funções algébricas (adições e multiplicações) em *hardware*, esta freqüência pode chegar até a 200MHz. Portanto, o cálculo da equação 4.10, por exemplo, pode ser realizado em 0,2μs. Esse resultado foi obtido considerando o pior caso, ou seja, sem considerar otimizações que poderiam ser conseguidas através de paralelizações dos cálculos necessários; demonstrando-se assim que mesmo que seja necessário o cálculo de uma quantidade maior em função de estar-se considerando robôs com mais graus de liberdade ainda assim é possível afirmar que esta proposta pode ser eficaz.

4.2 Acionamento de um pêndulo utilizando o processador NIOS

O pêndulo acionado além de ser um exemplo didático clássico no estudo de sistemas de controle, especificamente para a área de robótica ele serve para a realização de estudos e testes de comportamento de juntas, ou articulações, de estruturas de robôs. Os robôs de cadeias cinemáticas seriais podem ser interpretados como constituídos por um encadeamento de pêndulos acionados.

4.2.1 Apresentação do experimento montado

O pêndulo utilizado nesse experimento consiste de um sistema mecânico relativamente simples composto de dois eixos e engrenagens do tipo polias e correias sincronizadas. O motor está acoplado a um dos dois eixos através de uma engrenagem que reduz a velocidade por um fator de 4:1. O primeiro eixo está acoplado ao segundo também através de outra engrenagem com fator de redução de 4:1. Portanto, no primeiro eixo a velocidade do motor fica reduzida de 4:1 e no segundo de 16:1 em relação ao eixo do motor. O corpo do elo pode ser conectado a qualquer um dos eixos, em dependência da redução desejada[1].

O sensor utilizado para medir a posição angular do elo é um *encoder* óptico modelo *HEDS*-6310, que possui uma resolução de 1024 pulsos por volta. Este *encoder* dispõe de três fases de saída, sendo duas (FA e FB) deslocadas de 90 graus entre si, cujos sinais são usados para as medições de deslocamento angular, e uma terceira fase (FS) que emite um pulso de referência por rotação e que pode ser utilizado para estabelecer o referencial das medições.

O sistema para acionamento do pêndulo, ver Figuras 4.9 e 4.10, foi implementado utilizando-se a placa de desenvolvimento proposta nesse trabalho para o seu controle e geração dos sinais de PWM e, uma segunda placa, responsável pelo circuito eletrônico de potência para

[1] O projeto mecânico e a construção deste robô foram realizadas pelo Eng. Eduardo Gavira Bonani, atualmente aluno de mestrado que desenvolve seu trabalho de pesquisa no LSMR.

o acionamento dos motores, desenvolvida também no LSMR (Jungbeck, 2005). A idéia foi testar alguns subsistemas da placa com funções de operação fundamentais nas tarefas de controle de sistemas robóticos reais utilizando-se como paradigma o pêndulo acionado. Os blocos utilizados foram:

- Fonte de Alimentação;
- Interface *RS-232* (comunicação com o NIOS);
- Memória *SDRAM* (armazenamento do programa a ser executado pelo NIOS);
- Configuração *JTAG* e *AS* (ambas foram testadas);
- Dispositivo *Cyclone* (controle e geração dos sinais de PWM);
- Circuito de *PLL* (geração do relógio de $80MHz$);
- Pinos de E/S;
- Chaves e *Leds*.

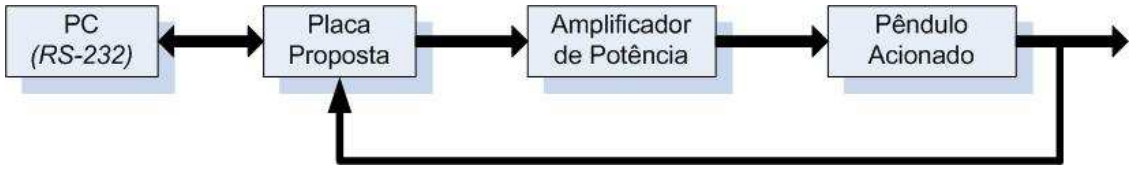

Figura 4.9: Diagrama de blocos do sistema completo.

O pêndulo foi acionado por um circuito PWM implementado no PLD, cujos sinais de PWM gerados pelo PLD são amplificados e condicionados pela placa de potência responsável pelo acionamento direto do motor. Foi implementado também um sistema para contagem dos sinais provenientes do *encoder* que servem para estabelecer as medidas de posição do pêndulo. Através dessa estratégia foi possível fechar a malha de controle (realimentação).

A técnica de acionamento através de modulação por largura de pulso foi utilizada por ser uma técnica de chaveamento muito eficiente do ponto de vista do controle energético e preciso, principalmente para o acionamento de motores elétricos de corrente contínua. E nestas experimentações do trabalho de mestrado foi utilizada a concepção para sua operação em quatro quadrantes, o que permite a aceleração e desaceleração em ambos os sentidos de giro dos motores. Isso em sistemas robotizados industriais é algo fundamental pois os robôs são máquinas que visam ter destrezas de movimentos em amplos espaços de trabalho e que esses movimentos sejam principalmente contínuos e precisos em posicionamento.

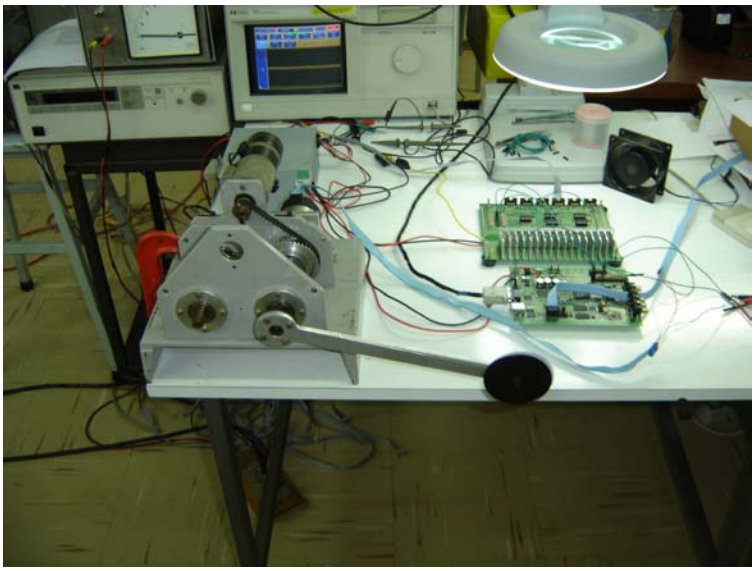

Figura 4.10: Montagem do sistema completo.

O processador NIOS foi utilizado para a geração e o controle do circuito de *PWM* onde a variação do *duty cycle* foi realizada através de uma palavra digital enviada à placa por um PC via interface de comunicação *RS-232*.

4.2.2 O sistema

A modulação *PWM* foi desenvolvida e integrada dentro do processador NIOS através de uma interface lógica (periférico) (Figura 4.11). Seu funcionamento pode ser explicado com o auxílio da Figura (4.12). O circuito *PWM* possui um contador de n *bits*, iniciando em 0 e finalizando em 2^n, e um valor de referência. O princípio de funcionamento consiste em se adotar um valor de referência qualquer de maneira que quando o contador atingir este valor pré-estabelecido, com o uso de um comparador, o nível do sinal de saída permaneça constante. Pela Figura (4.13) é fácil compreender esta situação. Portanto, para variar o *duty cicle* do *PWM*, é necessário apenas a variação do valor da referência. O circuito para geração do sinal de *PWM* foi implementado em 12 *bits*.

Para o acesso ao processador NIOS através de um PC (via interface *RS-232*), foi feito um programa em linguagem C utilizando a própria interface do *software Quartus II* o qual alterava o *duty cycle* do *PWM* de acordo com que o usuário determinasse. Esse programa foi compilado e executado através do *software* NIOS *Shell* da Altera.

Os sinais gerados pelo circuito de *PWM* estão mostrados na Figura 4.14. Esse sinais foram obtidos utilizando um osciloscópio *Tektronix TDS360*.

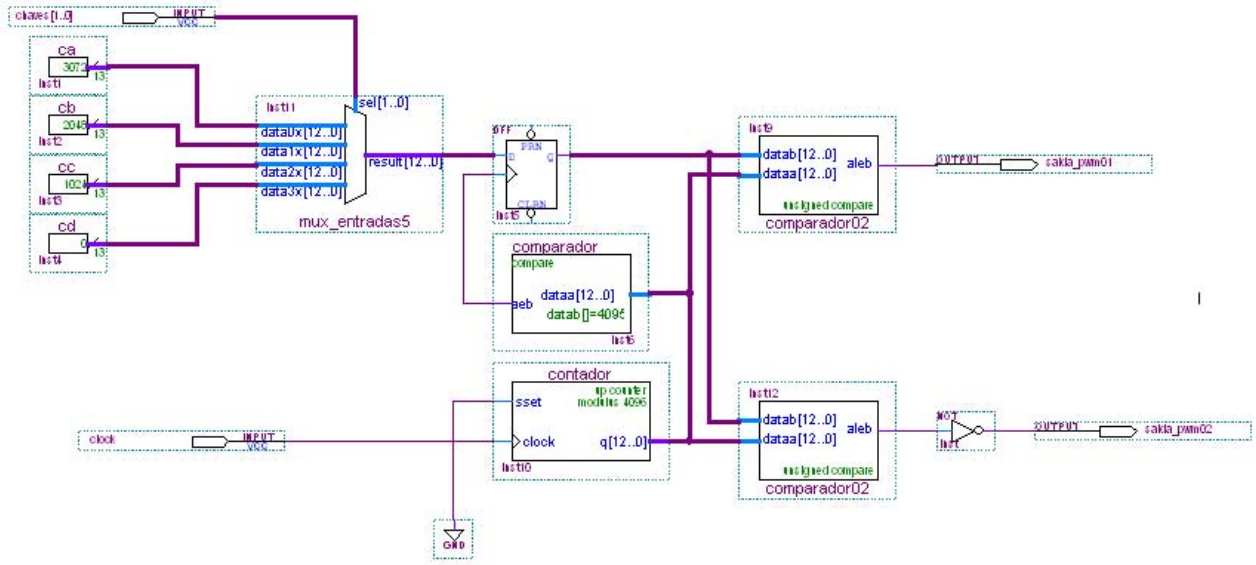

Figura 4.11: Circuito *PWM* implementado.

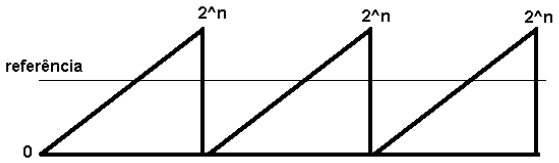

Figura 4.12: Funcionamento do circuito *PWM*.

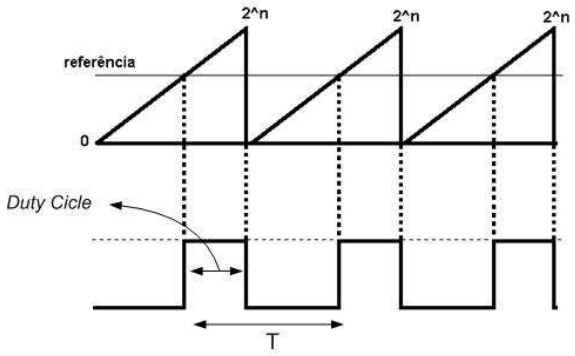

Figura 4.13: Esquema da conversão do contador.

A realimentação do sistema foi feita utilizando-se um circuito para contagem dos pulsos enviados pelo *encoder*. Na saída desse circuito de contagem foi adicionado um comparador com

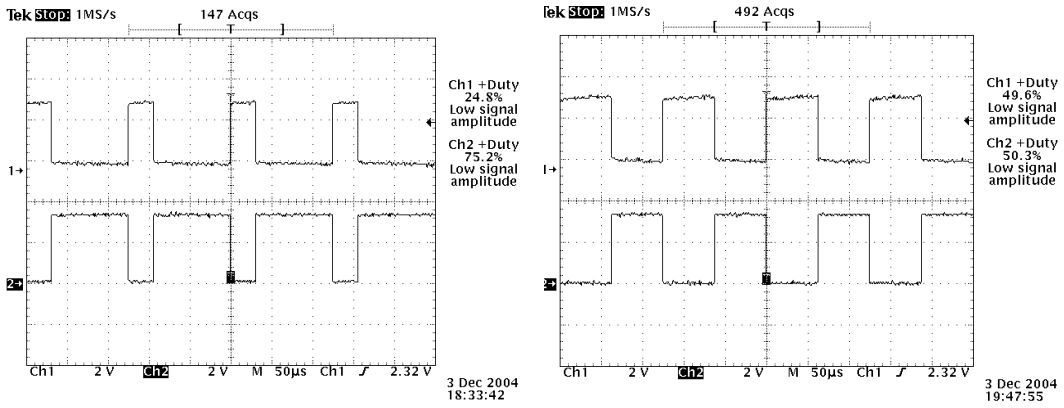

Figura 4.14: Sinais de PWM gerados pelo circuito implementado.

o qual determinava-se o número de pulsos (constante em uma das entradas do comparador) que o motor deveria girar em um determinado sentido. A inversão do sentido de giro do motor ocorria toda vez que a contagem atingia o valor da constante do comparador. Todo o sistema que foi implementado no *PLD* para esta finalidade está mostrado nas Figuras 4.15 e 4.16.

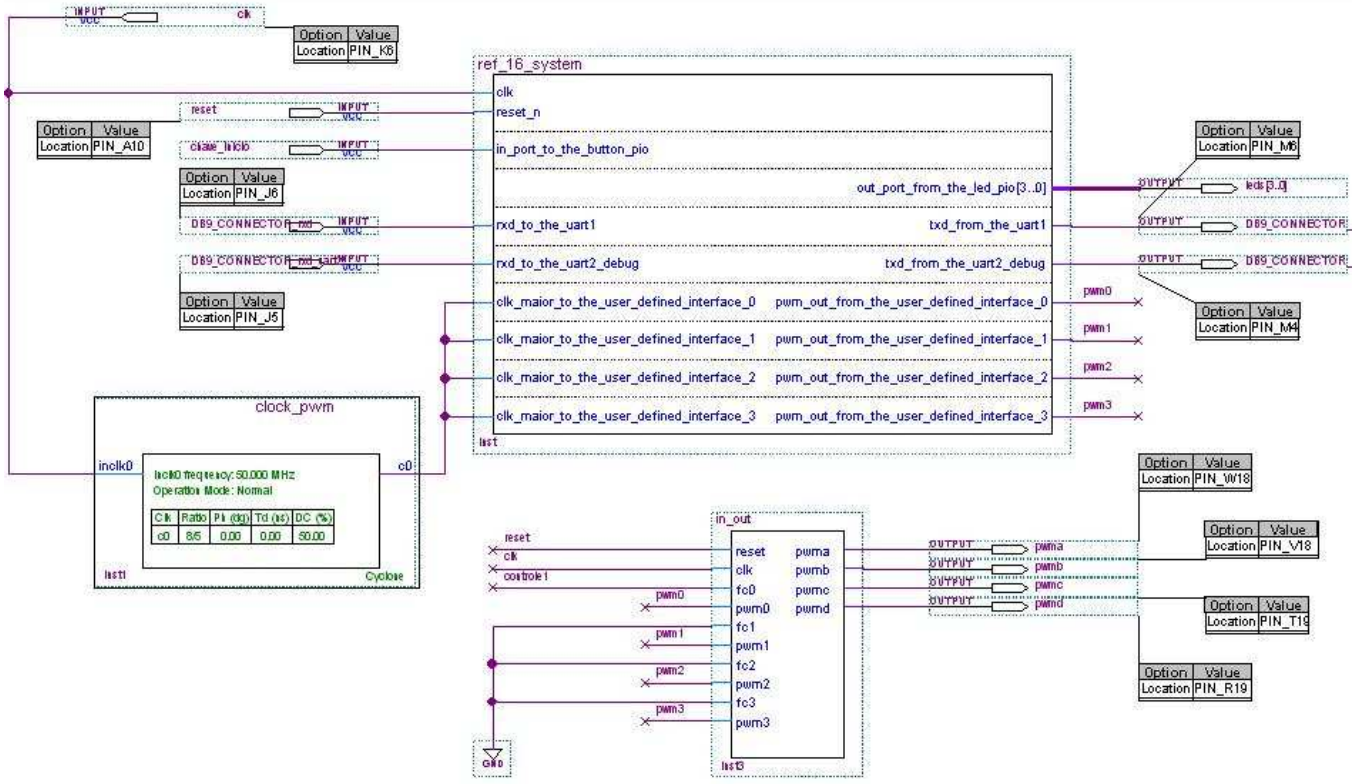

Figura 4.15: Sistema implementado no *PLD* - Parte 1/2.

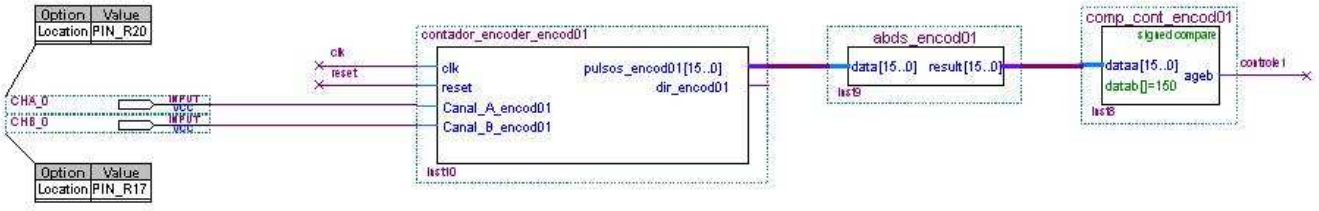

Figura 4.16: Sistema implementado no *PLD* - Parte 2/2.

4.3 Acionamento do robô cobra utilizando o *xPC Target*

O *xPC Target* permite que algoritmos desenvolvidos no *Simulink* sejam compilados e executados num microcomputador alvo com um sistema em tempo real (xPC Target, 2005). O computador usado para este fim tem um processador Intel *Celeron* 400 MHz com 128 M*bytes* de memória *RAM*. Este microcomputador foi conectado, via porta serial *RS-232*, com um outro computador, hospedeiro, que processa via *Matlab/Simulink* os algoritmos propostos.

Com o objetivo de desenvolver uma plataforma de baixo custo e complexidade de construção e que, ao mesmo tempo, ofereça as condições necessárias para a validação prática dos conceitos propostos neste trabalho, foi projetado e construído um pequeno manipulador planar redundante de 4 graus de liberdade[2].

Este manipulador é formado por quatro elos de alumínio de $0,2\ m$ de comprimento cada. A transmissão dos movimentos dos motores, que ficam sob a base do robô, é feita por polias e correias sincronizadas de forma a eliminar folgas nas transmissões dos movimentos das juntas.

Este robô, Figura 1.5, possui 4 motores de corrente contínua que o acionam por meio de caixas de redução de 80:1. O acionamento dos motores é feito pelo circuito de potência cuja montagem está mostrada na Figura 4.17 (Madrid, 1994). Este circuito é capaz de acionar 1 motor com técnica *PWM*. Cada motor do robô tem acoplado ao seu eixo um *encoder* incremental de 500 pulsos por rotação[3]. A geração dos sinais *PWM*, assim como a contagem dos pulsos dos *encoders* foram implementadas em *hardware* utilizando a placa de desenvolvimento proposta nesse trabalho, e em conformidade com o que foi apresentado no item 4.2.

Desta forma como o sistema do robô cobra foi montado, um sistema simples de visão (processamento todo realizado em *hardware*) pode ser adicionado externamente a ele de modo, por exemplo, a informar a posição da garra e/ou outras partes do seu corpo, além de possíveis obstáculos existentes no seu volume de trabalho.

[2]O projeto mecânico e a construção deste robô foram realizadas pelo Eng. Eduardo Gavira Bonani, atualmente aluno de mestrado que desenvolve seu trabalho de pesquisa no LSMR.

[3]Atualmente, os *encoders* estão adaptados aos eixos dos motores.

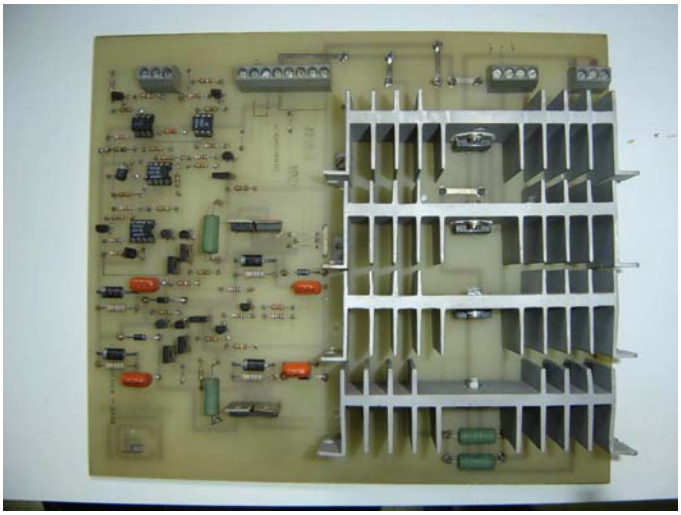

Figura 4.17: Base de acionamento dos motores.

4.3.1 Sistema de Controle

O sistema de controle está sendo implementado utilizando o *xPC target* da *MathWorks*. A idéia de se usar esta plataforma é a de proporcionar um meio de testar rapidamente os algoritmos que estão sendo desenvolvidos.

A comunicação dos dados entre o microcomputador alvo (*target*) e o robô foi realizada através do uso de uma placa de interface desenvolvida no LSMR, ver Figura 4.18. O circuito dessa placa utiliza-se da tecnologia de comunicação com barramento *ISA*.

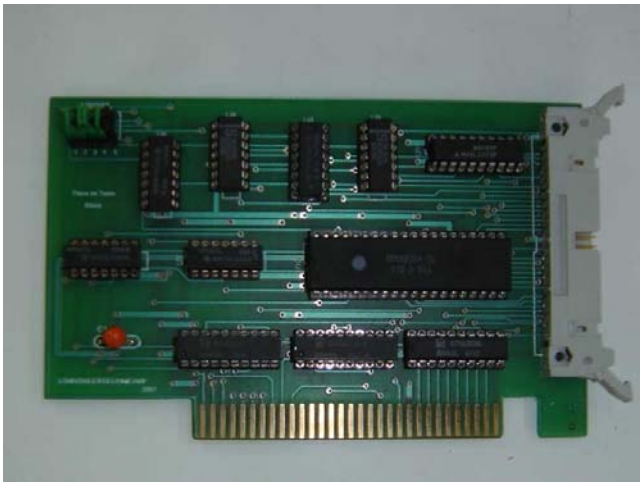

Figura 4.18: Placa de aquisição *ISA*.

A geração dos sinais *PWM* para o acionamento e os circuitos responsáveis pelos contadores dos pulsos dos *encoders*, foram implementados em *hardware* (placa de desenvolvimento).

A placa de desenvolvimento também foi utilizada para a realização de toda a decodificação de endereços para comunicação entre o computador alvo e o hospedeiro. Todo o sistema utilizou aproximadamente 1400 *LE*.

Os testes com os acionamentos dos motores do robô foram realizados através de uma interface amigável projetada dentro do ambiente do simulador *Matlab/Simulink*, ou seja, através de uma janela contendo cursores, ver Figuras 4.19 e 4.20, por onde foi possível variar-se o *duty cycle* de cada PWM individualmente.

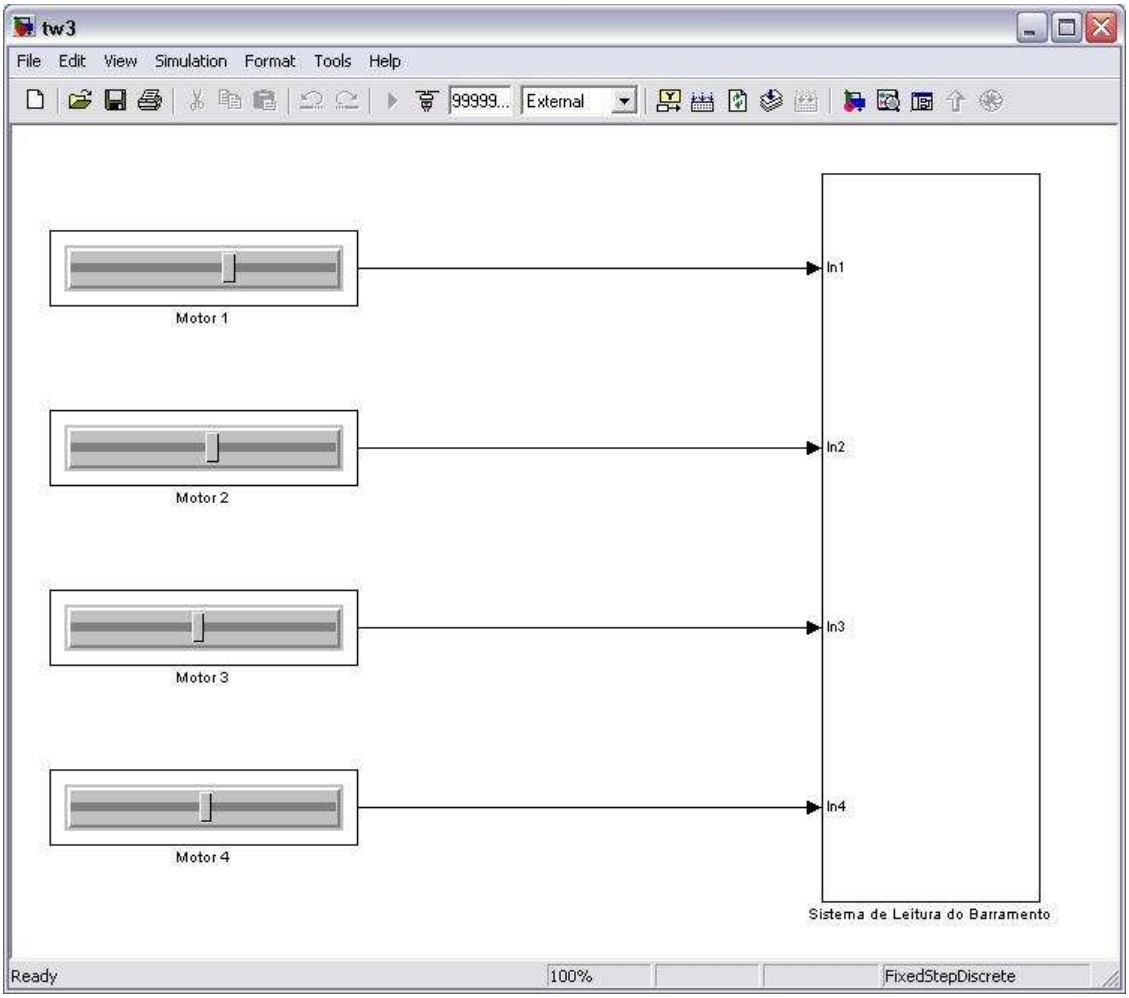

Figura 4.19: Acionamento dos motores via *Matlab/Simulink*.

O sistema de potência, responsável pelo acionamento dos motores, utilizado opera na freqüência de 5KHz e possui uma potência de aproximadamente 360 VA (24V x 15A - para cada motor) (Madrid, 1994). A filosofia básica do amplificador *PWM* utilizado em topologia Ponte H é fazer com que a partir de uma fonte de alimentação com tensão fixa de +V, seja possível injetar na carga, tensões de +V e -V. Isto é conseguido através do circuito esquematizado na Figura 4.21. Quando S = 1, as chaves S1 e S3 fecham, e S2 e S4 abrem, então aparece a tensão

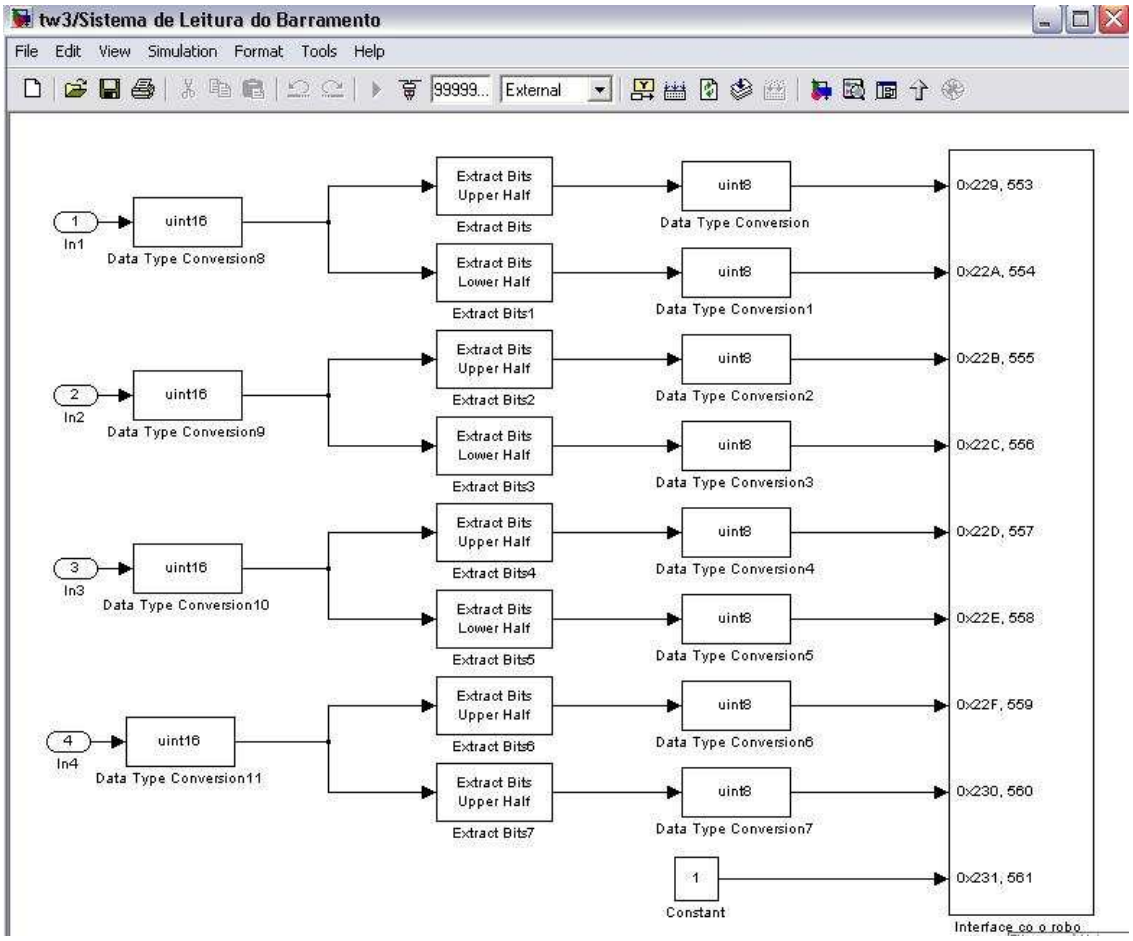

Figura 4.20: Sistema de leitura do barramento via *Matlab/Simulink*.

+V do lados esquerdo da carga, e quando S = 0, S1 e S3 fecham, fazendo com que apareça a tensão +V do lado direito da carga. A Figura 4.22 mostra todo o sistema.

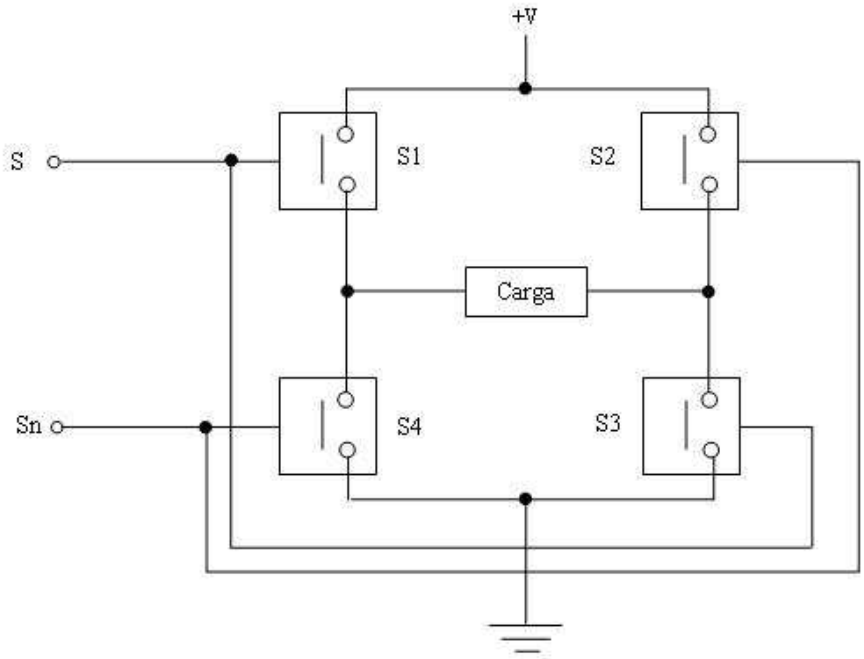

Figura 4.21: Esquema do amplificador PWM em topologia Ponte H.

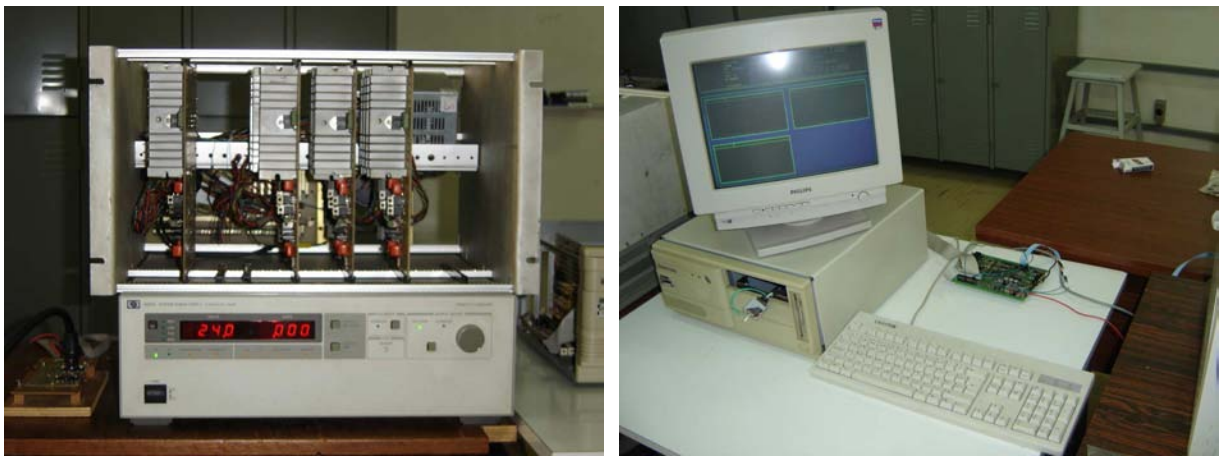

Figura 4.22: Foto da bancada experimental.

Capítulo 5

Conclusões e Perspectivas Futuras

5.1 Conclusões

Baseando-se nos resultados obtidos e mostrados nos experimentos que constam no capítulo 3, é possível afirmar que o sistema proposto pode viabilizar o controle de robôs industriais acionados eletricamente, inclusive com a possibilidade de utilização de estratégias que consideram a resolução de modelos matemáticos de suas dinâmicas e cinemáticas, prevendo-se ainda a incorporação auxiliar de visão artificial. A afirmação que isso seja possível é principalmente oriunda das observações dos resultados obtidos nas experimentações para a solução ultra-rápida que pode-se conseguir para as funções transcendentais. Ficou comprovado por estes resultados obtidos que tais funções podem ser solucionadas em apenas um ciclo de máquina, sendo que para o sistema projetado o relógio que fornece a cadência de trabalho para a operação destes respectivos circuitos pode ser de até 112MHz, o que significa resolver uma função desta natureza a cada 8,93 nanossegundos. Da mesma maneira, e ainda mais rapidamente se for o caso, é possível paralelamente resolver operações algébricas básicas como adição e multiplicação. Devido a isso, e conforme mostrado no capítulo 3, torna-se possível realizar cálculos muito complexos utilizando-se processamento paralelo e em tempo real para processos com constantes de tempo típicas dos sistemas robóticos.

Com a resolução da modelagem dinâmica em tempo real será possível tornar os robôs industriais muito eficientes, mesmo para a realização de tarefas de grande complexidade, tendo-se melhores condições para a realização de atividades que envolvam precisão acurada e repetibilidade, e pode-se aumentar em muito a velocidade de realização das tarefas, com maior controle sobre os efeitos causados pelas ações de forças e torques não-lineares, e possivelmente otimizar os gastos energéticos globais dos sistemas alvos dessas aplicações.

A partir dos resultados obtidos nas experimentações envolvendo o pêndulo acionado, foi possível demonstrar que o sistema eletrônico proposto pode implementar desde malhas

de controle realimentadas simples até de grandes complexidades, conforme as exigências e restrições impostas para os servomecanismos robóticos; isso possibilita que mais facilmente possam ser aplicados e testados métodos e técnicas avançadas como as propostas em publicações científicas, e com enormes facilidades para suas adaptações e reprogramações.

Destaca-se também que estas experimentações com o pêndulo acionado serviram como método de depuração para uma grande quantidade de sub-sistemas incorporados no projeto, tornando-os validados em relação ao objetivo do trabalho estabelecido.

Uma outra constatação importante foi sobre a utilização do processador NIOS. Através dele foi possível controlar todos os periféricos utilizados no experimento e ao mesmo tempo utilizá-lo no controle do servomecanismo.

Através das experimentações envolvendo o robô Cobra, foi possível também verificar as eficiências do sistema quando aplicado ao controle de diferentes processos, sendo que neste caso, foi utilizada uma estratégia variante que pode ser também eficiente para aplicações que exijam garantias de tempo real, e que paralelamente também demandem rapidez nas ações, como é o caso dos sistemas robóticos industriais. Esta segunda estratégia foi baseada no *Matlab/Simulink* denominada *xPC Target*, e ficou comprovado que sua realização pode ser conseguida de maneira muito simples quando integrada com o sistema eletrônico proposto.

Como na atualidade existem diversos tipos de *PLD* com diversas tecnologias (*Cyclone II*, *Stratix I* e *II* e *PLD* de outros fabricantes (*Atmel*, *Xilinx* etc)), se for o caso, poder-se-ia usar a mesma metodologia concebida nesse trabalho para projetar outros circuitos semelhantes para tais finalidades com maior ou menor poder de processamento o que daria flexibilidade para ajustar relações entre custos e benefícios.

Baseando-se no fato de que o sistema proposto é capaz de implementar máquinas de estado, então tornam-se incalculáveis as possibilidades de inserções de estratégias para obter-se soluções de equacionamentos e algoritmos, principalmente para técnicas avançadas de controle moderno e inteligência artificial.

Com o *PLD* utilizado no trabalho, e também com outros diversos disponíveis na atualidade, é possível embutir-se microcontroladores e/ou *DSP* dedicados e estender-se portanto as respectivas potencialidades, utilizando-se o que se pode ter de mais avançado em termos de arquiteturas de computadores para o processamento paralelo conhecidas até o momento, com uma redução sem precedentes históricos nos custos de produção e com enorme acréscimo de desempenho em relação a outros sistemas menos flexíveis que possam oferecer tais características.

Conclui-se também que devido às reduzidas dimensões físicas que pode-se conseguir, na confecção da placa de circuito impresso que contém o sistema, e também ao seu grande poder de processamento e velocidade, abrem-se promissoras possibilidades para a realização de sistemas de controle embarcados para tempo real (balões dirigíveis não tripulados, barcos e aviões não

tripulados etc).

Por estas razões, acredita-se que este trabalho obteve excelentes resultados no contexto das expectativas nacionais para o domínio tecnológico de sistemas de controles eletrônicos que ofereçam grande poder de processamento, adaptabilidade, e que sejam economicamente viáveis para aplicações em processos desde simples até de grandes complexidades.

5.2 Perspectivas Futuras

Visando a continuação e a extensão da pesquisa realizada, foi feita a seguinte listagem contendo idéias e sugestões para trabalhos futuros:

- O trabalho realizado servirá de fomento para realização/desenvolvimento de vários outros que já estão em desenvolvimento e que ainda estão planejados para serem realizados no LSMR;
- Realizar/Implementar testes na interface de comunicação USB;
- Implementação de protocolo para comunicação e realização de testes no sistema de visão implementado na placa de desenvolvimento;
- Implementação do processador NIOS para realizar/auxiliar no controle do robô Cobra;
- Implementações de equações dinâmicas no sistema desenvolvido para controle de robôs industriais;
- Integração do sistema desenvolvido com sistemas analógicos, como por exemplo, os *FPAA* (*Field Programmable Analog Array*) (hibridismo);
- Estudo da utilização do sistema desenvolvido para outras aplicações, por exemplo: telefonia, telecomunicações, simulações de sistemas etc;
- Proposições e testes de técnicas para a identificação e o controle de sistemas dinâmicos que podem operar no caos;

Apêndice A

Índice de Autores

Referências Bibliográficas

656, P. I. (2005), 'Referência para o padrão ITU 656', http://www17.cds.ne.jp/ stray/c-cir656.html.

Aguirre, L. A. (2000), *Introdução a Identificação de Sistemas: Técnicas Lineares e Não-Lineares Aplicadas a Sistemas Reais*, Editora UFMG.

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

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

Arribas, P. C. e Macia, F. M. H. (2002), FPGA board for real time vision development systems, *em* 'Proceedings of the Fourth IEEE International Caracas Conference on Devices, Circuits and Systems', pp. T021–1 – T021–6. IEEE Conference Proceeding.

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

Bailak, G. V., Rubinger, B., Jang, M. e Dawson, F. (2004), Advanced Robotics Mechatronics System: emerging technologies for interplanetary robotics, *em* 'Canadian Conference on Electrical and Computer Engineering', Vol. 4, pp. 2025 – 2028. IEEE Conference Proceeding.

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

Boluda, J. A., Pardo, F., Blasco, F. e Pelechano, J. (1999), A pipelined reconfigurable architecture for visual-based navigation, *em* 'Proceedings. 25th EUROMICRO Conference', Vol. 1, pp. 71 – 74. IEEE Conference Proceeding.

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

Cyclone (2005), '*Link* para central de documentação do PLD Cyclone', http://www.altera.com/products/devices/cyclone/cyc-index.jsp.

de Aguiar Dias, M. (1991), Controlador Programável a Multimicroprocessadores para Controle Hierárquico de Robôs, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP.

de Souza, M. A. T. (2000), Otimização de controladores nebulosos de Takagi-Sugeno utilizando algoritmos genéticos, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP.

Denning, D., Irvine, J., Stark, D. e Delvin, M. (2004), Multi-user FPGA co-simulation over TCP/IP, *em* '15th IEEE International Workshop on Rapid System Prototyping', pp. 151 – 156. IEEE Conference Proceeding.

Fazanaro, F. I. (2005), 'Controle de um Conjunto de Antenas de Comunicação presente em um Sistema composto por um Barco Robô Autônomo e a sua Central de Monitoramento', Trabalho de Iniciação Científica (SAE).

Guardia Filho, L. E., Nicolato, F. e Madrid, M. K. (2004), Hardware Implementation of Transcendental Functions Using Programmable Logic Devices, *em* 'VI Induscon - Joinville - SC'. IEEE Conference Proceeding.

Fischman, M. A. e Le, C. (2004), Digital beamforming developments for the joint NASA/Air Force Space Based Radar, *em* 'IEEE International Geoscience and Remote Sensing Symposium.', Vol. 1. IEEE Conference Proceeding.

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

He, P., Jin, M. H., Yang, L., Wei, R., Liu, Y. W., Cai, H. G., Liu, H., Seitz, N., Butterfass, J. e Hirzinger, G. (2004), High performance DSP/FPGA controller for implementation of HIT/DLR dexterous robot hand, *em* 'Proceedings. ICRA '04. 2004 IEEE International Conference on Robotics and Automation', Vol. 4, pp. 3397 – 3402. IEEE Conference Proceeding.

Hoshi, N., Utsumi, Y. e Oguchi, K. (2004), A Web-accessible FPGA-based direct torque controller for permanent magnet synchronous motor, *em* 'The 4th International Power Electronics and Motion Control Conference', Vol. 3, pp. 1325 – 1330. IEEE Conference Proceeding.

Jungbeck, M. (2001), Implementação de controladores neurais de Kim-Lewis-Dawson com parâmetros otimizados por algorítmos genéticos, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP.

Jungbeck, M. (2005), Estudo Comparativo Experimental de Técnicas de Controle de Robôs Manipuladores, Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP. Em preparação.

Kolodko, J. e Vlacic, L. (2003), Experimental system for real-time motion estimation, *em* 'AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics', Vol. 2, pp. 981 – 986. IEEE Conference Proceeding.

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

Li, J., Shrivastava, P., Gao, Z. e Zhang, H.-C. (2004), 'Printed Circuit Board Recycling: A State-of-the-Art Survey', *IEEE Transactions on Electronics Packaging Manufacturing* **27**(1), 33–42.

Madrid, M. K. (1988), Robô-Manipulador Mecânico TRRR para Posicionamento Espacial com Controle Digital Hierárquico à Micro-processadores., Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP.

Madrid, M. K. (1994), Controle de Trajetórias Contínuas por Seccionamento em Sub-Trejetórias Usando Inteligência Artificial num Robô Multi-Tarefas, Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP.

Martinez-Alvarez, J. J., Guerrero-Gonzalez, A., Pedreno-Molina, J. L., Villaescusa-Fernandez, A., Ferrandez, J. M. e Lopez-Coronado, J. (2002), Hardware implementation of a controller based on neurobiological adaptive models of the human motor-control system, *em* 'IEEE International Conference on Systems, Man and Cybernetics', Vol. 5, p. 3 pp. IEEE Conference Proceeding.

Mathworks (2005), '*Link* para a página principal', http://www.mathworks.com/.

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

Nicolato, F. (2002), Arquitetura de Hardware para a Extração em Tempo Real de Características de Múltiplos Objetos em Imagens de Vídeo: Classificação de Cores e Localização de Centróides, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP.

Nicolato, F. (2005), Desenvolvimento de Hardware para o Rastreamento de Trajetórias Contínuas de Robôs Industriais Baseado em Busca Heurística, Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação - UNICAMP. Em preparação.

NIOSI (2005), '*Link* para obter informações/documentação do software', http://www.altera.com/literature/lit-nio.jsp.

NIOSII (2005), '*Link* para obter informações/documentação do software', http://www.altera.com/literature/lit-nio2.jsp.

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

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

QuartusII (2005), '*Link* para obter informações/documentação do software', http://www.altera.com/products/software/products/quartus2/qts-index.html.

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

Roggen, D., Hofmann, S., Thoma, Y. e Floreano, D. (2003), Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot, *em* 'Proceedings of The 2003 NASA/Dod Conference on Envolvable Hardware.'. IEEE Conference Proceeding.

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

Sciavicco, L. e Siciliano, B. (1996), *Modeling and Control of Robot Manipulators*, McGraw-Hill International Editions.

SOPC (2005), '*Link* para obter informações/documentação do software', http://www.altera.com/products/software/products/sopc/sop-index.html.

Stratix (2005), '*Link* para central de documentação do PLD Stratix', http://www.altera.com/products/devices/stratix/stx-index.jsp.

Tang, W. e Yip, L. (2004), Hardware implementation of genetic algorithms using FPGA, *em* 'The 2004 47th Midwest Symposium on Circuits and Systems', Vol. 1. IEEE Conference Proceeding.

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

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

Vahid, F. (2003), 'The Softening of Hardware', *IEEE Computer Society* pp. 27 – 34. April.

VoB, N., Eisenbach, T. e Mertsching, B. (2004), A rapid prototyping framework for audio signal processing algorithms, *em* 'IEEE International Conference on Field-Programmable Technology', pp. 375 – 378. IEEE Conference Proceeding.

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

xPC Target (2005), '*Link* para a página central de documentação', http://www.mathworks.com/access/helpdesk/help/toolbox/xpc/.