UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Proposta de Ambiente Baseado em Computação Reconfigurável para Aplicação em Protótipos de Sistemas Embarcados

Autor: Carlos Raimundo Erig Lima
Orientador: João Maurício Rosário
Co-orientador: Didier Dumur

06/03

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE PROJETO MECÂNICO

Proposta de Ambiente Baseado em Computação Reconfigurável para Aplicação em Protótipos de Sistemas Embarcados

Autor: Carlos Raimundo Erig Lima
Orientador: João Maurício Rosário
Co-orientador: Didier Dumur (Supelc-FR)

Curso: Engenharia Mecânica
Área de Concentração: Projeto Mecânico

Tese de doutorado acadêmico apresentada à comissão de Pós-graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Doutor em Engenharia Mecânica.

Campinas, 2003.
SP – Brasil

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE PROJETO MECÂNICO

TESE DE DOUTORADO

Proposta de Ambiente Baseado em Computação Reconfigurável para Aplicação em Protótipos de Sistemas Embarcados

Autor: Carlos Raimundo Erig Lima
Orientador: João Maurício Rosário
Co-orientador: Didier Dumur (Supelc-FR)

Prof. Dr. João Maurício Rosário, Presidente.
Universidade Estadual de Campinas.

Prof. Dr. Alfranci Freitas Santos.
Centro Federal de Educação Tecnológica do Paraná.

Prof. Dr. Edson Roberto de Pieri.
Universidade Federal de Santa Catarina.

Prof. Dr. José Manoel Balthazar.
Universidade Estadual de Campinas.

Prof. Dr. Luiz Otávio Saraiva Ferreira.
Universidade Estadual de Campinas.

Campinas, 18 de fevereiro de 2003.

Agradecimentos

Presto a minha mais sincera homenagem a todos aqueles que de forma direta ou indireta ajudaram a construir este trabalho:

A minha família, pelo contínuo apóio, motivação e infinita paciência.

Ao pessoal do Laboratório de Automação Integrada e Robótica da Faculdade de Engenharia Mecânica da Unicamp, em especial ao meu orientador prof. João Maurício Rosário pelos valiosos conselhos e orientação.

A todos os professores e funcionários do *Service d'Automatique de Supelec*, França, em especial ao meu co-orientador M. Didier Dumur.

A todos os professores e funcionários do CEFET-PR e do Departamento de Projeto Mecânico da Faculdade de Engenharia Mecânica da UNICAMP que ajudaram de forma direta e indireta na conclusão deste trabalho. Agradecimentos especiais ao Professor Alfranci Freitas Santos pela minuciosa revisão gramatical.

Ao técnico mecatrônico Almiro Franco da Silveira Junior e Dr. Marcos Saramago e a todos os alunos da equipe Marthe do Curso de Engenharia Mecatrônica da UNICAMP.

Aos integrantes da família Garcia pelo apoio na construção de vários componentes mecânicos.

A empresa Altera e seus representantes brasileiros, PI Componentes, pelo suporte oferecido.

.. e mãos a obra, que se faz tarde; temos muito que fazer, que dizer e que mostrar.
Cervantes.

Resumo

LIMA, Carlos Raimundo Erig, *Proposta de Ambiente Baseado em Computação Reconfigurável para aplicação em Protótipos de Sistemas Embarcados*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2003. 200 p. Dissertação (Doutorado)

Devido à rápida evolução das inovações tecnológicas de hardware e de software na área de sistemas embarcadados, tornar-se mais e mais necessário desenvolver aplicações baseadas em metodologia que levem em conta a facilidade de futuras modificações, modernizações e expansões do sistema projetado. Considerando estes requisitos, este trabalho apresenta uma proposta de ambiente baseado em computação reconfigurável aplicado ao projeto de protótipos de sistemas embarcados. A principal motivação deste trabalho de Doutoramento é propiciar um ambiente que facilite o desenvolvimento de protótipos de sistemas embarcados, usando ferramentas simples para controlar os diferentes sensores e atuadores nesta categoria de projeto. O software e hardware implementados são estruturados em blocos independentes, através da implementação da arquitetura hierárquica aberta, distribuindo as diversas ações de controle em níveis crescentes de complexidade e utilizando recursos de computação reconfigurável. Propõe-se a validação deste ambiente em três dispositivos: a automação de uma cadeira de rodas e dois robôs móveis.

Palavras Chave

Computação Reconfigurável, Sistemas Embarcados, Robótica Móvel.

Abstract

LIMA, Carlos Raimundo Erig, *Proposal of Enviroment Using Reconfigurable Computing Applied to Embedded Systems Prototype Design*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2003. 200 p. Dissertação (Doutorado).

Nowadays Software and Hardware technologies are driven by innovation speed. As a consequence it becomes more and more necessary to develop applications based on methodologies which take into account the easiness of future modifications, updates and improvements in the designed system. According to these requirements, this work presents a Proposal of Enviroment Using Reconfigurable Computing Applied to Embedded Systems Prototype Design. The main motivation of this work is to provide an environment to simplify the embedded systems prototype design, using simple tools to control the several sensors and actuator usually present in this project class. The developed software and hardware are structured in independent blocks, through open architecture implementation using reconfigurable logic, allowing the easy expansion of the system, better adapting the embedded system to the tasks associated to it. Finally, an experimental validation is realized using three practical embedded systems (wheel chair and mobile robots).

Keywords*:*

Reconfigurable Computing, Embedded Systems, Mobile Robotics.

Índice

Lista de Figuras

Lista de Tabelas

Nomenclatura

AHDL - Altera Hardware Description Language

API – Application Program Interface

ASIC - Application Specific Integrated Circuits

CAD - Computer Aid Design

CCM - custom computing machines

CPLD - Complex Programmable Logic Devices

DFGA - Dynamically Field Programmable Gate Array

DSP – Digital Signal Processor

EPLD – Erasable Programmable Logic Device

FCCM - FPGA-based custom computing machines

FFT – Fast Fourier Transform

FPGA - Field Programmable Gate Array

IP-CORE – Intellectual Property Core

PLDs - Programmable Logic Devices

PLL – Phase-Locked Loops

VHDL - VHSIC Hardware Description Language

Capítulo 1

Introdução

O desenvolvimento de protótipos de sistemas embarcados tem sido alvo de inúmeros trabalhos na área de automação e controle. Impressoras de computadores, automóveis, robôs industriais, robôs móveis, aparelhos de telefonia, cadeiras de rodas, próteses antropomórficas, e inúmeros outros dispositivos que congregam sensores e atuadores para a execução de alguma tarefa e que exigem maior ou menor volume de controle, são sistemas embarcados ou utilizam sistemas embarcados. Usualmente são utilizados microcontroladores com algoritmos desenvolvidos em várias linguagens de *software* para realizar o controle destes dispositivos. Mais recentemente, alguns destes algoritmos foram também desenvolvidos em trabalhos associados com computação reconfigurável, com vantagens em termos de desempenho.

Entre todos os campos associados com projeto de sistemas embarcados, as tecnologias de *software* e *hardware* são as que tem experimentado a mais rápida evolução. É muito grande a quantidade de novos microprocessadores, interfaces de comunicação, interfaces de potência, sensores, compiladores, sistemas operacionais e sistemas de desenvolvimento fornecidos aos engenheiros a cada ano. Em função desta acelerada evolução tecnológica, a idéia de se utilizar estruturas abertas e reconfiguráveis, que possam adaptar-se a novas demandas, torna-se muito atraente, sendo mesmo um pré-requisito na consideração de um projeto de um sistema embarcado. Sistemas de desenvolvimento baseados em computação reconfigurável (sistemas de

hardware reconfigurável) apresentam características adequadas para auxiliar na execução desta classe de projetos. Eles apresentam, entre outras vantagens, as características de baixo consumo, alta velocidade de operação, capacidade de integração, flexibilidade, facilidade de programação e operação modular.

O conceito de sistemas abertos tem sido estudado nos últimos anos por várias instituições no campo de máquinas ferramentas e engenharia de produção, focalizando os aspectos de modularidade, os efeitos da arquitetura de controle e da rede de comunicação no desempenho do sistema. O objetivo deste conceito aplicado à arquitetura reconfigurável é permitir uma fácil e rápida adaptação de dispositivos embarcados a novas evoluções tecnológicas, para uma melhor portabilidade e capacidade de intercâmbio para o sistema final. A divisão da estrutura em pequenos blocos funcionais, com interfaces bem especificadas, permite uma melhor definição das tarefas de uma equipe de projeto multidisciplinar, bem como a rápida adaptação de um determinado bloco a uma nova evolução tecnológica. Este tipo de solução não pode ser alcançada em muitos dos sistemas comerciais disponíveis, pois estes não fornecem informações ou não apresentam recursos para permitir este tipo de manipulação.

Este trabalho implementa uma proposta de ambiente baseado em computação reconfigurável para implementação em protótipos de sistemas embarcados. São considerados aspectos de *hardware* e *software*. A motivação deste trabalho dentro do contexto da Tese de Doutorado é propiciar um ambiente que facilite o desenvolvimento de protótipos de sistemas embarcados, enfatizando o desenvolvimento de ferramentas simples que permitam o controle e a monitoração dos diferentes sensores e atuadores nesta categoria de projeto. Como conseqüência, é proposto o desenvolvimento de pequenos módulos independentes com interfaces de comunicação bem definidas como parte de uma estrutura orientada a uma arquitetura aberta.

Através da proposta de uma arquitetura hierárquica e aberta, distribuindo as diversas ações de controle em níveis crescentes de complexidade e da utilização de recursos de computação reconfigurável, propõe-se a validação deste ambiente em três dispositivos: dois robôs móveis

experimentais e uma cadeira de rodas automatizada. Na validação experimental, feita nos três dispositivos embarcados propostos, fica clara a flexibilidade e adaptabilidade do ambiente desenvolvido.

Dentre os sistemas embarcados, os robôs móveis têm-se apresentado como plataformas para consolidação de conhecimento em diversas áreas de pesquisa, como modelagem, controle, automação, sistemas de potência, sensores, transmissão de dados, eletrônica embarcada e engenharia de *software*, sendo cada vez mais usadas em instituições de ensino e pesquisa. Já a cadeira de rodas automatizada, além de apresentar, quanto ao aspecto de consolidação de conhecimento, muitas das vantagens dos robôs móveis, apresentam um forte caráter social, sendo adotada por esta razão.

Uma revisão bibliográfica sobre computação reconfigurável enfatizando alguns trabalhos relacionados ao estado da arte na área é apresentada no Capitulo 2. No Capitulo 3 é apresentado o ambiente proposto, onde são mostrados os objetivos e propriedades do sistema, com ênfase nos aspectos de implementação de *hardware* e *software*. No Capítulo 3 são apresentadas também as estruturas mínimas necessárias para o funcionamento do ambientes proposto. O Capítulo 4 trata da proposta e implementação usando lógica reconfigurável de controladores, onde são apresentados exemplos de implementação controladores na forma genérica RST através da utilização de linguagem gráfica e linguagem VHDL. No Capítulo 5 são mostrados exemplos práticos de aplicação do ambiente proposto: em uma cadeira de rodas e em dois robôs móveis experimentais. A validação experimental feita no Capítulo 5 demonstra de forma clara a flexibilidade do ambiente proposto na resolução de problemas associados com sistemas embarcados com diferentes níveis de complexidade. O Apêndice A apresenta uma revisão sobre os controladores GPC, controlador abordado como exemplo no Capítulo 4. O Apêndice B apresenta informações adicionais sobre o *hardware* desenvolvido. O Apêndice C apresenta o código dos blocos de *software* implementados. No Apêndice D é apresentado o código dos blocos implementados em lógica reconfigurável. No apêndice E é apresentada a relação de artigos apresentados em congressos e publicados em periódicos nacionais e estrangeiros.

Capítulo 2

Revisão da Literatura

2.1 Computação Reconfigurável

Sistemas reconfiguráveis são sistemas que apresentam a característica de, através da substituição de parte de seu *software* ou *hardware*, adaptarem-se a tarefas específicas. Eles têm por objetivo obter alto desempenho com baixo custo de produção sendo uma alternativa às máquinas de Von Neumman implementadas nos sistemas com microprocessadores. Sistemas de *software* reconfigurável são bastante comuns, sendo exemplos os sistemas embarcados em automóveis, em eletrodomésticos e mesmo em vídeo games. Nestes sistemas, a mudança de uma ROM ou de um CD-ROM leva à reconfiguração de funções responsáveis pela operação dos mesmos [Miyazaki 1998]. Sistemas de *hardware* reconfigurável são mais recentes e estão associados ao aparecimento das FPGA (*Field Programmable Gate Array*) na década de 90. Em [Page, 1996] é proposta a união em um único circuito integrado de um microprocessador e uma FPGA para atender dinamicamente novas aplicações. O advento de dispositivos como FPGAs mudou o ponto de balanço do compromisso entre flexibilidade e desempenho. Com as FPGA os objetivos de desempenho são mantidos com o aumento da flexibilidade [Kalra 2001].

Comparados aos sistemas de *software* reconfigurável os sistemas de *hardware* reconfigurável apresentam um maior potencial em termos de desempenho e adaptabilidade. Outras siglas estão associadas aos sistemas de hardware reconfigurável: CCM (*Custom*

Computing Machines) ou FCCM (*FPGA-based Custom Computing Machines*). As expressões computação reconfigurável e lógica reconfigurável também estão associadas a sistemas de *hardware* reconfigurável. Doravante, neste trabalho, as expressões sistema reconfigurável, computação reconfigurável e lógica configurável são relacionadas com sistemas de *hardware* reconfigurável.

Tradicionalmente, a execução de um algoritmo na computação convencional pode seguir dois métodos: utilizar uma tecnologia de *hardware*, a exemplo de ASICs (*Application Specific Integrated Circuits*) ou de placas de circuito impresso, ou utilizar microprocessadores programáveis por *software.* O primeiro método apresenta como vantagem a grande velocidade de execução, mas não é flexível a modificações posteriores. O segundo método, embora apresente alta flexibilidade para modificações, não é executado com a mesma velocidade dos algoritmos programados em *hardware* [Compton, 2002]. O fato dos microprocessadores executarem de forma seqüencial suas tarefas (máquina de Von Neumann) faz com que os algoritmos programados por *software* tenham tempos de processamento que não são aceitáveis para muitas aplicações [Pimentel e Le-Huy, 2000][Dido et al., 2002]. A computação reconfigurável tem por objetivo suprir a lacuna entre a solução por *software* e a solução por *hardware*, atingindo desempenhos muito superiores aos desempenhos obtidos por *software*, mas obtendo uma flexibilidade muito maior que a flexibilidade obtida com uma solução por *hardware* [Chen et al., 2000] [Compton, 2002] [Coric et al., 2002].

De outro modo, sistemas embarcados apresentam grande dinâmica tecnológica, ou seja, estão sujeitos a grandes variações tecnológicas em curto espaço de tempo, seja pela demanda por novas tarefas e desempenhos, seja pela disponibilidade de novas tecnologias de sensores e atuadores. Restrições de recursos em sistemas embarcados acentuam a necessidade de novas idéias de projeto. É comum a utilização de blocos de software funcionalmente testados visando reduzir o tempo de projeto de um sistema (API – *Application Program Interface*). Em *hardware* existem blocos funcionais que podem assumir a mesma função. Estes blocos podem ser combinados com outros circuitos para implementar um determinado algoritmo. Este aspecto modular permite facilitar a manutenção e a atualização de projetos [Compton,

2002][Renner et al., 2002] [Coric et al., 2002]. A utilização de IP-CORE (*Intellectual Property Cores*), permite a integração de soluções já desenvolvidas por diversos fornecedores para minimizar o tempo de projeto. Barramento PCI, interfaces de comunicação, e funções de processamento de sinal, como FFT (*Fast Fourier Transform*), codificação e decodificação de imagens digitais e mesmo Microprocessadores e DSPs são exemplos de IP-CORE disponíveis [Ito e Carro, 2000][Kean 2002] [Diniz e Park, 2002].

As PLDs (*Programmable Logic Devices*), também chamadas de EPLD (*Erasable Programmable Logic Devices*), ou ainda CPLD (*Complex Programmable Logic Devices*), e as FPGA *(Field Programmable Gate Array*) ou DFGA ((*Dinamically Field Programmable Gate Array*) são dispositivos que permitem a execução de algoritmos diretamente em *hardware*. Com estes dispositivos programáveis é possível executar os algoritmos explorando o paralelismo inerente da solução por hardware, executando-os muito mais rápido do que se os mesmos algoritmos fossem executados de forma seqüencial por microcontroladores ou por DSPs, sujeitos ao modelo de Von Neumann. As PLDs e FPGAs apresentam diferenças quanto à aplicação e quanto à estrutura interna. As FPGA são geralmente aplicadas em sistema com menor sensibilidade ao custo e de maior complexidade. [Miyazaki 1998] [Pimentel e Le-Huy, 2000].

Em [Miyazaki, 1998] é apresentada uma notação para classificação dos tipos de dispositivos lógicos reconfiguráveis:

- Dispositivos de lógica configurável: São dispositivos lógicos que podem ser “customizados” uma única vez.

- Dispositivos de lógica reconfigurável: São dispositivos que podem ser customizados mais de uma vez. Estes dispositivos adotam tecnologias EPROM, EEPROM ou FLASH e podem ser reprogramados após serem montados em uma placa de circuito impresso.

- Dispositivos de lógica dinamicamente reconfigurável: São dispositivos que permitem sua

reprogramação durante a operação, mesmo após a montagem em uma placa de circuito impresso. Esta propriedade é chamada de reconfiguração *in-circuit.*

- Dispositivos de interconexão dinamicamente reconfigurável: É a expressão para dispositivos interconectados que podem ser programados por conexões pino a pino após a montagem em placa de circuito impresso.

- Dispositivos de lógica virtual: Este tipo de dispositivos dinamicamente reconfiguráveis apresenta uma capacidade parcial de reconfiguração. Apenas parte do dispositivo pode ser reprogramado enquanto a outra parte do dispositivo executa alguma função definida. Em outras palavras, diferentes circuitos lógicos podem partilhar ao longo do tempo uma mesma parte do dispositivo. São também chamados de dispositivos de lógica adaptativa ou de dispositivos de lógica compartilhada.

Entre as vantagens que se pode apresentar estão o paralelismo, velocidade [Aporntewan e Chongstitvatana, 2001], capacidade de atualização, padronização de plataformas, amortização do custo de desenvolvimento e alto desempenho. Entre as desvantagens estão em ambientes de desenvolvimento pobres se comparados com os ambientes de desenvolvimento de processadores.

Paralelamente ao desenvolvimento de dispositivos que permitiram a implementação da computação reconfigurável, ambientes de projeto, depuração, simulação e testes foram desenvolvidos pelos diversos fabricantes. Estes ambientes permitem a criação de módulos desenvolvidos com linguagens de alto nível de abstração, chamadas linguagens de descrição de hardware, a exemplo de VHDL (*VHSIC Hardware Descriptive Language*) e AHDL (*Altera Hardware Descriptive Language*). A linguagem VHDL é a linguagem padrão IEEE para descrição de hardware, a qual provê um ambiente integrado de projeto possibilitando projeto, simulação, teste e documentação de circuitos digitais. De outro modo, é possível implementar blocos com representações de mais baixo nível de abstração, a exemplo de esquemáticos (linguagem gráfica). A integração de blocos criados com diferentes linguagens permite a

criação de projetos de forma bastante flexível, facilitando a interação entre os membros de uma equipe de trabalho. A síntese dos circuitos lógicos em PLD é feita através de sistemas CAD (*Computer Aided Design*), permitindo a utilização de diferentes interfaces de projeto. Diversos fabricantes de sistemas de desenvolvimento e de dispositivos lógicos reconfiguráveis disputam o mercado mundial. Entre eles destacam-se Altera [Altera, 2002], Xilinx [Xilinx, 2002], Atmel [Atmel, 2002], Triscend [Triscend, 2002] e Actel [Actel, 2002]. Outros fabricantes de *software* apresentam produtos CAD de apoio ao projeto: Mentor Graphics [Mentor, 2002], Synopsys [Synopsys, 2002] e Accolade [Accolade, 2002].

Algumas vantagens da metodologia de projeto baseado em PLD e VHDL são:

- Maior facilidade na interação de uma equipe de trabalho.
- Diminuição dos tempos de projeto, de eventuais correções ou ainda de integração de novas versões.
- Opera de forma paralela. Esta é a característica mais vantajosa em relação aos microprocessadores. Não obedece ao modelo de Von Neumann.
- Permite o desenvolvimento modular e hierárquico de um projeto.
- A existência diversas interfaces de desenvolvimento, baseadas em linguagens gráficas (esquemáticos) ou em linguagens de descrição de hardware (VHDL, AHDL).
- Permite as abordagens de projeto *top-down* e *bottom-up*.
- Permite a utilização de funções pré-testadas (IP-CORE), diminuindo o tempo de projeto.

Inúmeras aplicações usando computação reconfigurável podem ser apresentadas nas mais diversas áreas: comunicação, controle, manipulação matemática, tolerância a falha, criptografia .

No artigo [Woerner e Lehman, 1995] é apresentada a aplicação de FPGA no projeto do veículo *Pathfinder*, enviado ao planeta Marte pela NASA. Um sistema de controle de inversores de potência, que utiliza FPGA, é apresentado em [De Pablo et al., 1997]. Em [Empringham et al., 2000] é aplicado uma FPGA para implementar o controle de um motor de indução. Outra implementação de um controlador de motor de indução usando linguagem VHDL é explorado em [Cirstea et al., 2000] e em [Cirstea e Dinu, 2001]. Um controle adaptativo *fuzzy* de motor utilizando linguagem VHDL é apresentado em [Changuel et al., 1996]. Em um projeto de controle digital de cadeira de rodas, o trabalho [Chen et al., 2000] emprega FPGA. O artigo [Coric et al., 2002] implementa algoritmos em hardware para acelerar o processamento de imagens médicas, a exemplo de tomografia computadorizada. Em [Endemano, 2002] é apresentada a implementação de um modelo de simulação de um tipo particular de motor usando VHDL. Em [Jeon et al., 2002] é implementado um algoritmo para aceleração de desaceleração de motores usados em CNC e robôs industriais. Em [Baraza et al., 2002] é implementado um sistema de análise de tolerância à falhas baseado em ferramentas implementadas em VHDL. Em [Dido et al., 2002] são implementados algoritmos de processamento de sinal de vídeo de alta resolução (HDTV) em FPGA. As FPGAs são utilizadas como auxiliares em proposta de projeto de sistemas de navegação por satélite em [Thor e Akos, 2002]. A implementação de algoritmos em FPGA de sinais para reconhecimento de voz é feita em [Varga et al., 2001]. O trabalho [Hasuko et al., 2001] usa FPGA na implementação de um sistema de monitoração de ambiente radioativo.

Diversos trabalhos associados à implementação de funções matemáticas, controladores digitais, filtros e controladores avançados podem ser citados: A implementação de uma PLL (*Phase-locked loop*) digital é reportada em [Kobayashi e Haratsu, 1995]. Os trabalhos [Kollig et al., 1996] e [Kollig et al., 1997] apresentam considerações sobre o projeto de sistemas descritos por algoritmos matemáticos ou por equações a diferenças, usando VHDL. O artigo [Klotchkov e Pedersen, 1996] apresenta diferentes maneiras de implementar funções aritméticas em FPGAs. Através de FPGAs é implementado o algoritmo de um PID digital no artigo [Samet et al., 1998] e é proposto um algoritmo *fuzzy* para a sintonia de um PID em [Hu e Li, 1996]. Um controlador de motor AC usando FPGA é apresentado em [Kharrat et al., 1998]. Exemplos de implementação

através de FPGA e VHDL de controladores por espaço de estados são mostrado em [Garbergs e Sohlberg, 1996] e [Garbergs e Sohlberg, 1998]. Em [Hwang e Han, 1999] é apresentado um filtro preditor de erro em lógica reconfigurável. O artigo [Fanucci, 2002] descreve a implementação de algoritmo FFT com a utilização de lógica reconfigurável. A modelagem, simulação e implementação de um controlador usando lógica reconfigurável é relatada em [Wegrzyn et al., 1998]. O trabalho [Khriji et al., 1999] desenvolve um hardware dedicado que implementa filtros usados em processamento de imagem. Em [Harkin et al., 2001] e em [Aporntewan e Chongstitvatana, 2001] são implementados algoritmos genéticos através de uma abordagem de *software* e *hardware* com utilização de dispositivos lógicos reprogramáveis. O artigo [Daly e Marnane, 2002] apresenta um implementados em FPGA. O trabalho [Swaminathan et al., 2002] implementa um algoritmo de código corretor de erro em sistemas de comunicação digital usando hardware reconfigurável. [Pimentel e Le-Huy, 2000] apresenta exemplos de um controlador PI (proporcional e integral), um gerador PWM (modulação por largura de pulso) e um conversor digital-analógico. Uma solução alternativa para projeto de interfaces em sistemas de controle baseados em microprocessadores é descrita em [Vargas et al., 2001]. O trabalho [Valls et al., 1997] apresenta o conceito de computação digito-serial com implementação de filtros FIR (*Finite Impulse Response*) nas formas inversa e canônica usando FPGA da Altera. [Rathna et al., 1994] apresenta uma metodologia de síntese de arquiteturas para implementação de filtros IIR (*Infinite Impulse Response*) em FPGAs e o artigo de [Peiró et al., 1999] implementa com EPLD uma série de filtros FIR. Em [Franco et al., 2000] é apresentada a implementação de um filtro adaptativo não-linear usando FPGA. No artigo [Monteiro et al., 2001] é implementado um código CRC (*Cyclic Redundancy Checking*) para uso em sistemas de telecomunicação com o uso de FPGA. O trabalho [Köster e Teich, 2002] introduz o conceito de máquinas de estado finitas implementadas em hardware reconfigurável. Uma aplicação de redes neurais, usando VHDL é relatada em [Acosta e Tosini, 2001]. Em [Yamaguchi et al., 2000] é apresentada uma técnica baseada em computação reconfigurável para processamento criptográfico.

Outros trabalhos apresentam metodologias e técnicas de projeto com computação reconfigurável. Em [Renner et al., 2002] é apresentada uma metodologia de projeto de controladores baseados em *hardware* e *software* para aplicações mecatrônicas. As possíveis

arquiteturas encontradas em computação reconfigurável são exploradas em [Page 1996]. O trabalho [Pimentel e Le-Huy, 2002] apresenta as técnicas mais recentes no projeto com lógica reconfigurável. Uma descrição de um compilador C para sistemas com microcontroladores que tenham unidades funcionais reconfiguráveis é feita em [Ye et al., 2000]. Em [Snider et al., 2001] é explorado um modelo para utilizar o paralelismo em dispositivos reconfiguráveis. Uma técnica de implementação de controladores em Paralelo usando Redes de Petri e com aplicação em dispositivos reconfiguráveis é apresentada em [Kozlowski et al., 1995]. Os trabalhos [Yasunaga et al., 2000] e [Harkin et al., 2001] apresentam metodologias de projeto com sistemas reconfiguráveis. Um esquema de criptografia para utilização em funções IP-CORE é apresentados em [Kean, 2002].

2.2 Ferramenta de Projeto MAX+PLUS II ™

Os blocos desenvolvidos para serem programados nas PLDs usadas no ambiente proposto foram desenvolvidos, testados e simulados com a ferramenta de projeto Altera MAX+PLUS II™ [Altera, 2002]. Esta ferramenta apresenta interfaces de projeto em linguagem gráfica, linguagem VHDL e linguagem AHDL. A escolha deste fornecedor deve-se a um programa de incentivo educacional (*Altera University Program*) da empresa Altera utilizado pela UNICAMP em diversas Faculdades, inclusive na Faculdade de Engenharia Mecânica, nos cursos de pós-graduação e graduação que permite a utilização destas ferramentas de projeto com custo relativamente baixo.

São apresentadas algumas informações sobre esta ferramenta, visando o melhor entendimento do texto da tese.

A ferramenta MAX+PLUS II™ permite filosofia de projeto hierárquico, onde sub-blocos podem ser detalhados a partir de blocos hierarquicamente superiores. Estes blocos e sub-blocos podem ser projetados utilizando-se de diversas linguagens descritivas de *hardware*.

A Figura 1 mostra uma tela da ferramenta na qual a visão geral de um projeto é

apresentada em linguagem gráfica (esquemático). Neste projeto ("encoder"), podem ser observados dois sub-blocos: "enc16" e "jan_enc". Doravante, referências a nomes de blocos representados em linguagem gráfica, VHDL ou AHDL são feitas colocando-se os nomes dos blocos entre aspas. Referências a nomes de sinais são feitas em negrito a exemplo do sinal **clock** presente na Figura 1.

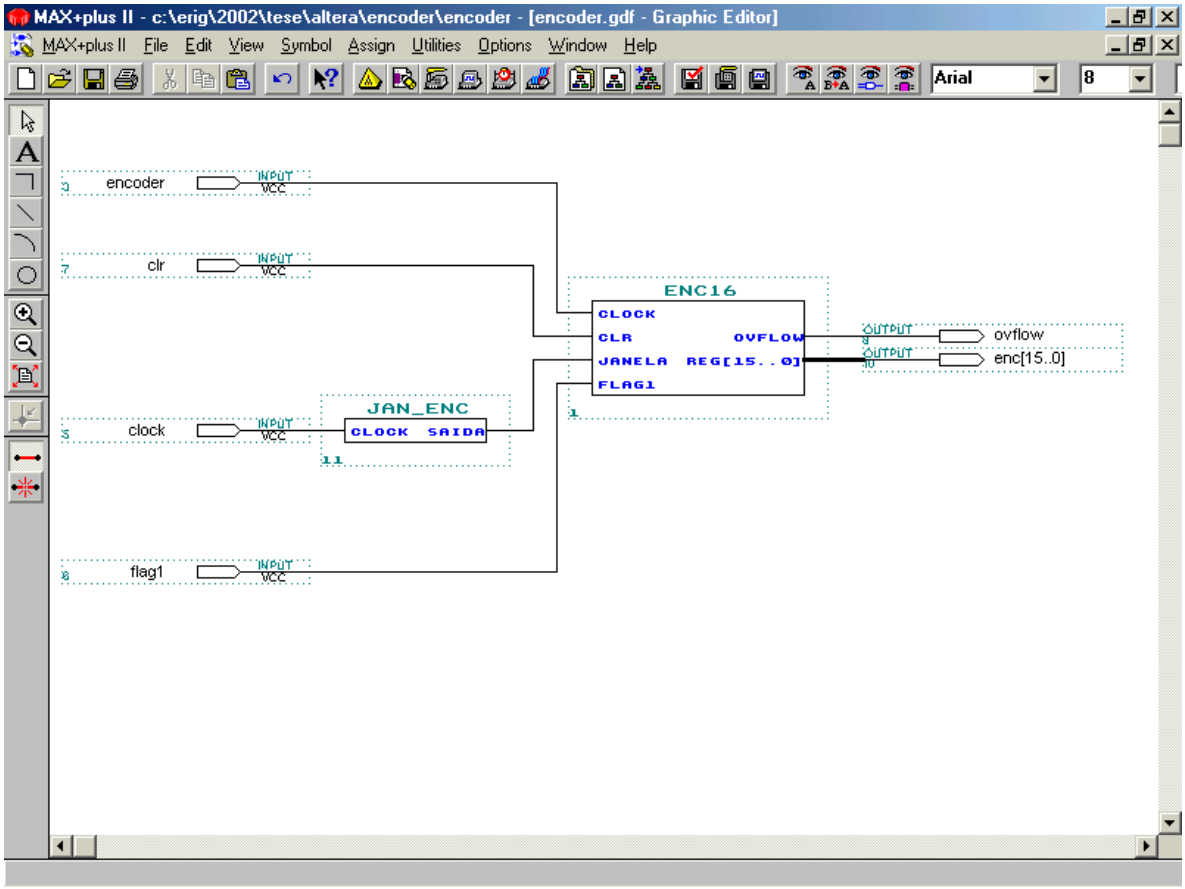


Figura 1 - Tela da ferramenta MAX+PLUS IITM representando um projeto em linguagem gráfica (esquemático).

O sub-bloco "enc16", visto parcialmente na Figura 2, é projetado em linguagem VHDL. É comum nos diversos projetos desenvolvidos neste trabalho, a utilização de linguagem gráfica em alguns blocos e de linguagem VHDL em outros blocos. Isto se deve à adequação de uma ou outra linguagem a uma determinada função que se deseja implementar. Geralmente, por motivo

de clareza na documentação, considerando uma abordagem *top-down* do projeto, é utilizada a linguagem gráfica para os blocos de mais alto nível.

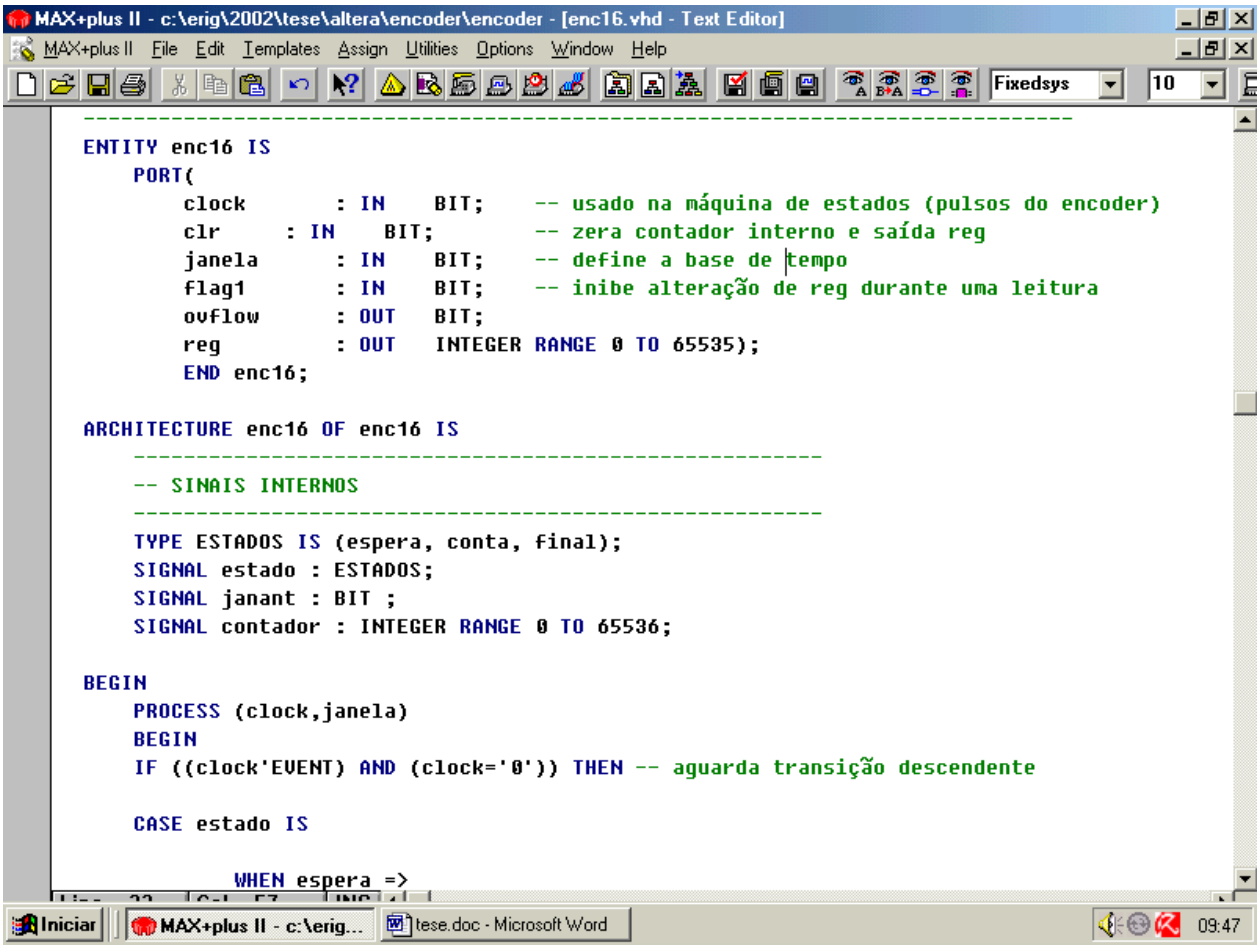


Figura 2- Tela da ferramenta MAX+PLUS IITM representando um sub bloco de projeto em linguagem VHDL.

Outra característica da ferramenta MAX+PLUS II^{TM} é a possibilidade de realizar a simulação do projeto a ser implementado. Isto permite a rápida depuração do projeto, abreviando o tempo do mesmo. A Figura 3 apresenta a simulação do bloco “encoder”.

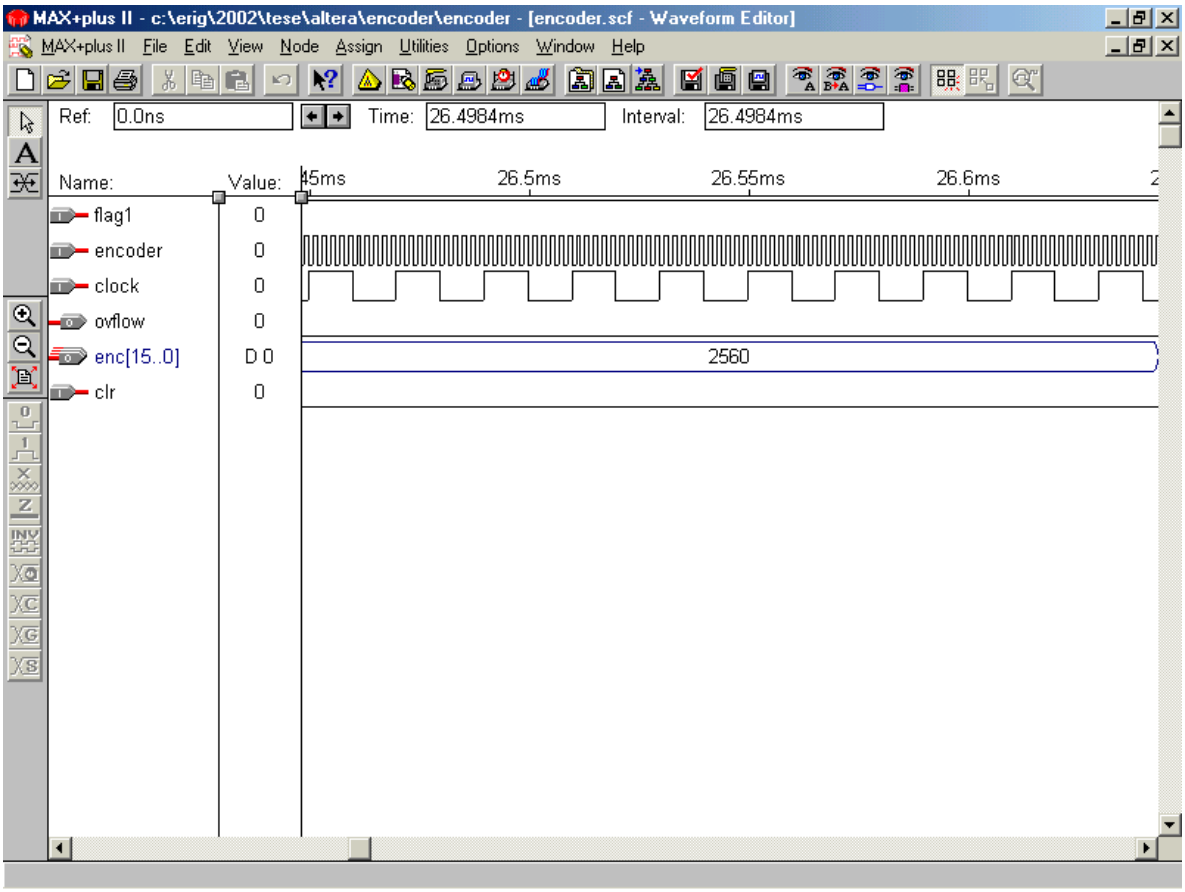


Figura 3 - Tela da ferramenta MAX+PLUS II™ representando uma simulação de projeto.

Além da Altera MAX+PLUS II™ outras ferramentas de projeto estão disponíveis no mercado. Não se busca neste trabalho fazer comparações entre ferramentas de diferentes fabricantes nem julgar seus méritos e defeitos. A escolha desta ferramenta, em particular, está associada à conveniência técnica e econômica no período de desenvolvimento desta tese. Contudo, os blocos em lógica reconfigurável aqui desenvolvidos podem ser reproduzidos em ferramentas de outros fabricantes, se assim se fizer necessário.

No Capítulo 3 é apresentada uma visão geral do ambiente proposto, onde a ferramenta MAX+PLUS II™ é utilizada.

Capítulo 3

Ambiente Proposto – Visão Geral

3.1 Introdução

Este capítulo apresenta o ambiente proposto, descrevendo os objetivos e propriedades do mesmo, bem como os elementos de hardware e software básicos para implementação em um protótipo de sistema embarcado.

O ambiente proposto é um conjunto de módulos de *hardware* e *software*, implementados com ênfase na utilização de lógica reconfigurável, integrados para dar suporte ao desenvolvimento de protótipos de sistemas embarcados. A ênfase na utilização de lógica reconfigurável na implementação deste ambiente apresenta como justificativas:

- Facilidade e rapidez na execução de projeto – Com a utilização de uma série de ferramentas fornecidas pelo mercado é possível a execução, o teste, a simulação e a depuração de um projeto com bastante eficiência. A divisão do projeto em pequenos blocos funcionais permite a melhor distribuição das tarefas entre os membros de uma equipe de trabalho. Blocos previamente desenvolvidos podem ser aproveitados em novos projetos, minimizando o tempo de execução destes.

- Minimização de custo – O desenvolvimento de um novo hardware para atender exigências de um novo projeto ou a novas exigências de um projeto antigo, implica em custos que podem

ser evitados pela utilização de lógica reconfigurável.

- Fácil expansão – Novos módulos podem ser gradativamente adicionados ao sistema original permitindo que novas funcionalidades sejam implementadas sem a necessidade de realizar um novo projeto de hardware.

- Velocidade de operação – Além da característica de operar em paralelo, algoritmos implementados com lógica reconfigurável são tipicamente mais rápidos que algoritmos implementados com lógica convencional (execução de linhas de código através de um processador).

- Domínio tecnológico - Lógica reconfigurável é apresentada como solução para inúmeros problemas práticos de engenharia. Dentro desta realidade é justificável o esforço para dominar esta área de conhecimento, buscando suas virtudes e limitações.

São objetivos do ambiente:

- Flexibilidade – O ambiente deve ser capaz de adaptar-se facilmente à aplicação desejada. Um novo conjunto de requisitos deve ser facilmente atendido sem implicar em novos projetos de *hardware*.

- Expansibilidade – O ambiente deve ser de fácil expansão, permitindo que novas funcionalidades sejam adicionadas a um sistema pré-existente.

- Transparência – O ambiente deve ser totalmente aberto, permitindo o acesso de novos usuários a todos os aspectos do mesmo. Isto vem de encontro às diferentes necessidades acadêmicas de graduação e pós-graduação a que este ambiente visa prestar suporte.

- Adequação à tarefa – O ambiente deve poder ser configurado para resolver problemas de maior ou menor complexidade. Isto permite a utilização adequada de recursos para resolver um determinado problema, evitando desperdícios.

- Baixo custo – São considerados aspectos que visam a minimização do custo no desenvolvimento de um protótipo de sistema embarcado. São utilizadas ferramentas de projeto de baixo custo ou sem custo associado. A própria utilização de lógica reconfigurável permite a minimização do custo total de um projeto pelas razões já apresentadas.

- Capacidade de comando e monitoração remota – Em muitas aplicações de sistemas embarcados a monitoração e modificação de parâmetros internos aos sistemas pode ser de grande utilidade, permitindo um menor tempo de projeto e depuração de erros ou atendendo a pré-requisitos deste projeto.

Em função dos objetivos apresentados é proposto um ambiente de arquitetura hierárquica, onde, através de diferentes níveis de controle dispostos hierarquicamente é possível atender aos objetivos de flexibilidade, expansibilidade, transparência, baixo custo, capacidade de comando, monitoração remota e adequação à tarefa.

3.2 Arquitetura de Controle Hierárquica

O ambiente proposto é baseado em uma arquitetura hierárquica e aberta, representada no diagrama de blocos da Figura 4.

Os diferentes blocos da figura são implementados em *software* ou *hardware*. A arquitetura é organizada em diversos blocos independentes, associados a interfaces de sensores e atuadores, comunicação e tratamento de memória. Quando limitações de recursos são atingidas, novos blocos podem ser adicionados, permitindo uma configuração adequada do dispositivo embarcado para cada tarefa.

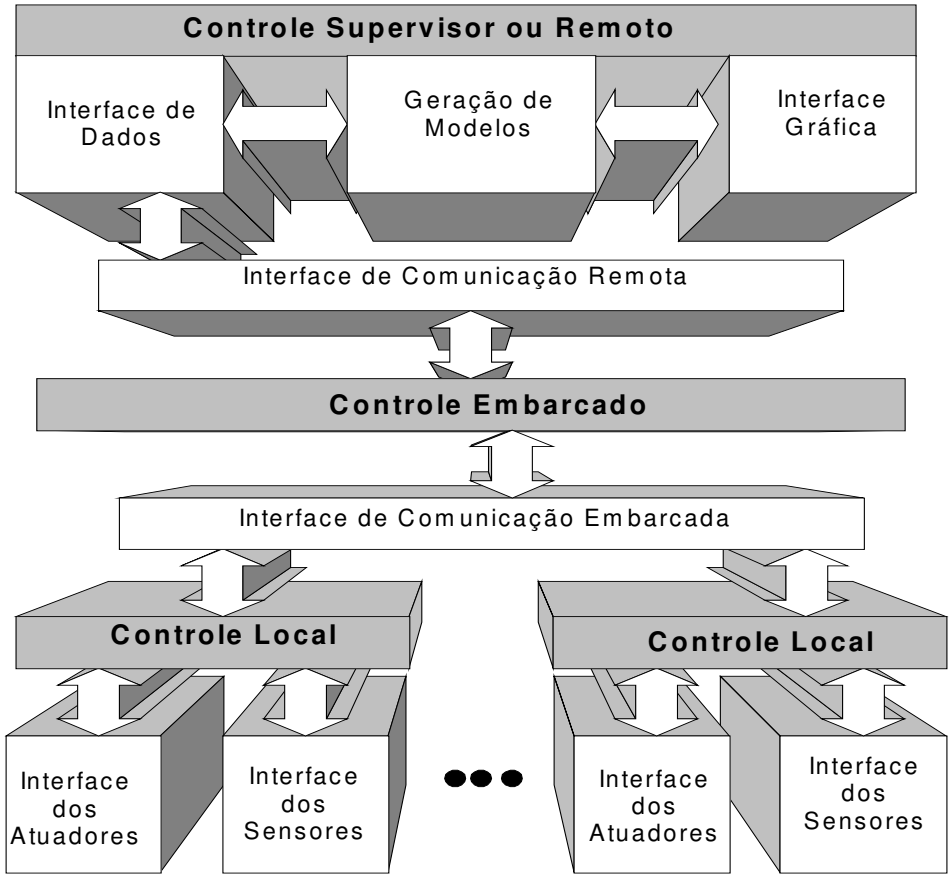


Figura 4 -Arquitetura hierárquica e aberta proposta.

A proposta geral do sistema pode ser visualizada como um sistema de vários níveis, organizado em três níveis hierárquicos de controle, que podem ser assim descritos:

- **Controle supervisor ou remoto**: É o nível de controle mais alto, onde a supervisão de um ou mais sistemas embarcados pode ser executada através de estratégias de controle globais. Este nível também permite gerenciar tarefas específicas, estabelecer correções nestas tarefas em função de informações obtidas da fusão de sensores, ou ainda modificar parâmetros de operação de um nível de controle inferior. Em um protótipo, o controle supervisório é também usado para facilitar a depuração do sistema, fornecendo parâmetros de operação do mesmo, facilitando a localização de erros no projeto.

- **Controle embarcado**: Neste nível o controle é processado pelo próprio sistema embarcado.

As estratégias de controle, neste nível, permitem a tomada de decisão com ou sem interferência do nível de controle supervisório. No caso de não haver comunicação entre o nível de controle supervisor e o nível de controle embarcado, este efetuará ações com base nas informações obtidas dos sensores e dos parâmetros previamente armazenados em memória. A transferência de dados entre este nível e o nível de controle supervisório é implementada através do uso de comunicação de dados serial. Para tanto, é implementada uma codificação serial que permite a utilização de diversos meios físicos, a exemplo de rádio freqüência, infravermelho, fibra ótica ou fio metálico. As tarefas associadas a este nível podem ser executadas por rotinas implementadas em lógica reconfigurável ou combinação de microprocessadores com lógica reconfigurável. Neste nível são implementadas eventuais estratégias de controle, a exemplo de controladores PID ou GPC.

- **Controle local**: Este nível está associado com as interfaces dos sensores e atuadores. A existência deste nível justifica-se pela freqüente necessidade de propiciar processamento complexo aos sensores e atuadores de sistemas embarcados. Em lógica reconfigurável, este processamento pode ser feito de forma paralela, permitindo que requisitos de tempo de execução do processamento dos sensores e atuadores sejam atendidos de forma mais fácil que utilizando computação convencional.

3.3 Comunicação Remota

Um dos aspectos necessários à implementação da arquitetura proposta é a existência de um enlace de comunicação remota ou operação remota entre o nível de controle supervisor ou remoto e o nível de controle embarcado, ou seja, a capacidade de realizar operações remotas sobre os sistemas embarcados considerados. Esta funcionalidade pode ser implementada através de rádio freqüência (RF), infravermelho (IR), fibra ótica ou fio metálico. Na Figura 5 é apresentado um exemplo onde é utilizado um enlace de RF para implementar a comunicação remota entre o nível de controle supervisor e o nível de controle embarcado.

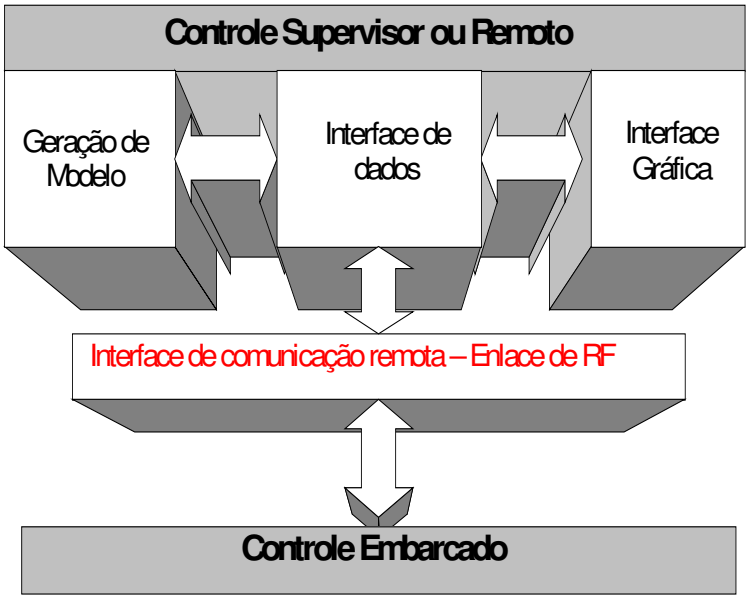


Figura 5 – Diagrama de blocos mostrando a interligação da camada de controle supervisor com a camada de controle embarcado através de um enlace de rádio freqüência.

A operação remota tem por função a comunicação bi-direcional entre o sistema embarcado (nível de controle embarcado) e um computador remoto (nível de controle supervisor). As seguintes funcionalidades são implementadas pela operação remota:

- Telecomando – Permite a operação remota do sistema embarcado, enviando comandos relacionados com a operação do sistema embarcado, determinando, no caso de um sistema embarcado móvel, que tipo de trajeto o mesmo vai executar, com qual velocidade, ou ainda, a execução de uma tarefa pré-definida.

- Monitoração de parâmetros – Permite o acompanhamento de variáveis associadas à operação do sistema embarcado. A corrente dos motores, a tensão das baterias, o estado dos sensores e a posição, a velocidade e a aceleração de um sistema móvel podem ser apresentados a um operador remoto ou podem ser usados para tomada de decisão pelo nível de controle supervisor.

- Alarme remoto – Permite que um operador remoto seja informado sobre algum evento

significativo, como por exemplo, carga muito baixa das baterias. Situações de alarme em sistemas críticos podem ser avaliadas, como no caso da operação com sistemas médicos e hospitalares.

A implementação da comunicação remota é feita por módulos construídos em lógica reconfigurável. O primeiro módulo é conectado através da porta paralela de um computador PC e faz parte do nível de controle supervisor. Este módulo é chamado de módulo de comunicação remoto e é detalhado na Seção 3.4 que descreve o nível de controle supervisor. O outro módulo, localizado no sistema embarcado, é chamado de módulo de comunicação local e faz parte do nível de controle embarcado. É detalhado na Seção 3.5 que trata do nível de controle embarcado. A Figura 6 ilustra exemplos de operação remota, usando enlace de RF, em dois sistemas embarcados móveis que serão explorados no capítulo 5 - análise experimental: uma cadeira de rodas e um robô móvel experimental.

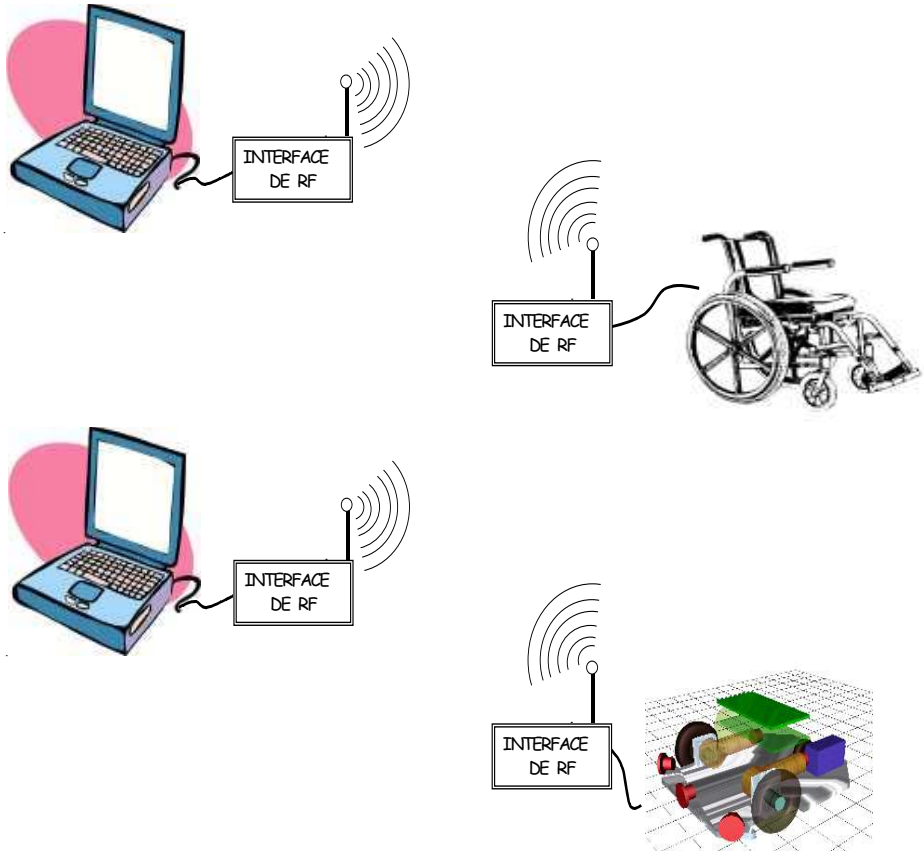


Figura 6– Exemplos de operações remotas exploradas no capítulo 5.

3.4 Nível de Controle Supervisor ou Remoto

O nível de controle supervisor ou remoto é construído como um conjunto de blocos de *software* e *hardware* que podem ser vistos de forma simplificada no diagrama de blocos da Figura 7. Os blocos de interface gráfica e interface de dados são implementados em *software*. Implementações de *hardware* e em lógica reconfigurável são feitas no bloco de *hardware* de suporte. O bloco de *hardware* de suporte é implementado em uma ou mais placas de circuito impresso. Estas placas de circuito impresso são utilizadas na implementação dos três níveis de controle e são descritas na seção 3.7.

As seguintes funcionalidades são implementadas neste nível:

- Comunicação serial com o sistema embarcado controlado (transmissão e recepção de dados e comandos).

- Protocolo de comunicação com o dispositivo controlado. Este protocolo apresenta rotinas de proteção contra erros de transmissão ou recepção.

- Interface de visualização de informações e envio de comandos e parâmetros para o dispositivo controlado em ambiente Windows e usando a porta paralela do PC.

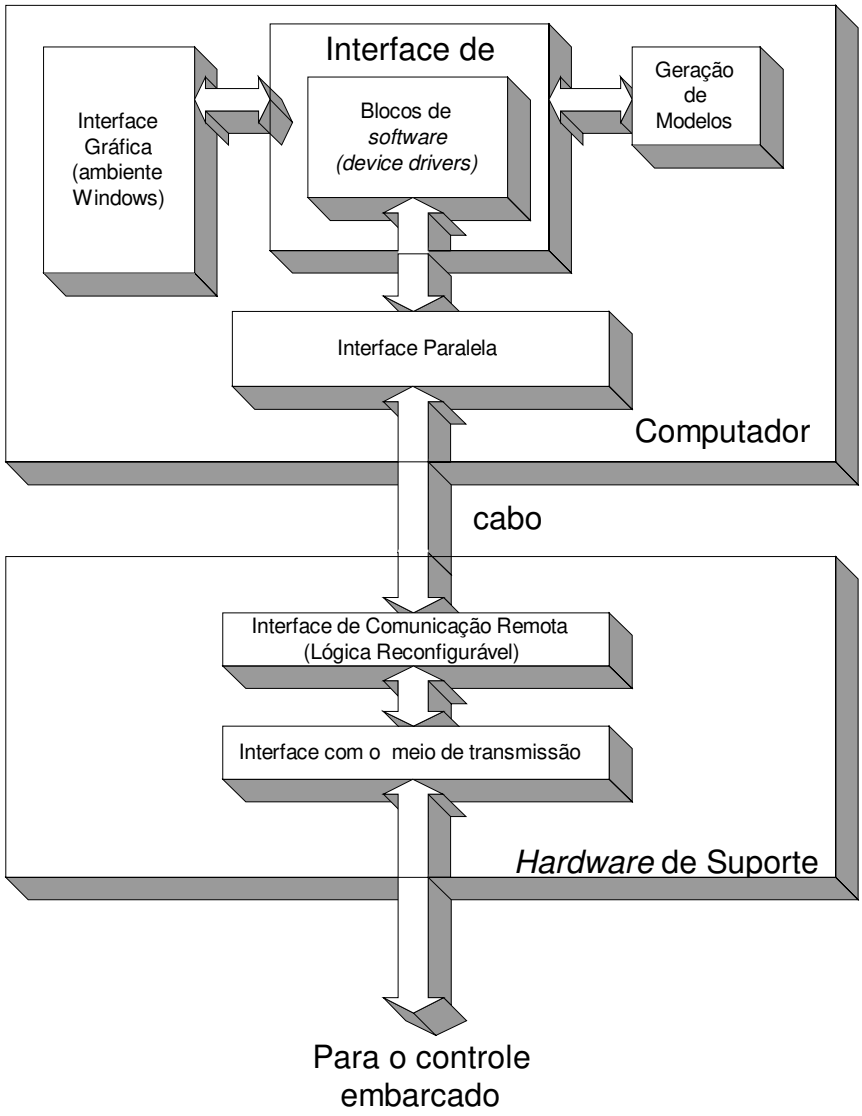


Figura 7 – Diagrama de blocos detalhando a interface de comunicação remota.

3.4.1 Interface Gráfica

O bloco interface gráfica é implementado para operar no ambiente Windows, permitindo a visualização de informações e a entrada de dados. Esta interface deve sofrer modificações em função do dispositivo que está sendo controlado, uma vez que os dados a serem considerados mudam em função da natureza do sistema embarcado considerado. As interfaces gráficas desenvolvidas neste trabalho são implementadas usando a ferramenta Visual Basic 6.0. A escolha desta ferramenta deve-se à facilidade e rapidez com que é possível implementar uma nova interface gráfica, permitindo gerar elementos gráficos de visualização e comando com um mínimo de trabalho. Contudo, consideradas as limitações das rotinas de *software* implementados na interface de dados (*device drivers*), novas interfaces gráficas podem ser implementadas usando linguagens como Visual C ou C++. Um exemplo de aplicação pode ser visto na Figura 8, onde um conjunto simples de comandos permite a realização de movimentos de um robô móvel.

A interface gráfica tem as seguintes funcionalidades:

- Visualização de informações sobre sensores e atuadores, registradores internos de controle e configuração existentes no dispositivo controlado. São exemplos de informações a serem visualizadas: velocidade de operação de um motor, valores armazenados em registradores, contadores associados a *encoders*, valores de posição e ângulo, medidas de corrente, temperatura ou pressão.

- Envio de comandos de operação. São exemplos de comandos a serem enviados: enviar dados, iniciar movimento de um atuador, ligar ou desligar sensores ou atuadores, calibrar sensores, modificar parâmetros de operação.

- Geração de alarmes de operação. São exemplos de alarmes de operação: perda de comunicação com o dispositivo controlado, erro de comunicação, parâmetros fora de limites estabelecidos (bateria baixa, posição ou velocidade errada, por exemplo).

Figura 8- Exemplo de interface com o usuário utilizada para controlar remotamente os movimentos de um robô móvel.

3.4.2 Interface de Dados

O bloco de interface de dados implementa as rotinas (*device drivers*) que manipulam escrita e leitura dos registradores internos da interface de comunicação remota. Estas rotinas são caracterizadas por utilizarem *dll* (*dinamic library link*) criadas para operações de escrita e leitura em endereços de I/O dentro do ambiente Windows. Qualquer nova interface homem máquina criada fará uso destes *device drivers* para comunicação com a lógica reconfigurável e o *hardware* associado.

A camada dos *device drivers* que implementa as rotinas necessárias para acessar a interface paralela no modo EPP (*Enhanced Parallel Port*) implica em operações de escrita e leitura de I/O.

Em ambiente *Windows* isto não pode ser feito diretamente devido ao modo como o acesso aos endereços de I/O é gerenciado neste ambiente. São implementadas as funções:

- *Function reset_rx*() – gera *reset* no bloco "dec_rx", bloco de decodificação de recepção, liberando este bloco para uma nova recepção.

- *Function start_bit*() – inicia envio de *byte* bloco de transmissão "tx8".

- *Function config_bit_one*(*nbit As Integer*) – "seta" em nível um o bit definido pela variável 'nbit' do registro de controle.

- *Function config_bit_zero*(*nbit As Integer*) – "seta" em nível zero o bit definido pela variável 'nbit' do registro de controle.

- *Function write_data*(*x As Integer*) - escreve *byte* da variável 'x' no registro de saída.

- *Function read_data*() - lê o conteúdo do registro de saída.

- *Function read_config*() - lê o conteúdo do registro de configuração.

- *Function write_caracter*(*x As Integer*) - escreve *byte* da variável 'x' no registro de saída e o envia via bloco de transmissão "tx8".

- *Function write_register*(*add As Integer, data As Integer*) - escreve *byte* da variável 'data' no registro definido pelo endereço 'add'.

- *Function read_register*(*add As Integer*) - lê o conteúdo do registro definido pelo endereço 'add'.

- *Function read_status*() - lê o conteúdo do registro de status.

3.4.3 Geração de Modelos

O bloco geração de modelos é construído visando à tomada de decisões do controle supervisor em função de dados fornecidos pelo sistema embarcado. Este bloco é desenvolvido em função da natureza do sistema embarcado projetado e de suas especificações funcionais. Um exemplo de implementação do bloco de geração de modelos é a mudança de parâmetros de operação de uma interface de potência de uma cadeira de rodas quando a variável que indica a carga da bateria atinge um determinado valor. Outro exemplo é a mudança periódica de parâmetros de sintonia de um controlador em um laço de controle do motor de um robô móvel para avaliação de desempenho.

3.4.4 Interface Paralela

A interface de comunicação remota é conectada a um computador PC através de uma porta ou interface paralela configurada para operar no modo EPP (*Enhanced Parallel Port*). Este modo implementa um protocolo composto de quatro ciclos, representados na Figura 9. O modo de operação EPP permite operações de escrita e leitura bidirecionais de 8 bits.

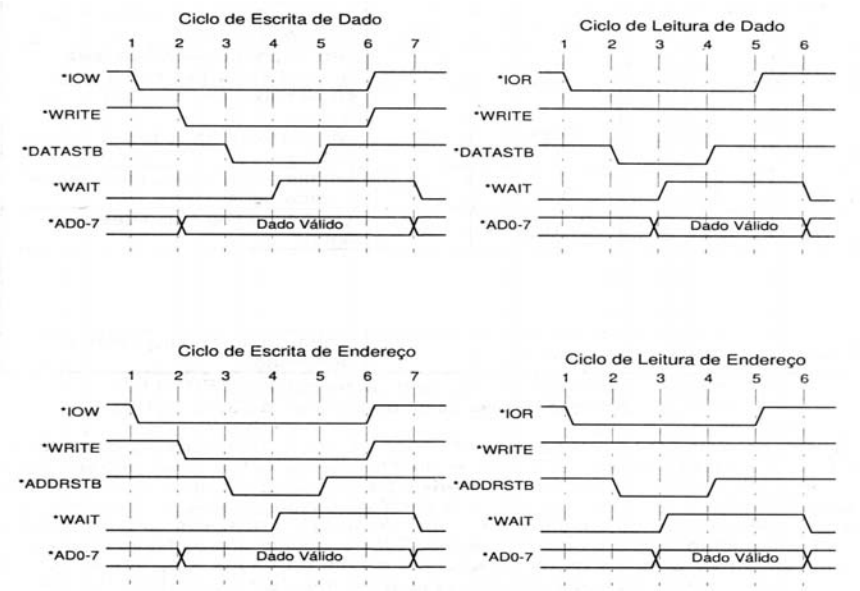


Figura 9– Ciclos de escrita e leitura implementados pelo protocolo de comunicação da interface paralela operando em modo EPP.

Os pinos utilizados pela interface paralela são apresentados na Tabela 1.

Tabela 1– Pinos da interface paralela usados pelo módulo de comunicação remoto.

Pino da porta paralela	Nome	Sentido	Descrição
1	write	out	Operação de escrita ou leitura
14	datastb	out	Transferência de dados
17	addrstb	out	Transferência de endereços
11	wait	in	Permissão para início e fim do ciclo
2	AD0	in/out	Bit de dado ou endereço
3	AD1	in/out	Bit de dado ou endereço
4	AD2	in/out	Bit de dado ou endereço
5	AD3	in/out	Bit de dado ou endereço
6	AD4	in/out	Bit de dado ou endereço
7	AD5	in/out	Bit de dado ou endereço
8	AD6	in/out	Bit de dado ou endereço
9	AD7	in/out	Bit de dado ou endereço
12	DV	in	Dado válido recebido
18 e 19	GND	in/out	Linhas de terra

Através da interface paralela são passados os dados a serem enviados ou recebidos pela interface de comunicação remota.

3.4.5 Interface de Comunicação Remota

A interface de comunicação remota tem por função o tratamento dos sinais seriais enviados e recebidos do nível de controle embarcado. Para tanto é utilizada uma codificação serial particular:

- O bit ‘0’ é transmitido por um pulso de largura 50μs em nível alto e 50μs em nível baixo.

- O bit ‘1’ é transmitido por um pulso de largura 100μs em nível alto e 100μs em nível baixo.

- Um pulso de 200μs em nível alto é usado para indicar início e término de transmissão.

A Figura 10 apresenta um exemplo desta codificação para a transmissão da seqüência "10101010".

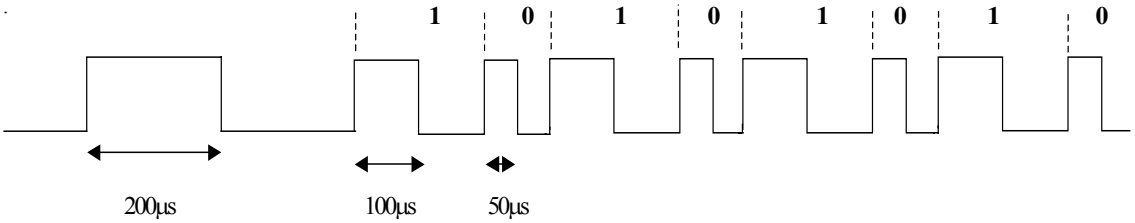


Figura 10– Exemplo de transmissão da seqüência "10101010" segundo a codificação proposta.

Diversos blocos da interface de comunicação remota são implementados em lógica reconfigurável. Na Figura 11 são representados estes blocos, destacando-se a interligação dos mesmos com a interface paralela é um dos possíveis meios de comunicação usados: um módulo de rádio – freqüência.

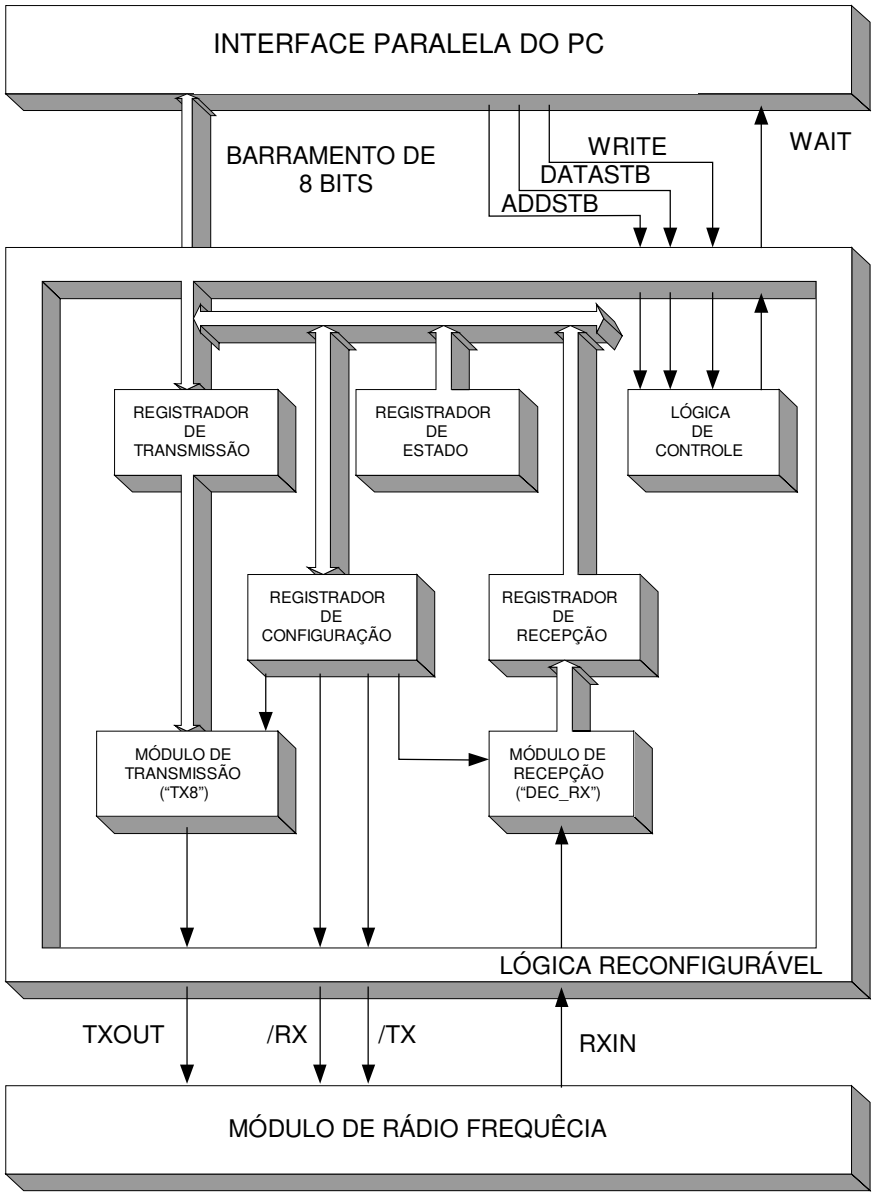


Figura 11– Diagrama dos blocos implementados em lógica reconfigurável na interface de comunicação remota.

Já na Figura 12 é detalhado o bloco "remoto" implementado em linguagem gráfica dentro do ambiente de desenvolvimento MAX+PLUS II. A descrição completa dos sub-blocos

implementados pode ser observada no apêndice D.

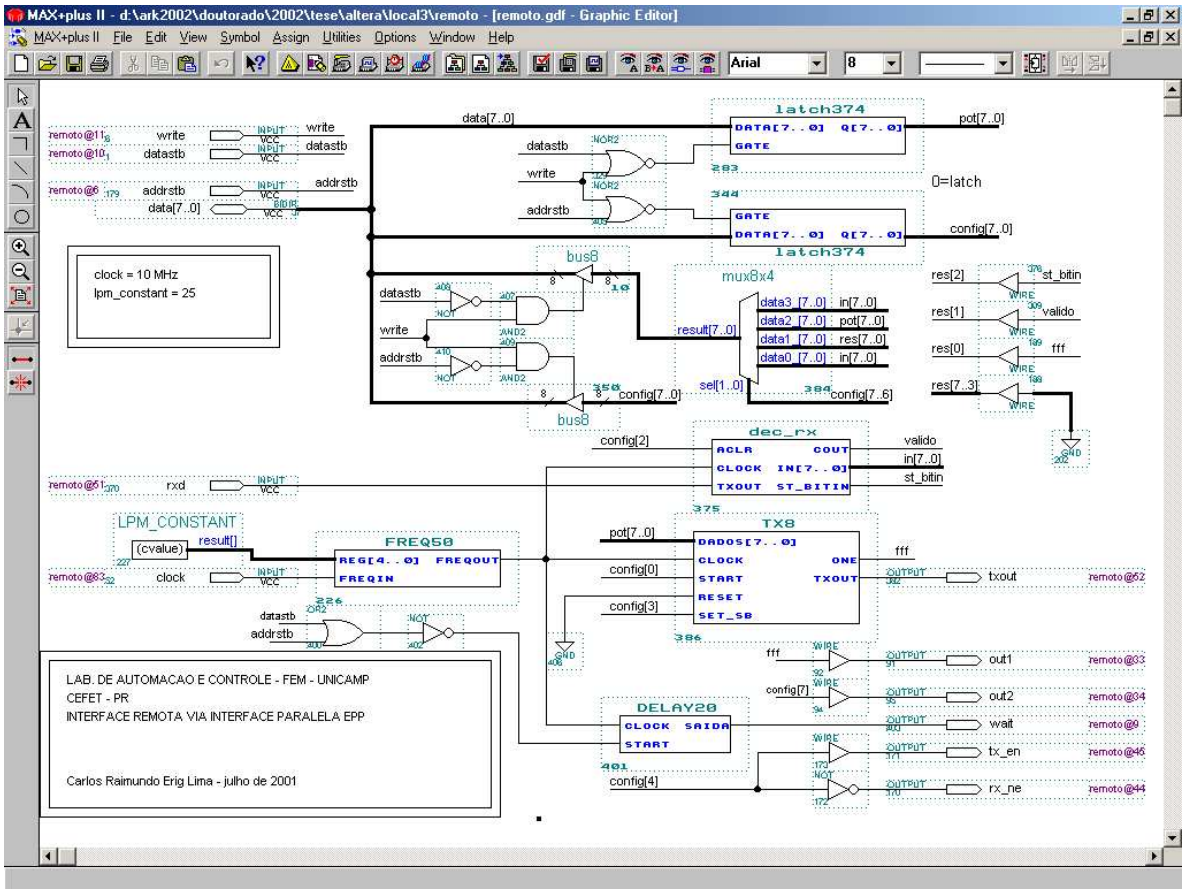


Figura 12 – Implementação em linguagem gráfica da interface de comunicação remota.

Os registradores da Figura 12 são detalhados:

- **Registrador de transmissão**: Registrador de 8 bits onde são armazenados os *bytes* que serão transmitidos pelo módulo "TX8". Pode sofrer operações de escrita e leitura. A operação de leitura é usada para verificação dos valores escritos, minimizando possíveis erros de transmissão.
- **Registrador de recepção**: Registrador de 8 bits onde são armazenados os *bytes* decodificados pelo módulo de recepção ("dec_rx"). Sofre apenas operações de leitura. A leitura não afeta o conteúdo do registrador.

- **Registrador de configuração**: Registrador de 8 bits onde são armazenados os bits de configuração dos diferentes blocos da EPLD. Pode sofrer operações de escrita e leitura. A operação de leitura é usada para verificação dos valores escritos, minimizando possíveis erros de operação. A Tabela 2 descreve os bits no registro de configuração.

Tabela 2- Descrição dos bits no registrador de configuração.

<table>
<tr><td colspan="3">REGISTRADOR DE CONFIGURAÇÃO</td></tr>
<tr><td>Bit usado</td><td>Função</td><td>Bloco da EPLD associado</td></tr>
<tr><td>Bit 0</td><td>Ativa inicio de transmissão do byte armazenado no registro de transmissão através do bloco de transmissão “TX8” ou do start-bit.</td><td>TX8</td></tr>
<tr><td>Bit 1</td><td>Reseta o bloco de recepção “dec_rx”</td><td>DEC_RX</td></tr>
<tr><td>Bit 2</td><td>Não usado. Disponível para uso futuro.</td><td></td></tr>
<tr><td>Bit 3</td><td>Define o tipo de transmissão “TX8” a ser feita no bloco de transmissão:
Bit 3 = 1 → transmissão de dado
Bit 3 = 0 → transmissão do start-bit</td><td>TX8</td></tr>
<tr><td>Bit 4</td><td>Programa o modo de operação do módulo de RF:
Bit 4 = 1 → transmissão de dado
Bit 4 = 0 → recepção de dado</td><td>Nenhum</td></tr>
<tr><td>Bit 5</td><td>Não usado. Disponível para uso futuro.</td><td></td></tr>
<tr><td>Bit 6
Bit 7</td><td>Define qual registrador será lido:
<table>
<tr><td>Bit 6</td><td>Bit 7</td><td>Registrador lido</td></tr>
<tr><td>0</td><td>0</td><td>Recepção</td></tr>
<tr><td>0</td><td>1</td><td>Estado</td></tr>
<tr><td>1</td><td>0</td><td>Transmissão</td></tr>
</table></td><td>MUX8X4</td></tr>
</table>

- **Registrador de estado**: Registrador de 8 bits onde são armazenados os bits que fornecem informações sobre o estado atual dos blocos internos da EPLD. Este registrador sofre apenas operações de leitura. A leitura não afeta o conteúdo do mesmo. A Tabela 3 apresenta a definição dos bits do registrador de estado.

Tabela 3 - Descrição dos bits no registrador de estado.

REGISTRADOR DE ESTADO		
Bit usado	Função	Bloco
Bit 0	Indica operação de transmissão de *byte* em andamento. Bit 0 = 1 → transmissão ocorrendo. Bit 0 = 0 → nenhuma transmissão ocorrendo.	TX8
Bit 1	Indica a existência de um *byte* válido no registro de recepção. Também indica se ocorreu a recepção de um *byte* válido. Bit 1 = 1 → *byte* válido recebido. Bit 1 = 0 → *byte* não recebido.	DEC_RX
Bit 2	Indica o recebimento de um star-bit no bloco de recepção "dec_rx". Bit 2 = 1 → start-bit recebido.	DEC_RX
Bit 3 –7	Não usado. Disponível para uso futuro.	

3.4.6 Interface com o Meio de Transmissão

O bloco interface com meio de transmissão interliga o bloco interface de comunicação remota com um dos possíveis meios de transmissão:

- Rádio freqüência.
- Infravermelho.
- Fio metálico.
- Fibra ótica.

Neste trabalho, apenas as interfaces de rádio freqüência e de fio metálico foram testadas. Os outros meios de comunicação propostos são deixados para possíveis implementações futuras. Contudo, é importante ressaltar que somente o bloco interface com o meio de transmissão precisaria de um novo projeto. O bloco interface de comunicação remota, no nível de controle

supervisor, e o bloco interface de comunicação local, no nível de controle embarcado, já implementam uma estrutura de comunicação serial completa.

A interligação física do bloco interface com o meio de transmissão com o bloco interface de comunicação remota pode ser feita através de simples ligações ponto a ponto, como ilustrado na Figura 13, a qual apresenta o diagrama de blocos simplificado da interligação da interface com o meio de comunicação (aqui chamada de interface de RF) com o módulo de comunicação escolhido para validação do ambiente (módulo BiM-433).

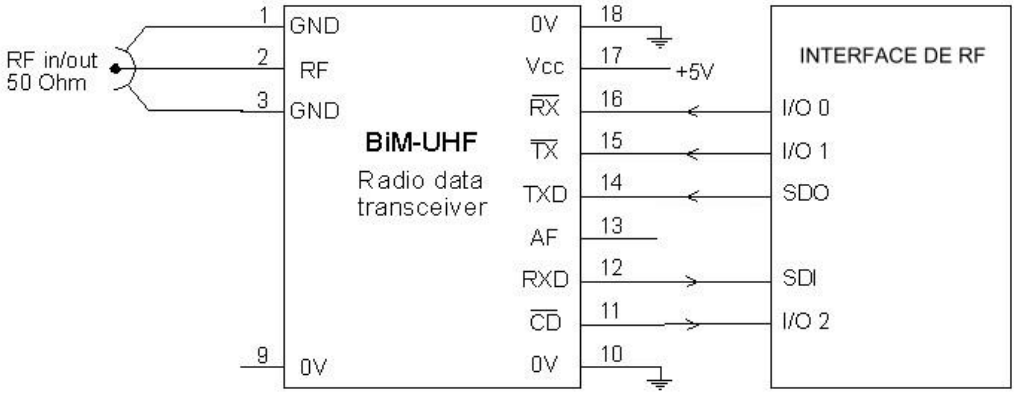


Figura 13 – Interligação do módulo BiM-433 com a interface de RF.

Neste caso, toda a implementação da interface de RF é realizada com lógica reconfigurável. Contudo, dependendo do modo de comunicação escolhido, recursos adicionais de *hardware* podem ser necessários.

3.4.7 Módulo de comunicação BiM-433

Neste trabalho optou-se pela utilização de um módulo transmissor-receptor BiM-433 [Radiometrix, 2001] apresentado na Figura 14. Além das pequenas dimensões e baixo consumo

deste módulo, a utilização de módulos de RF na aplicação proposta apresenta outras vantagens:

- Minimiza o número de elementos eletromecânicos e eletrônicos necessários para implementação da operação remota.

- Facilita o tratamento modular do sistema, sendo uma vantagem na evolução do projeto.

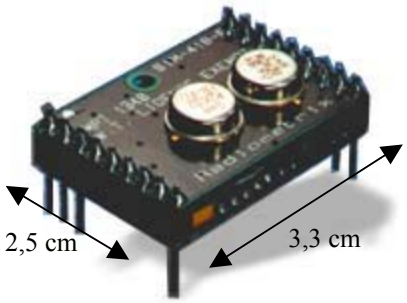


Figura 14 -Módulo de RF BiM-433 usado na implementação da comunicação remota.

O módulo de comunicação por rádio freqüência BiM-433 permite a comunicação por modo *half-duplex* com taxas de até 40 kbits por segundo. O alcance varia de 30 a 150 metros em função do ambiente de operação. Apresenta como vantagens baixo consumo, capacidade de transmissão analógica ou digital e alimentação única de 5V. Como seus sinais de controle são compatíveis com lógica CMOS, é necessário a utilização de resistores de *pull-up* para interface direta com a EPLD utilizada.

Algumas recomendações de operação são enfatizadas:

- O tempo T entre duas transições deve ser tal que 25μs <T< 2ms.

- A recepção é otimizada para formas de onda com ciclo de trabalho de 50%.

O Rádio-transmissor é um bloco de *hardware* variar com os requisitos da aplicação a ser usada. Características como máxima distância de operação, taxa de comunicação, tamanho e

consumo de energia devem ser consideradas na escolha do rádio-transmissor.

3.5 Nível de Controle Embarcado

O nível de controle embarcado encontra-se fisicamente localizado no sistema embarcado que se deseja projetar. Este nível pode ser implementado apenas com a utilização do *hardware* de suporte proposto. Esta configuração pode ser vista na Figura 15, onde o bloco lógica de controle é implementada em lógica reconfigurável. Uma outra alternativa pode ser vista na Figura 16, onde o bloco lógica de controle é implementado usando-se um microprocessador, microcontrolador ou ainda um DSP. Esta última configuração é utilizada em aplicações onde a demanda por lógica de controle seja tal que não possa ser facilmente implementada em lógica reconfigurável, demandando a construção de algoritmos em lógica convencional.

As seguintes funcionalidades são implementadas pelo nível de controle embarcado:

- Comunicação serial com o nível de controle supervisor ou remoto (transmissão e recepção de dados).

- Decodificação de comandos enviados pelo nível de controle supervisor ou remoto.

- Barramento de comunicação com o nível de controle local e lógica de controle.

- Lógica de controle responsável pelo tratamento de comandos recebidos do nível de controle supervisor, pelo tratamento de informações oriundas do nível de controle local (informações dos sensores) e pelo envio de sinais de comando para o nível de controle local (operação dos atuadores).

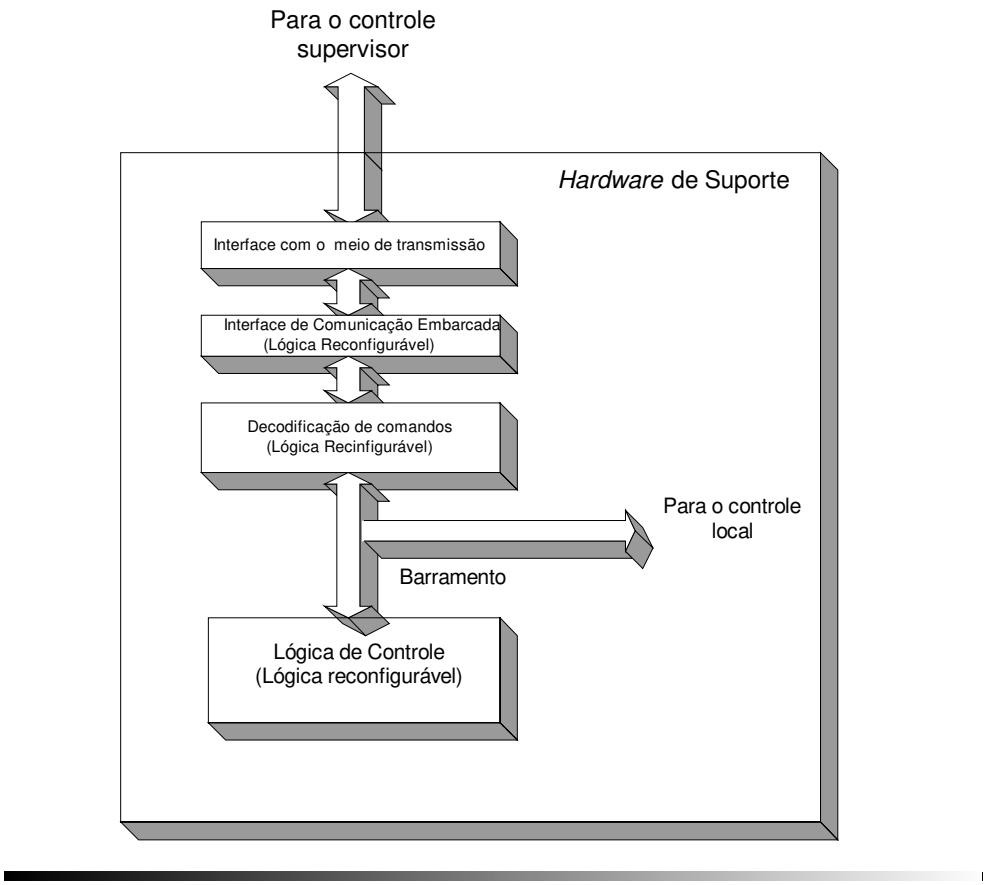


Figura 15 - Diagrama de blocos representando o nível de controle embarcado, com a lógica de controle implementada em lógica reconfigurável.

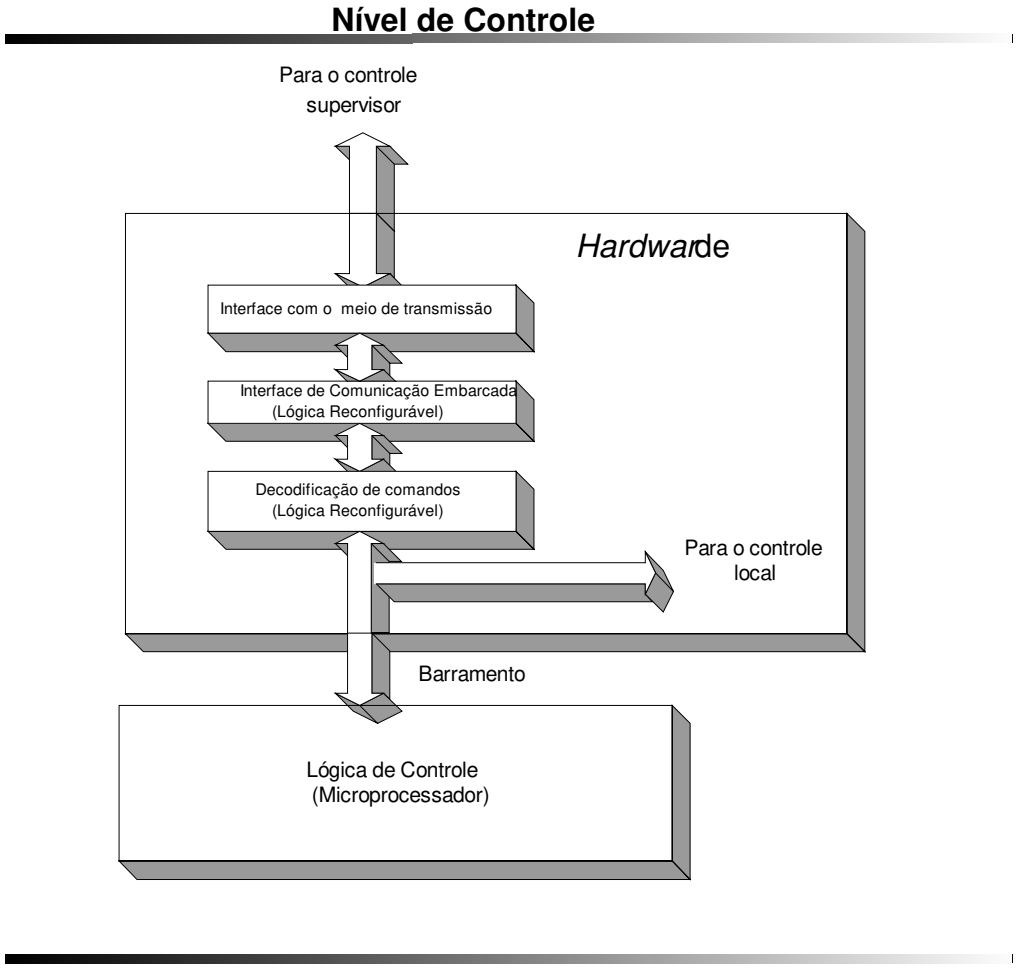


Figura 16 - Diagrama de blocos representando o nível de controle embarcado, com a lógica de controle implementada com um microprocessador externo.

3.5.1 Interface com o Meio de Transmissão

A interface com o meio de transmissão foi tratada no item 3.4.6.

3.5.2 Interface de Comunicação Embarcada e Interface de Decodificação de Comandos

Os blocos interface de comunicação embarcada e interface de decodificação de comandos são tratados em uma única seção por estarem fortemente integrados na implementação em lógica

reconfigurável.

A interface de comunicação embarcada é equivalente à interface de comunicação remota, só que localizada no sistema embarcado. Os módulos “TX8” (módulo de transmissão) e “dec_rx” (módulo de recepção) apresentados como elementos da interface de comunicação remota são também empregados na interface de comunicação embarcada. A Figura 17 apresenta o diagrama em blocos simplificado da interface de comunicação embarcada, onde são enfatizados os diferentes barramentos gerados para comunicação com o nível de controle local e com a lógica de controle.

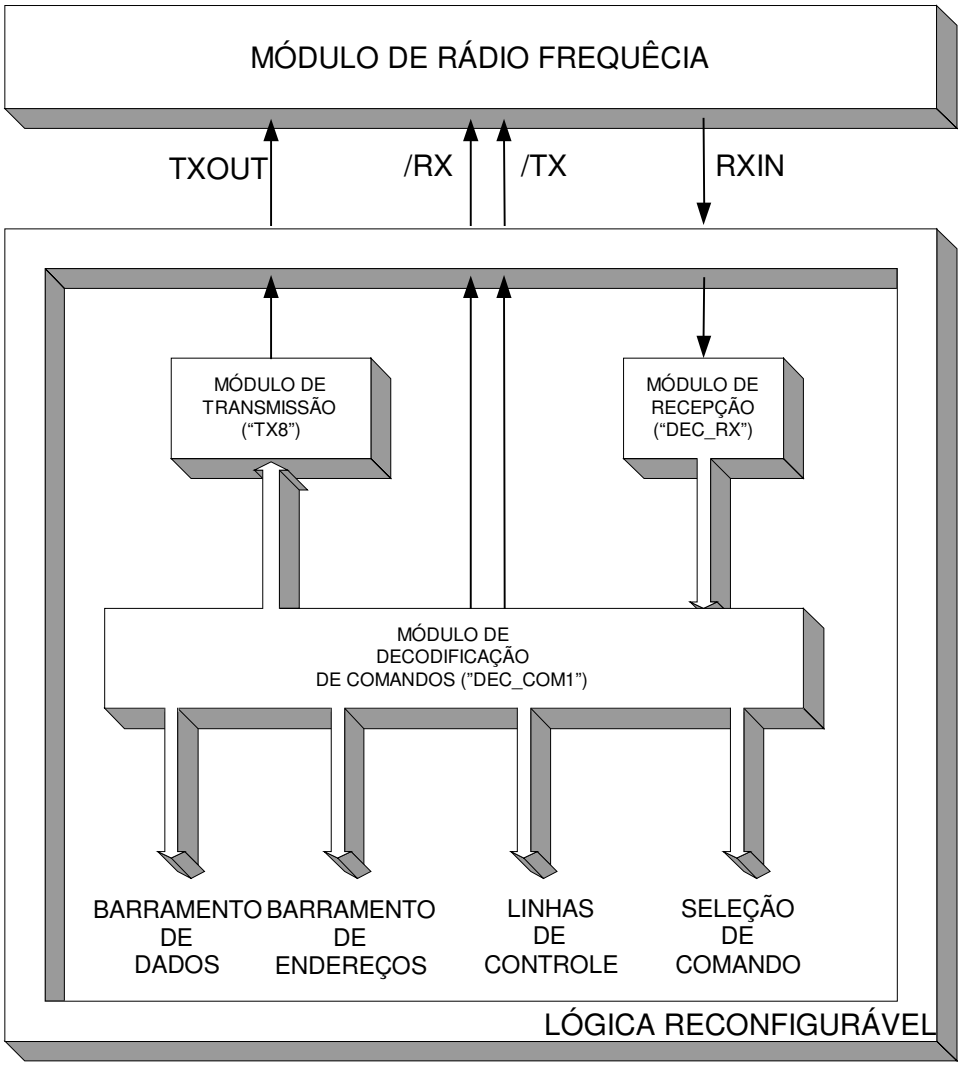


Figura 17- Diagrama de blocos simplificado da interface de comunicação embarcada implementado em lógica reconfigurável, bem como sua interligação com o módulo de rádio freqüência.

Já na Figura 18 é detalhado o bloco "com_local" implementado em linguagem gráfica dentro do ambiente de desenvolvimento MAX+PLUS II. A descrição completa dos sub-blocos implementados pode ser observada no apêndice D.

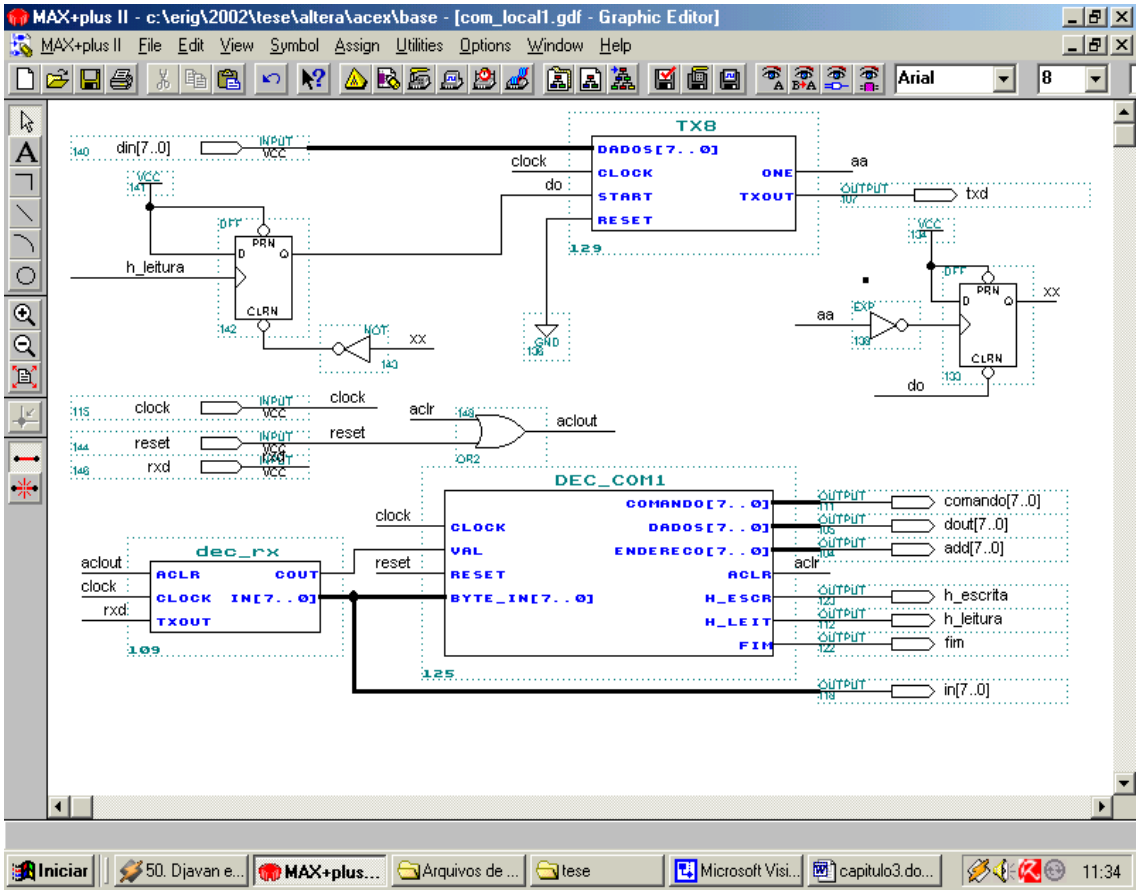


Figura 18 - – Implementação em linguagem gráfica da interface de comunicação embarcada e da interface de decodificação de comandos.

No módulo “dec_com1” uma máquina de estados determina se ocorre uma operação de escrita ou de leitura, criando os sinais de controle pertinentes. O módulo de codificação de transmissão traduz os dados armazenados em um endereço de memória de configuração para dados no padrão de transmissão serial adotado. O módulo de decodificação de recepção traduz dados no padrão de transmissão serial adotado em endereços e dados a serem armazenados na memória de configuração.

O módulo de decodificação de comandos apresenta as seguintes funcionalidades:

- Determina se uma operação de escrita ou de leitura.

- Associa os campos dos bits recebidos a registradores de deslocamento para os barramentos de dados e de endereços no caso de escrita em memória e a registradores de deslocamento para o barramento de dados no caso de leitura de memória.

- Cria sinais de controle de escrita e leitura na memória de configuração do sistema.

- Configura o módulo BiM-433 para recepção ou transmissão de dados.

3.6 Nível de Controle Local

O nível de controle local trata das interfaces dos sensores e atuadores a serem usados em um protótipo de sistema embarcado. Este nível de controle é considerado como um item separado no ambiente proposto devido aos seguintes pontos:

- Facilidade e organização de projeto, uma vez que cada sensor ou atuador considerado pode ser tratado como um bloco do sistema. Isto permite uma melhor divisão de tarefas entre os membros de uma equipe de projeto.

- Os sensores e atuadores tendem, cada vez mais, a exigir maiores quantidades de processamento em seu tratamento. Considerá-los como blocos individuais em um projeto vem a facilitar seu tratamento.

- Facilidade no re-projeto, uma vez que mudanças nas especificações de sensores ou atuadores podem ser feitas rapidamente.

O nível de controle local é composto por uma coleção de blocos associados ao barramento de dados, ao barramento de endereços e aos sinais de controle, fazendo interface com os atuadores e sensores, como pode ser observado na Figura 19.

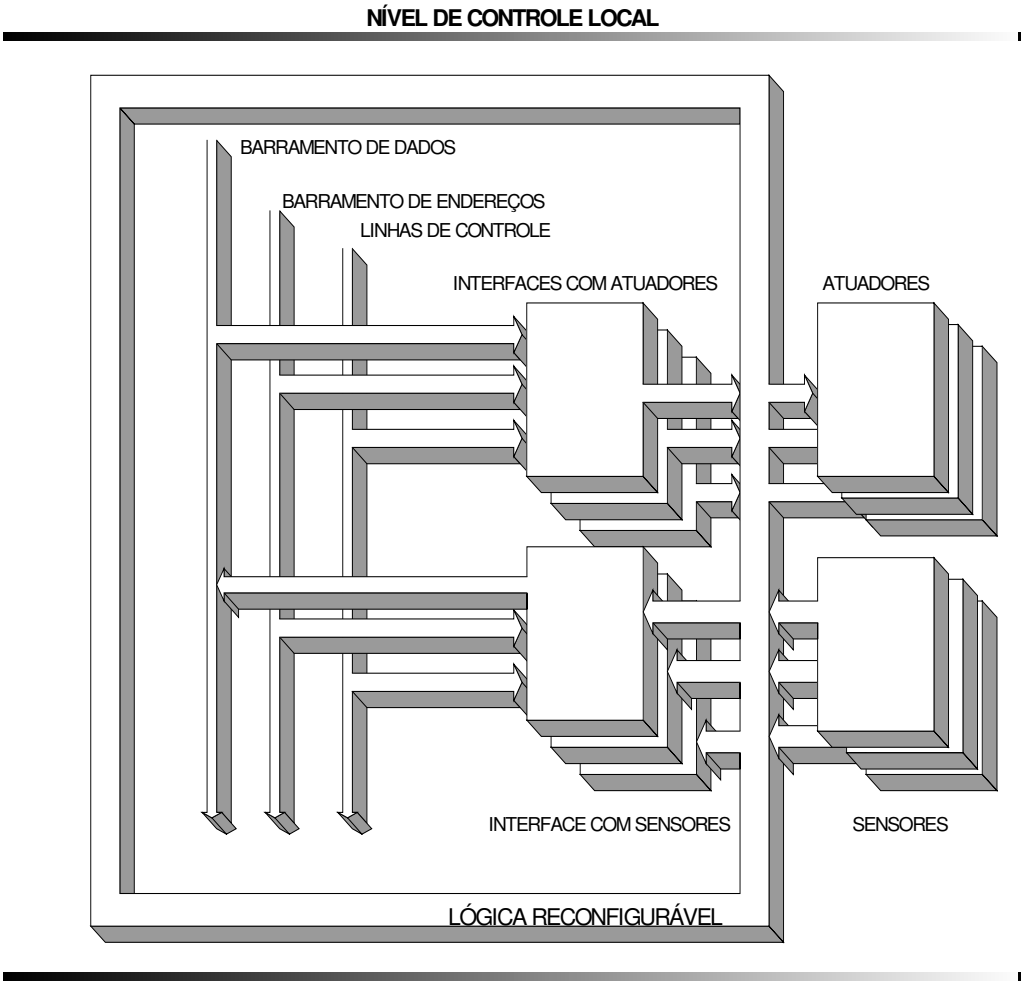


Figura 19 - Diagrama de blocos representando a interligação de um grupo de sensores e de um grupo de atuadores no nível de controle local, bem como sua interligação com o barramento de dados, o barramento de endereços e os sinais de controle.

No capítulo 5 são apresentados diversos exemplos de implementação em lógica reconfigurável de blocos associados a sensores e atuadores.

3.7 *Hardware* de Suporte

O *hardware* de suporte é composto de placas de circuito impresso desenvolvidas durante o período deste trabalho e de placas de circuito impresso fornecidas pela empresa PI componentes em função da participação do Laboratório de Automação Integrada e Robótica na 1ª Olimpíada

Universitária Altera. A placa fornecida pela empresa PI componentes é apresentada na Figura 20. Já as placas desenvolvidas foram criadas com a ferramenta de CAD CircuitMaker 2000 da empresa Protel International Limited. O desenvolvimento das mesmas segue uma orientação modular, permitindo a fácil expansão do sistema. A Figura 21 representa a arquitetura abordada, onde novas placas com funções adicionais podem ser acrescentadas.

Figura 20 - Placa de circuito impresso fornecida pela empresa PI componentes durante a 1ª Olimpíada Universitária Altera.

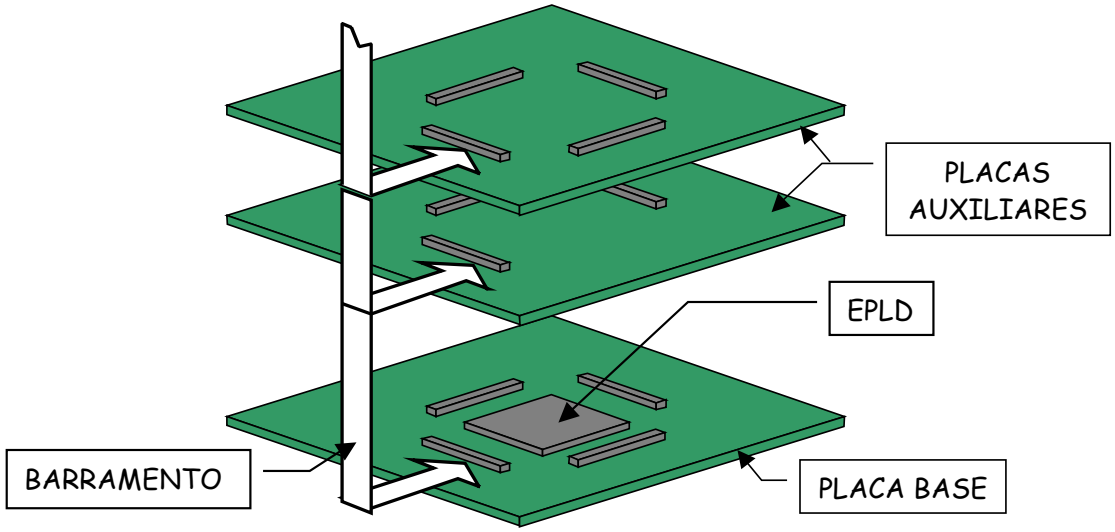


Figura 21 – Interligação por um barramento de diversas placas auxiliares à placa base. Esta é implementada com a EPLD EPM7128SLC84.

O barramento da figura disponibiliza todos os pinos da EPLD empregada para o uso de placas auxiliares. Deve-se observar que novas placas inseridas ao sistema (inclusive com outras EPLD) devem considerar os sinais de barramento já utilizados por outras placas, evitando eventuais conflitos de barramento.

A Figura 22 apresenta o diagrama esquemático da placa base implementada com a ELPD EPM7128SLC84. Esta placa apresenta um cristal oscilador e um conector JTAG para programação da EPLD *on board.* Os conectores CON1, CON2, CON3 e CON4 implementam o barramento de interligação com as placas auxiliares. As placas de circuito impresso correspondentes são apresentadas no Apêndice B.

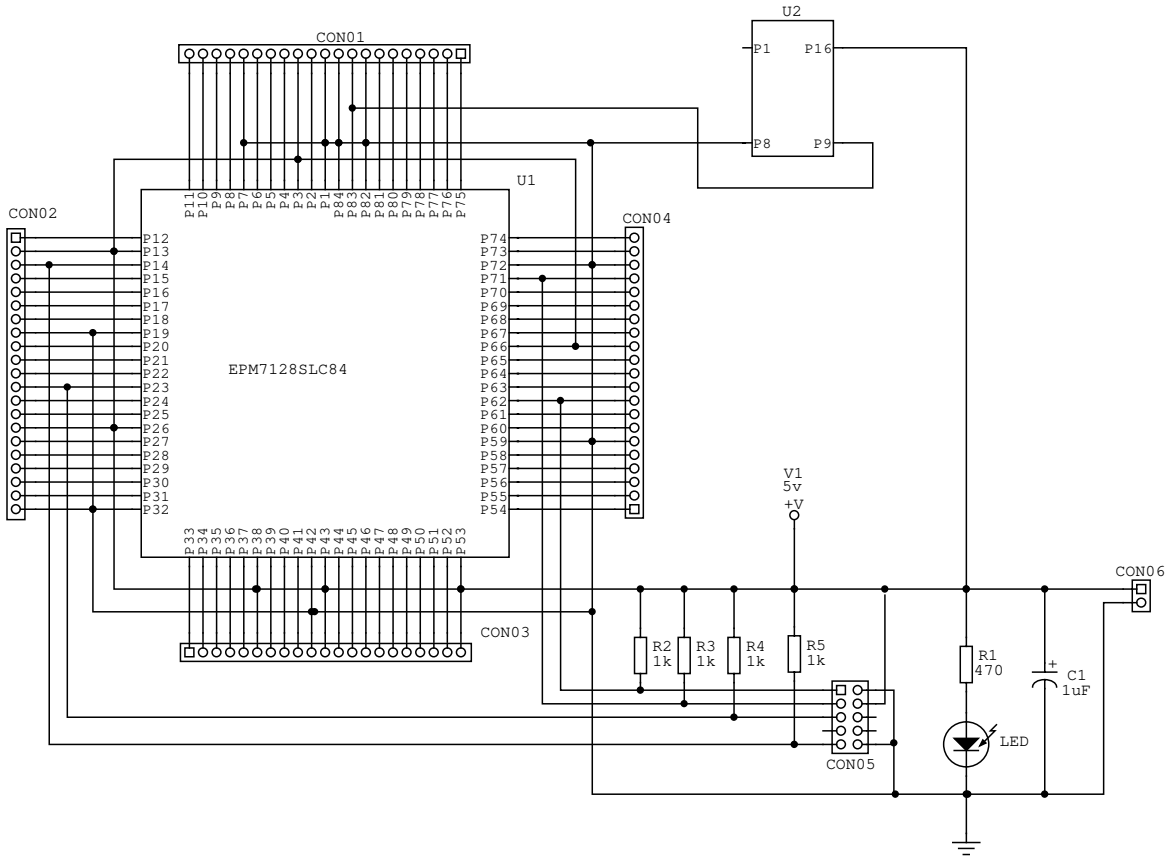


Figura 22 – Diagrama esquemático da placa base para a EPLD EPM7128SLC84 usada para o tratamento de diversos blocos do sistema reconfigurável.

A Figura 23 apresenta foto do produto final do projeto da placa base da Figura 22, onde se pode ver a EPLD utilizada, EPM7128SLC84.

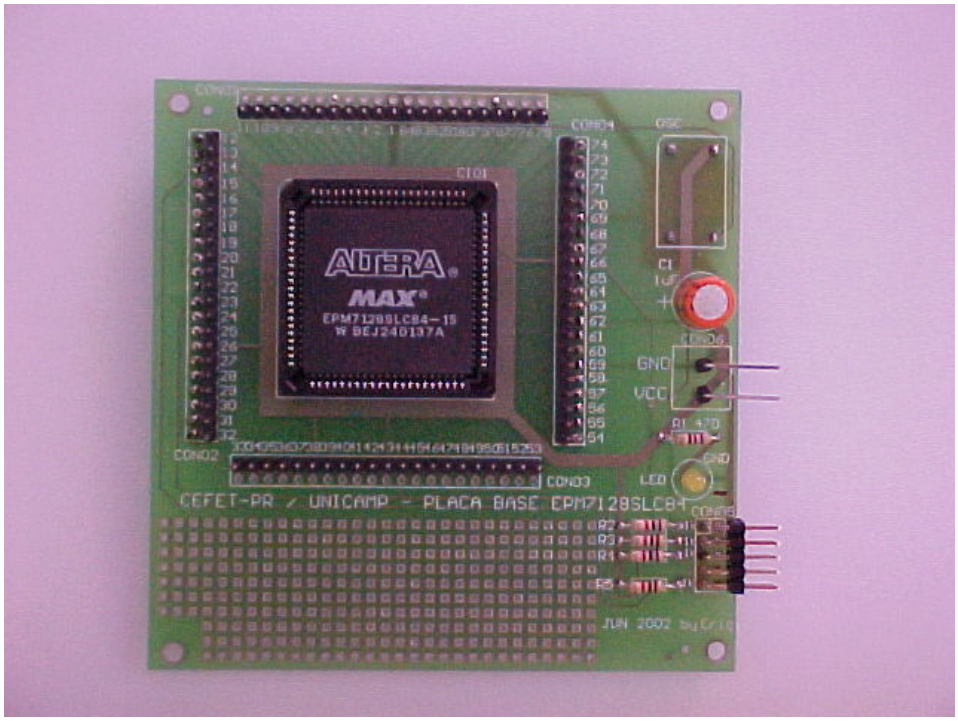


Figura 23 - Placa base para a EPLD EPM7128SLC84 usada para a implementação de diversos blocos do sistema reconfigurável.

A Figura 24 apresenta o diagrama esquemático da placa auxiliar que implementa a interface paralela e a interface de RF com o módulo de rádio freqüência BiM-433. Através de um cabo com 20 vias é possível conectar esta placa à interface paralela de um PC, permitindo o envio e o recebimento de dados entre a EPLD e o PC. Esta placa também apresenta oito *leds*, que podem ser usados para verificação visual do estado do sistema, por exemplo. As placas de circuito impresso correspondentes são apresentadas no Apêndice B. O produto final é apresentado na foto da Figura 25. É uma das placas auxiliares que podem ser conectadas à placa base.

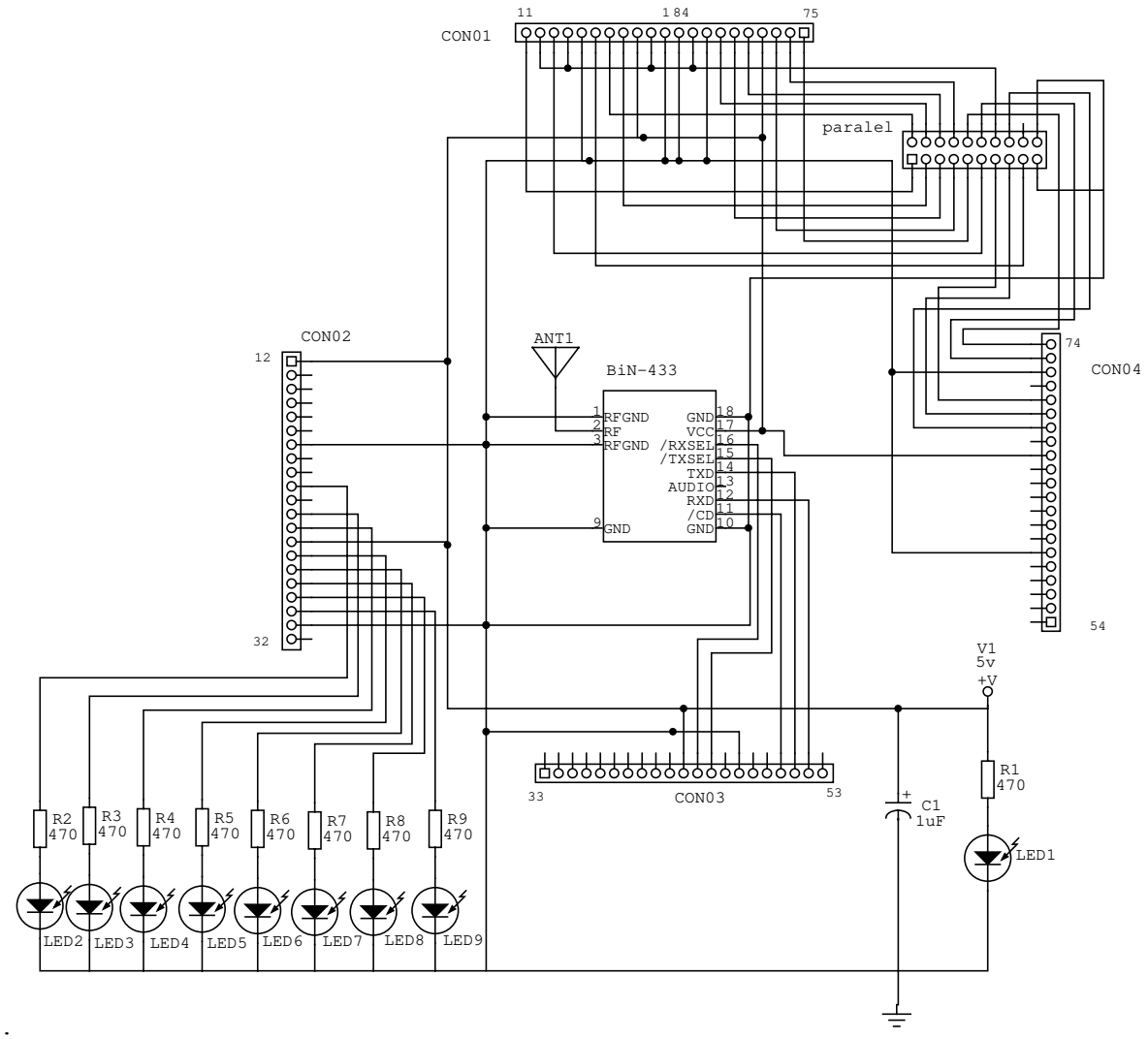


Figura 24 – Diagrama esquemático da placa de interface. Esta placa implementa a interface paralela e a interface de RF com o módulo BiM-433.

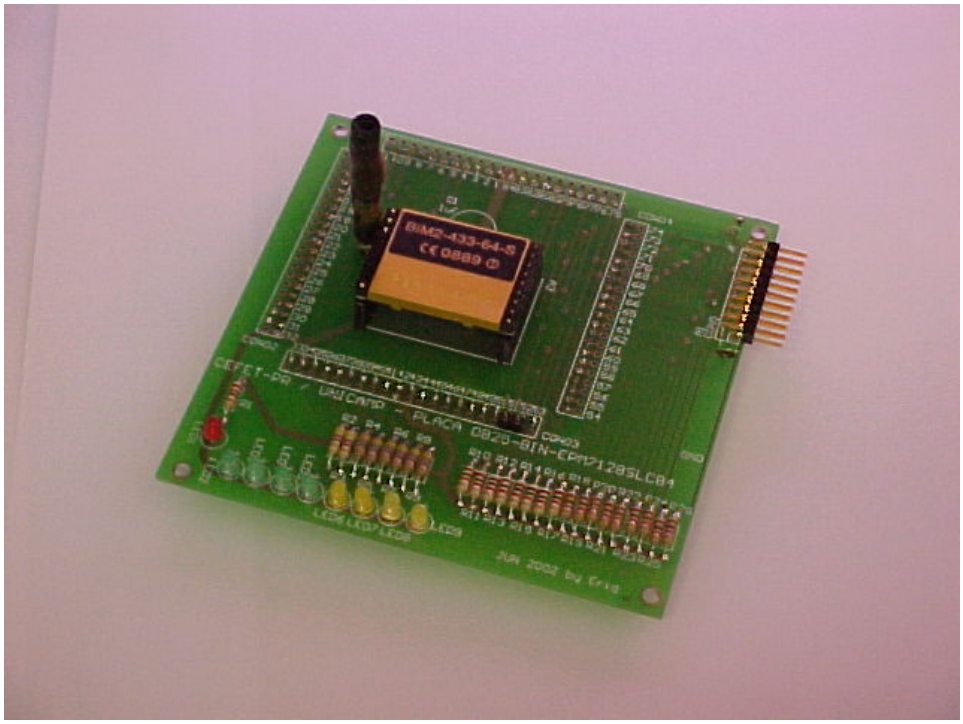


Figura 25 - Placa que implementa a interface paralela e a interface de RF com o módulo BiM-433.

Outra placa auxiliar construída para ser conectada ao barramento comum da placa base é a placa da Figura 26. Esta placa, com uma grande área para prototipagem, visa facilitar a adição de novos blocos de *hardware* ao sistema.

A Figura 27 apresenta o conjunto das placas auxiliares projetadas, conectadas por um barramento comum, à placa base. Esta configuração permite a fácil expansão do ambiente proposto, uma das características desejadas. Nesta mesma figura são mostrados os cabos de interligação com a interface paralela do PC e com a interface JTAG de programação da EPLD.

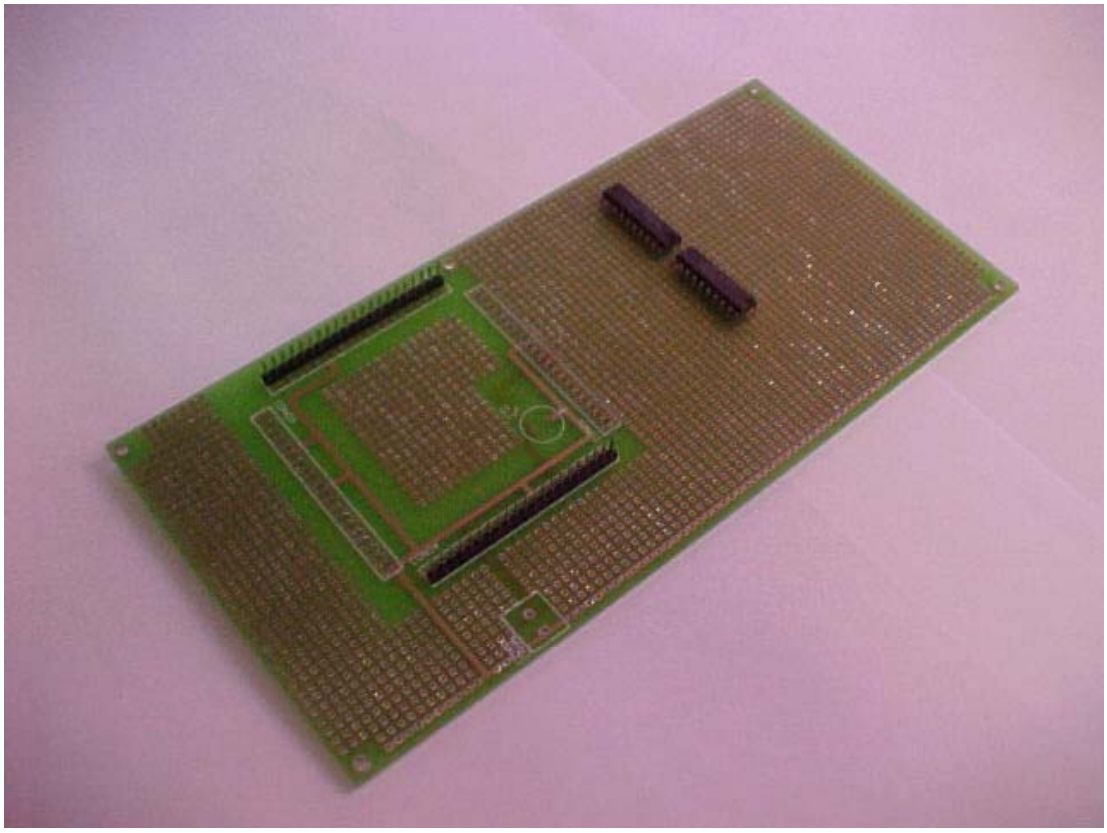


Figura 26 - Placa auxiliar com área de prototipação.

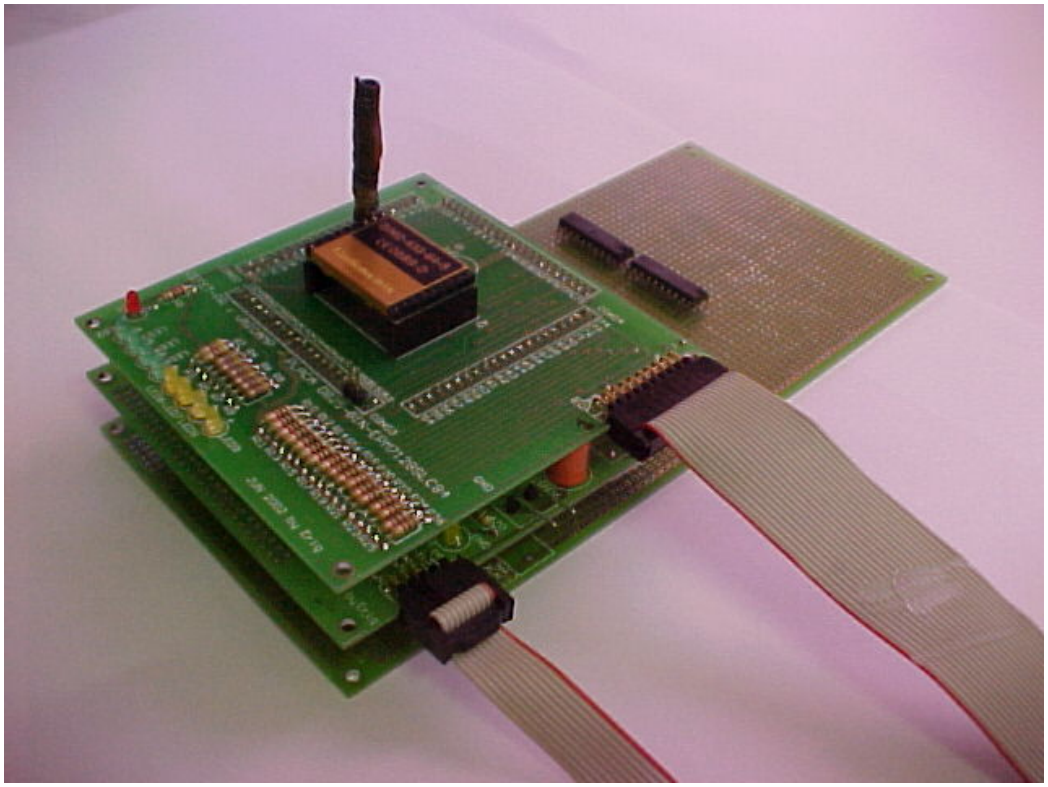


Figura 27 - Placas auxiliares conectadas à placa base, com os cabos de comunicação conectados.

A Figura 28 apresenta o diagrama esquemático da placa de potência projetada com base no circuito integrado L298. Esta placa permite o controle de motores DC, com máxima corrente de operação contínua de 4 A e máxima tensão de operação de 35 V. Com esta mesma placa é possível realizar o controle de uma fase de um motor de passo ou ainda de um motor AC através de modulação PWM. A Figura 29 apresenta foto do produto final obtido.

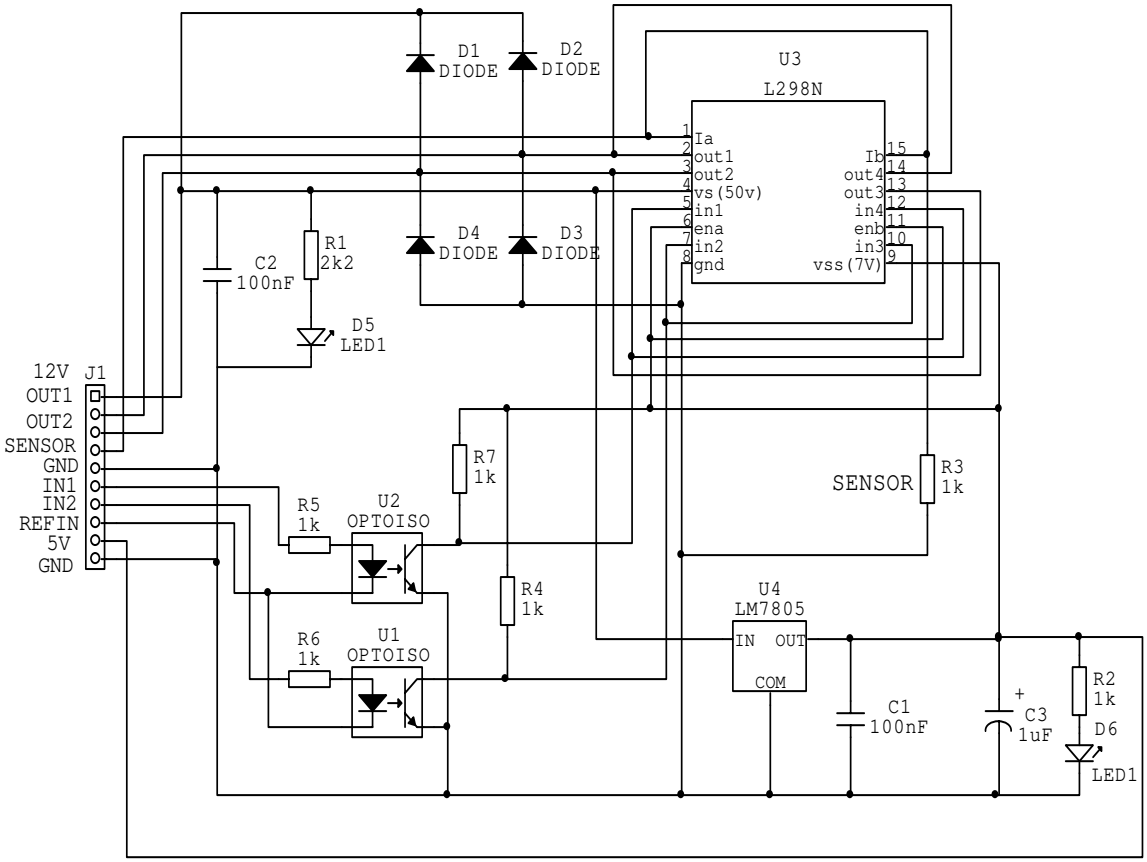


Figura 28 - Diagrama esquemático da placa de potência implementada com o circuito integrado L298N.

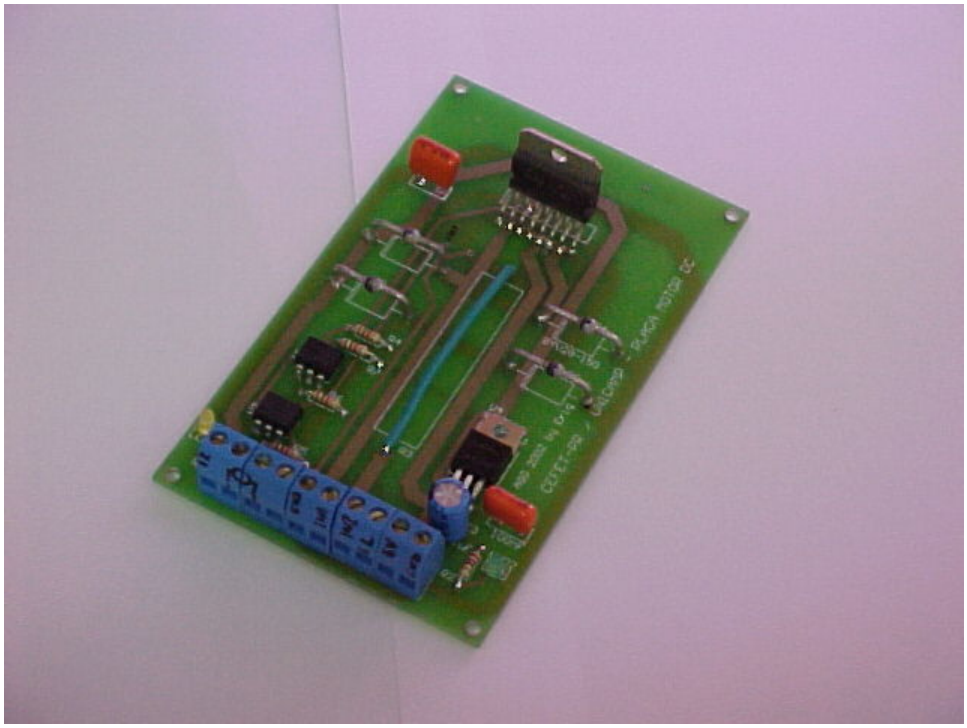


Figura 29 - Placa de potência projetada para controle de diversas famílias de motores.

3.8 Conclusão

Neste capítulo foi apresentada a uma visão geral do ambiente proposto. Procurou-se enfatizar os aspectos conceituais do ambiente, bem como aspectos de implementação de *software* e *hardware* que podem ser úteis para implementações práticas. Contudo, os blocos de *software* e *hardware* desenvolvidos podem (e devem) ser modificados, pois um dos objetivos do sistema é que seja aberto a modificações de seus blocos, permitindo o aprimoramento e a evolução do mesmo. No capítulo 4 são apresentados novos blocos associados ao projeto de controladores e implementados em lógica reconfigurável. Posteriormente, no capítulo 5, os elementos básicos desenvolvidos neste capítulo são utilizados e expandidos para implementação de sistemas práticos. Na apresentação destes projetos práticos, os novos elementos adicionados ao sistema, demonstram a necessidade de um ambiente de apoio a desenvolvimento de sistemas embarcados ser o mais aberto e flexível possível.

Capítulo 4

Implementação de Controladores com Lógica Reconfigurável

4.1 Introdução

Neste capítulo são propostas algumas soluções de implementação de controladores aplicados a sistemas embarcados utilizando computação reconfigurável. São consideradas duas estruturas de controle: um controlador PID implementado através de equações diferenças, e um controlador preditivo GPC. Neste último é proposta a implementação na forma genérica RST, através de equações diferenças derivadas dos polinômios RST.

4.2 Estrutura de Controle de um Atuador

Em sistemas embarcados móveis que utilizam motores de corrente contínua (cc) é possível realizar o controle de posição e velocidade das rodas acopladas a cada motor. Isto é necessário, por exemplo, no controle de trajetória de um sistema embarcado móvel. A Figura 30 apresenta um diagrama de blocos genérico de uma malha de controle implementado usando computação reconfigurável. O bloco PLD pode executar diversas estratégias de controle de posição/velocidade dos motores de corrente contínua acionados por um *driver* de potência PWM (Modulação por Largura de Pulso).

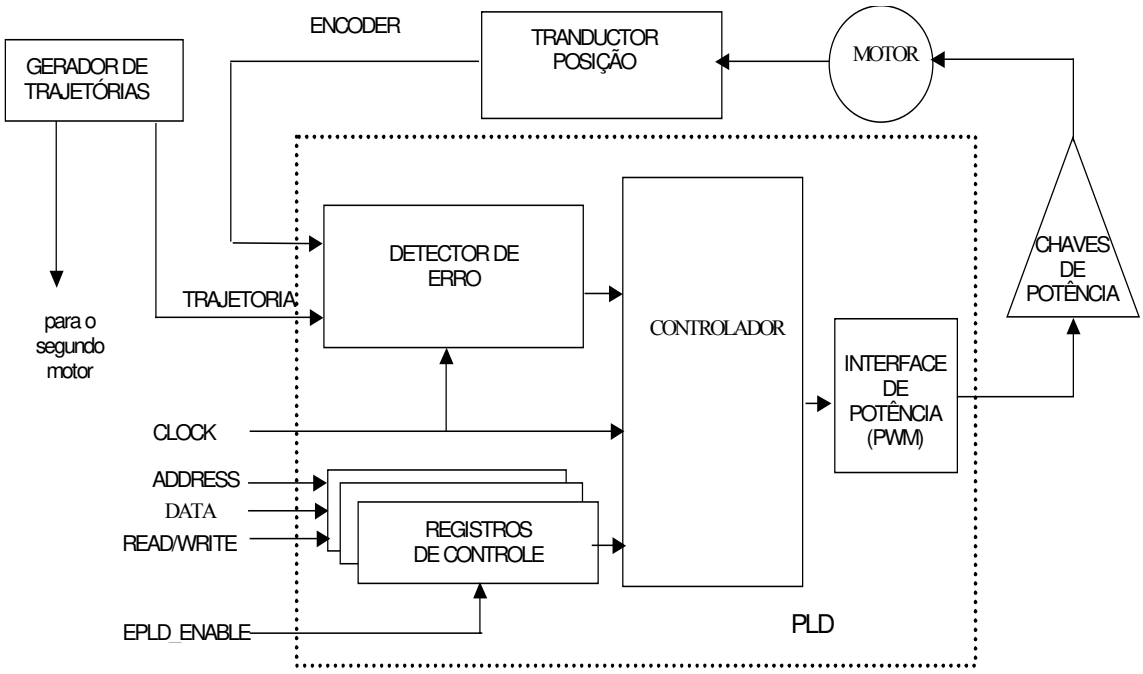


Figura 30 - Diagrama de blocos genérico de uma malha de controle de um motor cc implementado em computação reconfigurável.

4.2.1 Detecção de Erro

Na malha de controle apresentada na Figura 30 é considerada a implementação de um bloco detector de erro capaz de processar diretamente os sinais digitais oriundos de um transdutor de posição (*encoder* incremental) acoplado a cada motor e de um sinal digital de referência, representativo de uma trajetória. A saída da interface de potência é um sinal digital para um módulo de potência com controle PWM. Observe-se que nenhum conversor AD (Analógico-Digital) ou DA (Digital-Analógico) externo é necessário na malha de controle, permitindo uma minimização no número de componentes utilizados. Isto vem de encontro às características desejadas para a eletrônica de sistemas embarcados. Os sinais de entrada do bloco detector de erro são:

- TRAJETÓRIA: Sinal digital (trem de pulsos) representativo de comando (referência) para o sistema de acionamento. Este sinal tem associado ao seu período a velocidade com que a

trajetória é executada. O número de períodos (ou de pulsos) está associado ao deslocamento executado na trajetória.

- ENCODER: Sinal digital (trem de pulsos) proveniente de um transdutor de posição acoplado no eixo do motor (*encoder* incremental). Do mesmo modo, este sinal tem associado ao seu período a rotação do motor. O número de períodos (ou de pulsos) está associado ao deslocamento angular do eixo do motor.

Para ilustrar o comportamento do bloco detector de erro são apresentados na Figura 31 e Figura 32 exemplos dos sinais de entrada em duas situações típicas de detecção de erro (diferença dos períodos entre os sinais TRAJETÓRIA e ENCODER). Na Figura 31, o motor está sendo acelerado, com o sinal ENCODER apresentando um período maior que o do sinal TRAJETÓRIA. Na Figura 32 o motor está desacelerando, com o sinal ENCODER apresentando um período menor que o do sinal TRAJETÓRIA.

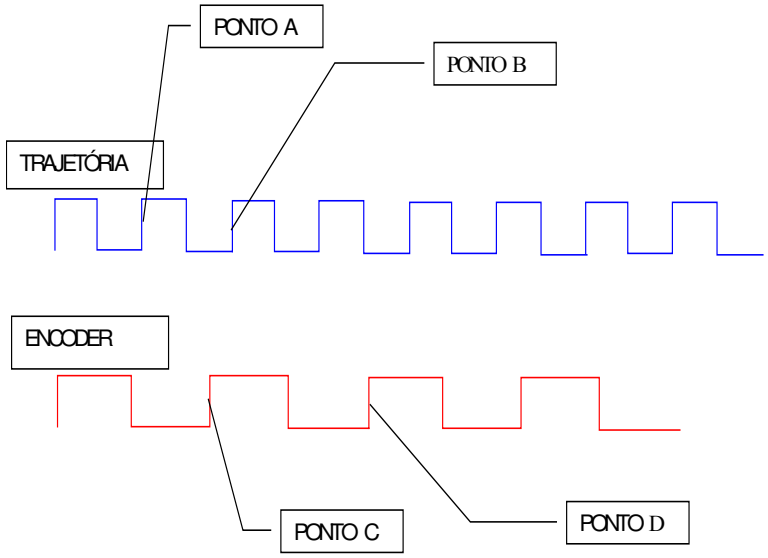


Figura 31 - Exemplo de sinais de entrada usados na detecção de erro. Situação de aceleração de um motor.

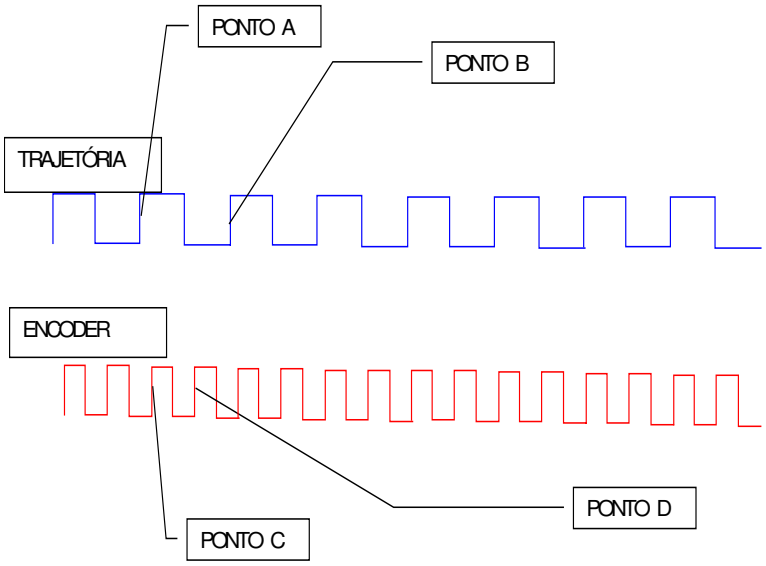


Figura 32 - Exemplo de sinais de entrada usados na detecção de erro. Situação de desaceleração de um motor.

Os pontos A, B, C e D são os pontos de transição de nível para os dois sinais de entrada. Estes pontos são significativos para caracterizar os eventos de transição de estados necessários para implementar uma máquina de estados síncrona em VHDL.

4.2.2 Estrutura de Controle implementada na forma polinomial RST

Na implementação de funções discretas em lógica reconfigurável deve-se levar em conta algumas características como resolução desejada, recursos máximos disponíveis e velocidade de execução do algoritmo construído. Uma solução de compromisso entre os recursos disponíveis e a velocidade de execução deve ser encontrada [Klotchkov e Pedersen, 1996] [Samet et al., 1998].

Como alternativa é possível implementar os controladores propostos sob a forma polinomial genérica RST. A representação de um controlador sob a forma polinomial genérica RST pode ser observada na Figura 32, onde a relação entre o sinal de entrada w e o sinal de saída y é dada por:

$$S(q^{-1})\Delta u(t) = -R(q^{-1})y(t)+T(q)w(t) \quad (4.1)$$

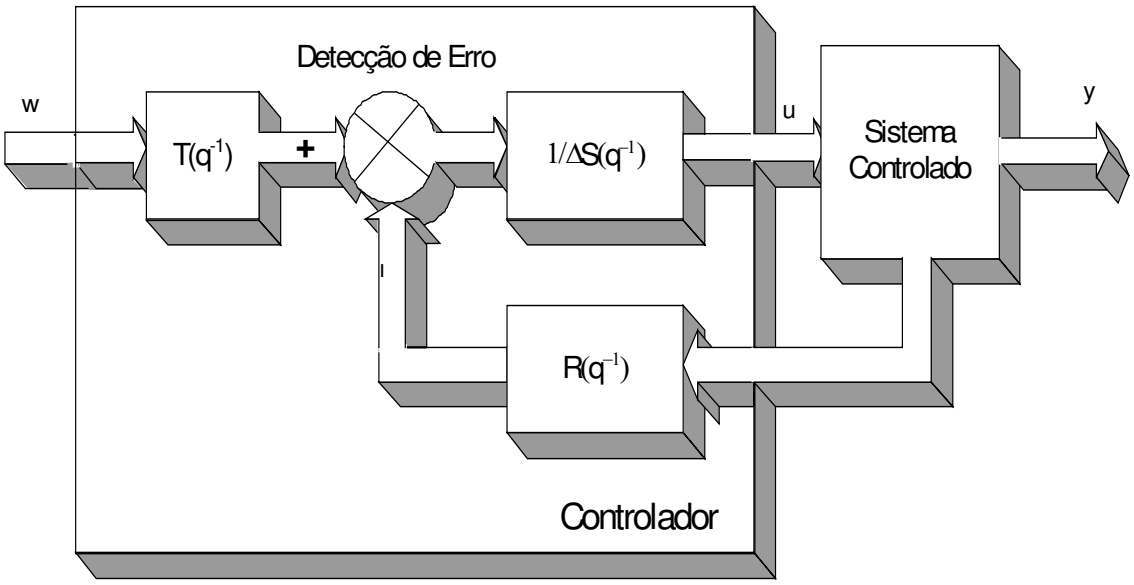


Figura 33 - Diagrama de blocos de um controlador na forma genérica RST.

4.3 Implementação em Computação Reconfigurável

A seguir são apresentadas as implementações e respectivas simulações dos principais elementos de uma malha de controle de sistemas embarcados utilizando computação reconfigurável. Assim, são apresentadas as implementações do bloco de detecção de erro, e de um controlador Proporcional, PID implementado através de equações diferenças na forma clássica e uma proposta de implementação na forma RST, e um controlador preditivo GPC na forma RST.

4.3.1 Implementação da Detecção de Erro

A detecção de erro é feita pela implementação da máquina de estados síncrona proposta na Figura 34. Esta máquina permite a comparação entre os períodos dos sinais de entrada sem a

necessidade de um sincronismo entre eles. Isto permite desprezar atrasos inerentes à malha de controle. Os estados ocorrem dentro de um ciclo de comparação. Cada ciclo de comparação é disparado pela transição de subida do sinal TRAJETÓRIA. Ao final de cada ciclo ocorre uma atualização do valor de erro a ser fornecido ao *drive* de potência PWM. A notação utilizada é apresentada na Tabela 4 e na Tabela 5.

Tabela 4 - Relação de estados da máquina de estados da Figura 34.

inicial	aguarda a subida do sinal TRAJETÓRIA para iniciar um ciclo de comparação;
Cc	habilita contador no sentido crescente;
Cd	habilita contador no sentido decrescente;
parado	contadores desabilitados, dentro do ciclo de comparação;
cc-cd	contadores crescente e decrescente habilitados.
Cc2	segunda habilitação do contador crescente dentro do ciclo de comparação.
final	estado que caracteriza o fim de contagem e prepara novo ciclo de comparação;
erro	sinal ENCODER não apresenta as transições para evolução dos estados.

Tabela 5 - Relação de sinais da máquina de estados da Figura 34.

CLOCK	sinal de sincronização;
T0→T1	rampa de subida do sinal TRAJETÓRIA;
T1→T0	rampa de descida do sinal TRAJETÓRIA;
E0→E1	rampa de descida do sinal ENCODER;
E1→E0	rampa de descida do sinal ENCODER;
T0	nível 0 do sinal TRAJETÓRIA;
T1	nível 1 do sinal TRAJETÓRIA;
E0	nível 0 do sinal ENCODER;
E1	nível 1 do sinal ENCODER.

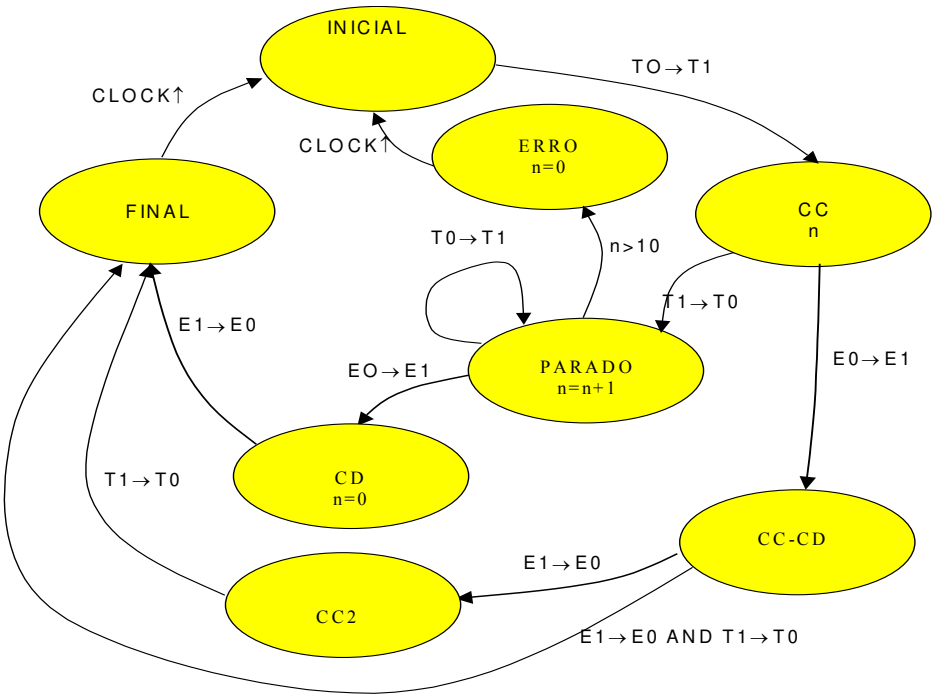


Figura 34 - Máquina de estados do bloco detector de erro.

A máquina de estados apresentada na Figura 34 é diretamente implementada em linguagem VHDL, o que é parcialmente descrito na Figura 35.

```
ENTITY detetor IS
      PORT( trajetoria, encoder, clock                      : IN BIT;
            fim_de_ciclo, erro_partida, habilita, sentido   : OUT BIT);
END detetor;
ARCHITECTURE detetor OF detetor IS       -- definição dos estados
TYPE  ESTADOS  IS  (inicial,  cc,  cc2,  cd,  cccd,  parado,  final,  erro,
resetc);
SIGNAL estado : ESTADOS;
SIGNAL contador : INTEGER RANGE 0 TO 255;
SIGNAL  ta, ea : BIT;
BEGIN
      PROCESS (trajetoria, encoder,clock)
      BEGIN
            IF ((clock'EVENT) AND (clock ='1'))THEN
            CASE estado IS
              WHEN inicial =>       -- Detecta TO→T1
                IF trajetoria ='1' AND ta = '0' THEN
                  estado <= cc;
                END IF;
              WHEN cc =>            --Detecta T1→T0
                IF  trajetoria ='0' AND ta = '1' THEN
                  estado <= parado;
                ELSIF encoder ='1' AND ea = '0' THEN    estado <= cccd;
                END IF;
                  .
                  .
                  .
              WHEN final =>   estado <= resetc;
               WHEN resetc =>  estado <= inicial;
              WHEN erro =>
                contador <= 0;
                estado <= inicial;
            END CASE;
            ta<=trajetoria;
            ea<= encoder;
      END IF;
      END PROCESS;
            habilita <= '1' WHEN (estado = cc OR estado = cd OR estado =
cc2) ELSE '0';
            sentido <= '0' WHEN estado = cd ELSE '1';
            fim_de_ciclo <= '1' WHEN estado = final ELSE '0';
            erro_partida <= '1' WHEN estado = erro ELSE '0';
            reset <= '1' WHEN estado = resetc ELSE '0';
END detetor;
```

Figura 35 – Descrição parcial da implementação em VHDL da máquina de estados síncrona para detecção de erro entre os períodos de dois pulsos digitais.

A máquina de estados é síncrona, sendo o sincronismo realizado pelo sinal de CLOCK. Isto não é estritamente necessário, ou seja, é possível implementar esta máquina sem ser síncrona. Contudo, a implementação de uma máquina assíncrona demanda a elaboração de rotinas mais complexas, não sendo justificada sua utilização neste projeto.

Esta máquina de estados é considerada posteriormente no projeto como um componente, chamado detector de erro. Esta característica de hierarquização é uma grande vantagem na organização dos projetos com lógica reconfigurável.

A parte implementada em linguagem gráfica pode ser vista na Figura 36. O bloco "detector", criado em VHDL, é agora considerado como um elemento gráfico do projeto. Além disto, o projeto fica bastante simplificado com a utilização de bibliotecas já disponíveis, a exemplo do contador utilizado (LPM_COUNTER), derivado de uma biblioteca de uso genérico.

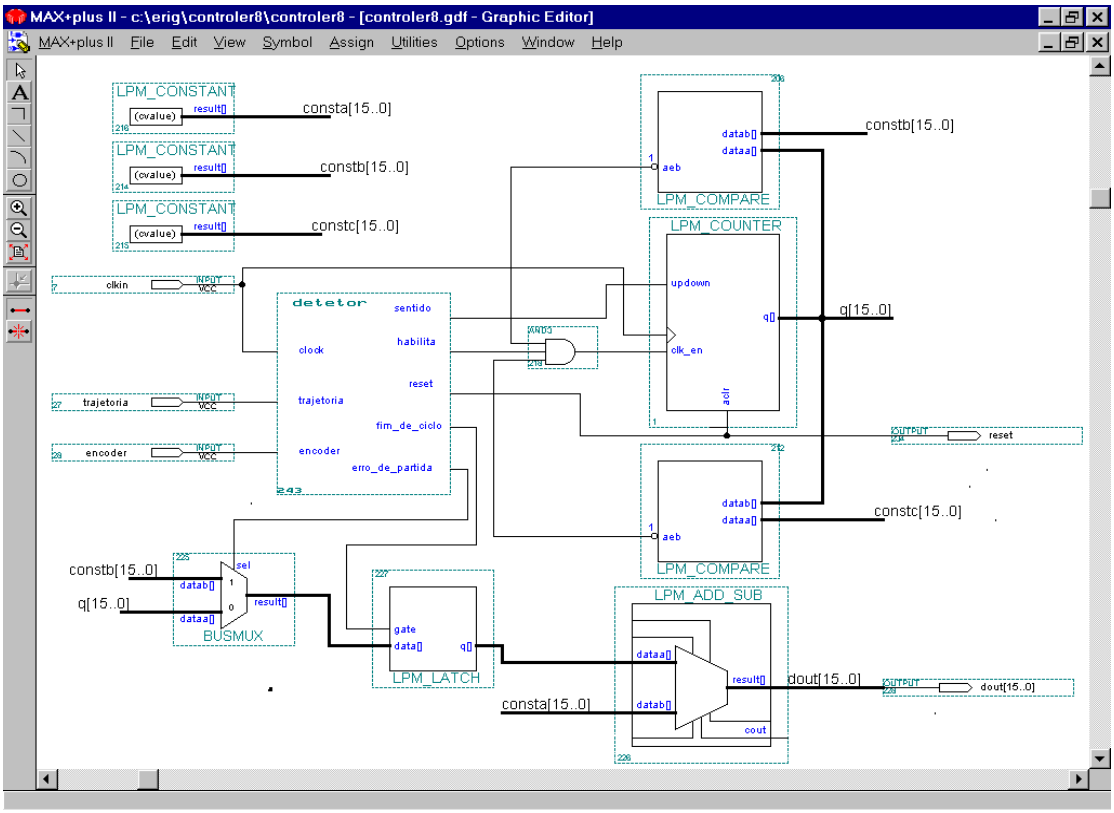


Figura 36 - Exemplo de blocos implementados em linguagem gráfica do controlador proporcional.

4.3.2 Simulação do Bloco Detector de Erro (Controlador Proporcional)

O ambiente de simulação permite a construção dos sinais das entradas e a conseqüente visualização dos sinais de saída simulados. Esta simulação é bastante sofisticada, permitindo que sejam considerados tempos de *setup* e *hold*, *delay* e *glitch.*

A Figura 37 apresenta um diagrama de tempos obtido através de uma tela de simulação gráfica, onde o contador apresenta resultados convergindo para zero à medida que a diferença entre os períodos dos sinais diminui. São apresentados dois grupos de simulação. Inicialmente é apresentado o diagrama temporal usando um sinal de CLOCK de 500 kHz. A seguir é utilizado um sinal de CLOCK de 50 kHz. Na prática, multiplicou-se a constante proporcional do sinal de erro por cinco, o que pode ser observado pelos valores finais do contador (dout[15…0]). O erro diminui a medida que os sinais de entrada convergem para um período igual.

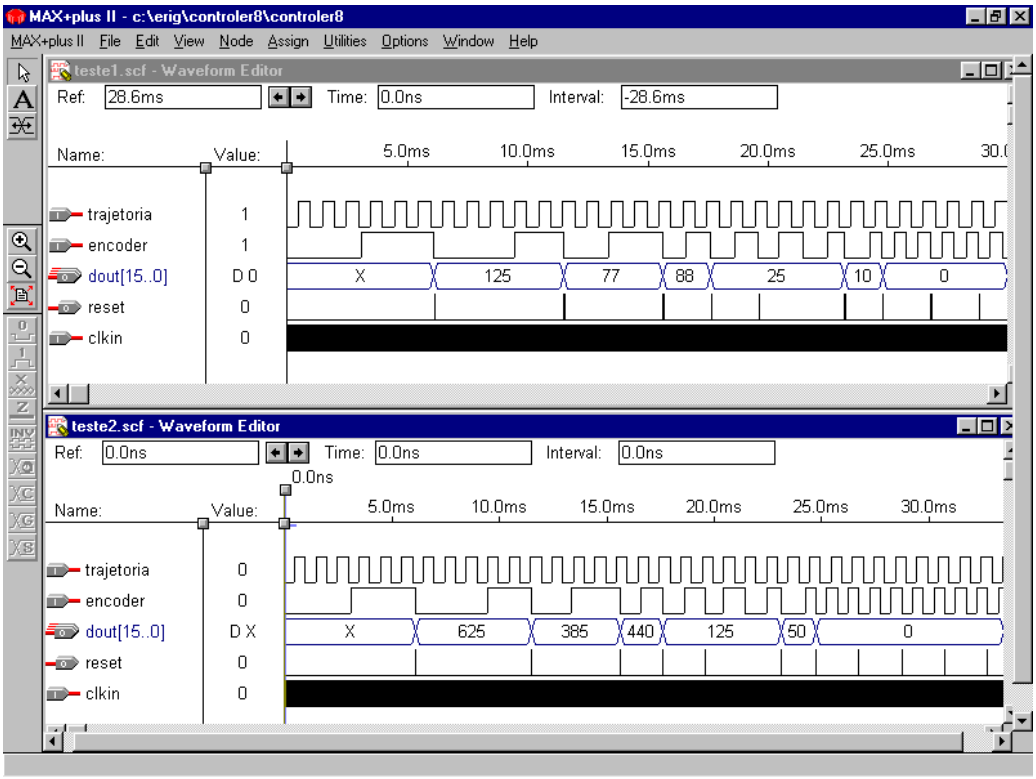


Figura 37 – Diagrama de tempos enfatizando a saída do contador de erro em função dos sinais TRAJETÓRIA e ENCODER.

Diferentes estratégias de controle podem ser aplicadas uma vez que sinal de erro é obtido. O próprio bloco de detecção de erro pode ser utilizado como um controlador proporcional onde o ganho varia em função da freqüência do sinal CLOCK, como pode ser visto na Figura 37.

4.3.3 Controlador PID Implementado com Lógica Reconfigurável

Uma alternativa para a implementação de equações de um PID digital é a utilização de bibliotecas construídas para manipulação algébrica em VHDL. Diversos trabalhos podem auxiliar este tipo de projeto [Klotchkov e Pedersen, 1996] [Kolling et al., 1998] [Samet et al., 1998].

Um controlador PID digital pode ser descrito [Aström e Wittenmark, 1997] como:

$$u[k] = u[k-1] + e[k](kp+kd+ki) + e[k-1](ki-kp-2kd)+ e[k-2]kd, \quad [4.2]$$

onde u[k] é a saída atual do PID, u[k-1] é a saída do PID no instante anterior de amostragem. Os termos e[k], e[k-1] e e[k-2] são os sinais de entrada do PID (sinal de erro) em diferentes instantes de amostragem. Os coeficientes do kp, ki e kd são os coeficientes de sintonia do PID.

A Figura 38 apresenta o diagrama de blocos que representa a implementação da equação [4.2]. O bloco de registro de erro armazenam os valores de e[k], e[k-1] e e[k-2], deslocando-os a cada amostra realizada (e[k-1] = e[k] e e[k-2] = e[k-1]). Já os registros de saída armazenam os valores de u[k] e u[k-1].

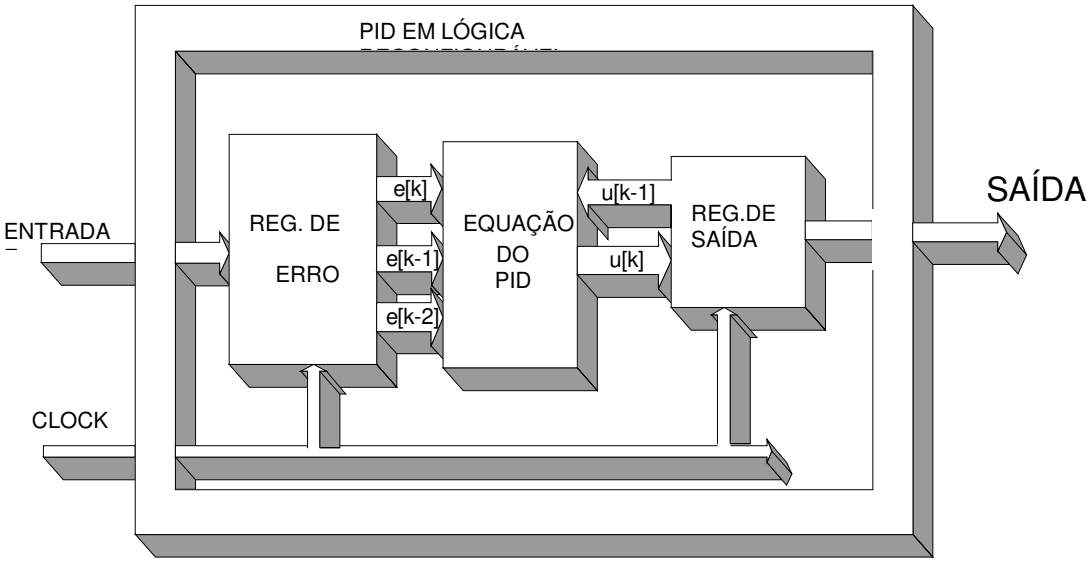


Figura 38 - Diagrama de blocos do PID digital implementado em lógica reconfigurável.

Na Figura 39 é apresentada a implementação em linguagem gráfica do PID da equação[4.1].

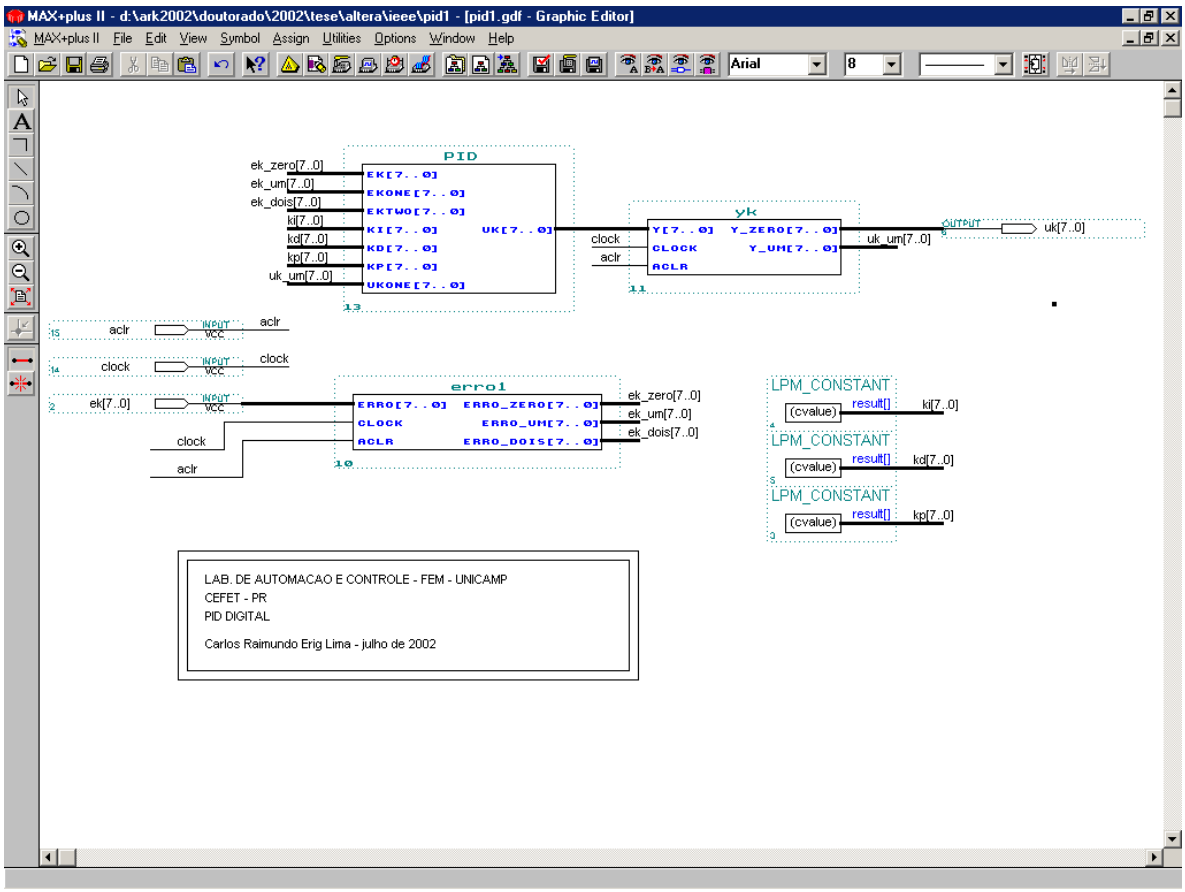


Figura 39 - Implementação em linguagem gráfica do diagrama de blocos da Figura 38.

O bloco “PID” é implementado em VHDL. Este bloco, através da utilização de bibliotecas específicas [Altera, 2002] para manipulação algébrica, permite o desenvolvimento da equação [4.1]. O bloco “erro1” e o bloco “yk” implementam os registradores de erro e os registradores de saída representados na Figura 38. A implementação do bloco “PID” é apresentada na Figura 40.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

ENTITY pid IS
        PORT (ek : IN std_logic_vector (7 DOWNTO 0);
                ekone : IN std_logic_vector (7 DOWNTO 0);
                ektwo : IN std_logic_vector (7 DOWNTO 0);
                ki : IN std_logic_vector (7 DOWNTO 0);
                kd : IN std_logic_vector (7 DOWNTO 0);
                kp : IN std_logic_vector (7 DOWNTO 0);
                ukone : IN std_logic_vector (7 DOWNTO 0);
                uk : OUT std_logic_vector (7 DOWNTO 0));

END pid;

ARCHITECTURE piddig OF pid IS

BEGIN

        uk <= ukone + ek * kp + ek * ki + ek * kd + ekone * kp - ekone * ki  - ekone *
kd + ektwo * kd;

END piddig;
```

Figura 40 - Implementação em VHDL da equação do PID digital [4.1].

A simulação do controlador implementado pode ser observada na Figura 41. No gráfico superior é usada uma entrada ek constante, com valores de Ki = 1, Kp = 10 e Kd = 10, observando-se a resposta do controlador para vários períodos de amostragem. No gráfico inferior é usada uma entrada ek constante, com valores de Ki = 0, Kp = 10 e Kd = 0, observando-se a resposta do controlador para vários períodos de amostragem.

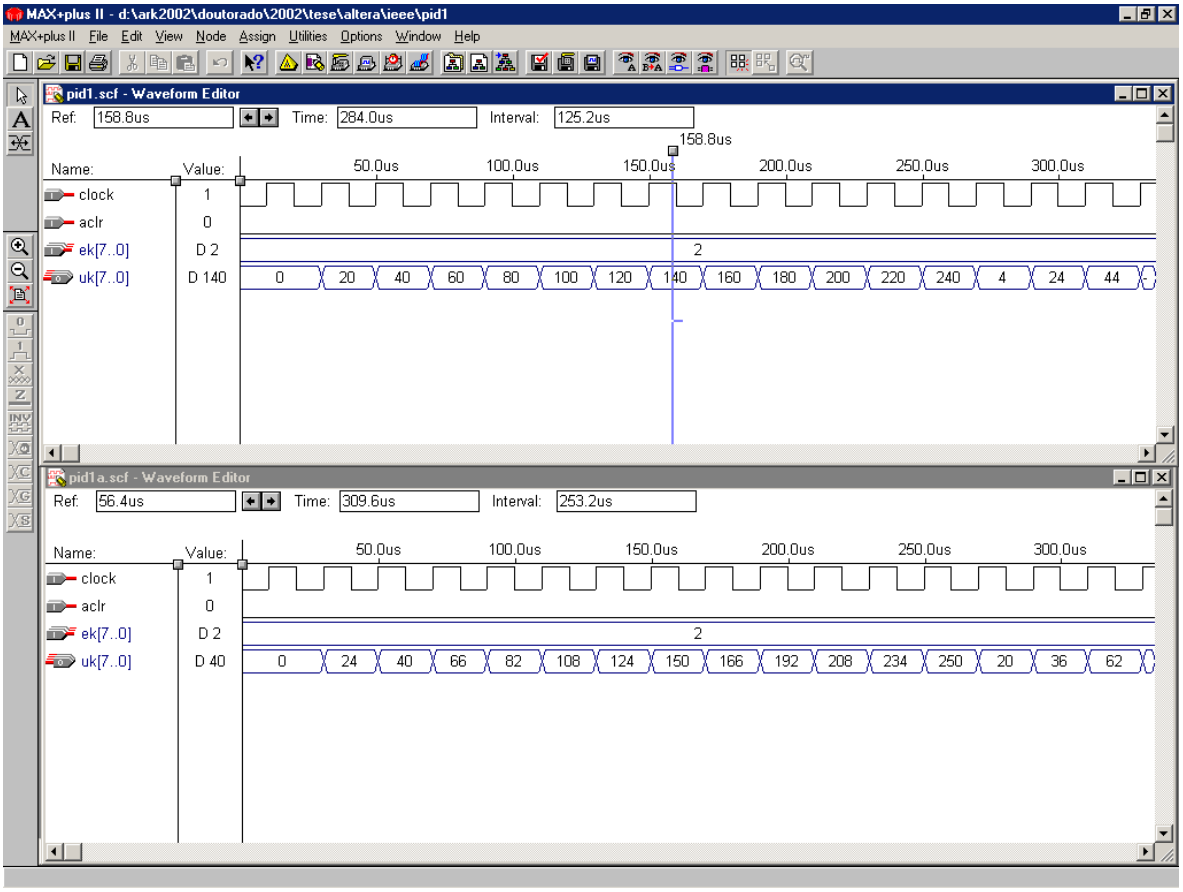


Figura 41 - Simulação do controlador PID implementado.

Uma implementação alternativa do controlador PID proposto pode ser feita usando a representação genérica RST vista na Figura 43.

4.4 Proposta de Implementação de um Controlador Preditivo Generalizado – GPC

Para a implementação deste tipo de controlador é necessário um modelo de referência interna do sistema controlado, sendo que este modelo é tratado, junto com parâmetros de sintonia, para geração de coeficientes de um polinômio RST [Dumur e Boucher,1994]. Uma síntese dos principais fundamentos teóricos necessários para a implementação de um Controlador Preditivo Generalizado - GPC é apresentado no apêndice A deste trabalho.

Trabalhos realizados pelo *Service d'Automatique* da SUPELEC utilizam polinômios RST obtidos pela execução de algoritmos em MatLab™ e posterior execução das equações a diferenças através de computação convencional (programa em Qbasic™). Isto pode ser visto na Figura 42, onde também é representada a alternativa de implementação usando lógica reconfigurável.

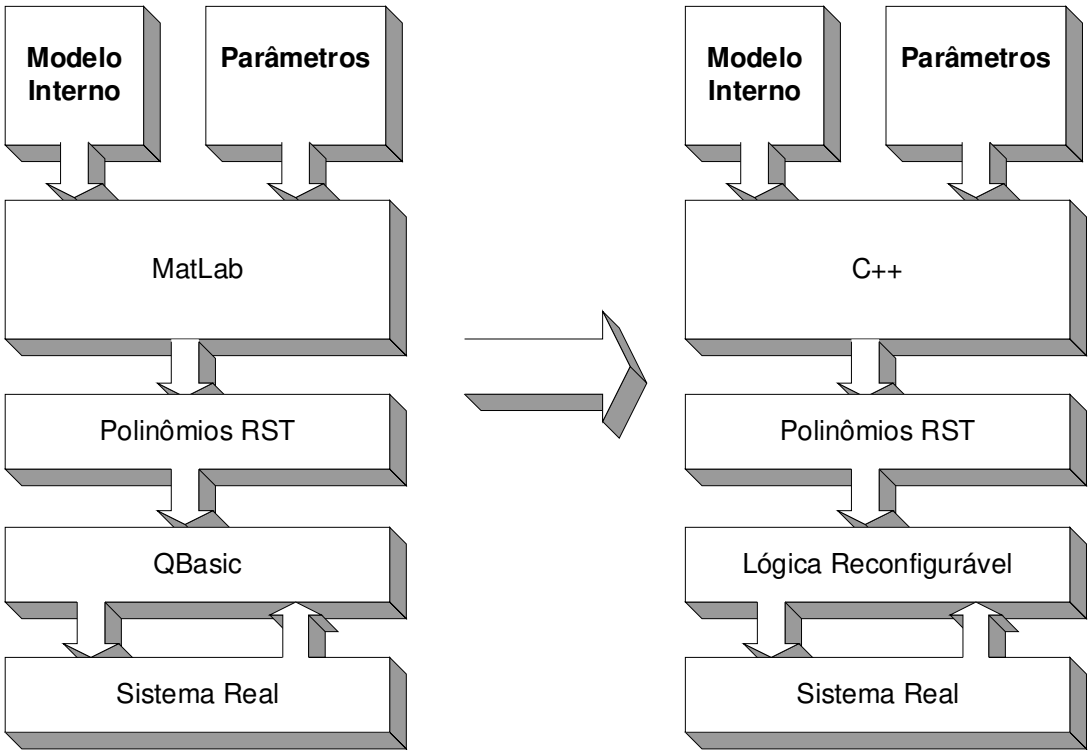


Figura 42 – Diagrama de blocos representando duas implementações do controlador GPC: usando lógica convencional e usando lógica reconfigurável.

Como foi apresentado anteriormente, o polinômio RST é uma forma equivalente do controlador GPC proposto, onde utilizando os mesmos procedimentos um controlador PID poderá também ser facilmente escrito nessa forma, dando assim um caráter genérico na implementação em computação reconfigurável. Os coeficientes deste polinômio são gerados em um programa específico no nível de controle supervisor e são passados como parâmetros de operação para o nível de controle embarcado, onde o polinômio RST resultante pode ser implementado em lógica reconfigurável, conforme é apresentado na Figura 43

O Controle Preditivo apresenta a vantagem de permitir o tratamento de sistemas não lineares e variantes no tempo. Sua principal desvantagem está no tempo de execução de seus algoritmos, o que inviabiliza sua aplicação em sistemas com constantes de tempo muito pequenas. Por esta razão, a aplicação de controle preditivo se concentra em plantas químicas [Maia, 1994] [Silva, 1997], muito embora já haja um bom número de aplicações envolvendo motores elétricos [Dumur et al., 1992]. Dentro deste contexto é razoável buscar novas técnicas de implementação visando diminuir o tempo de execução dos algoritmos de controle preditivo.

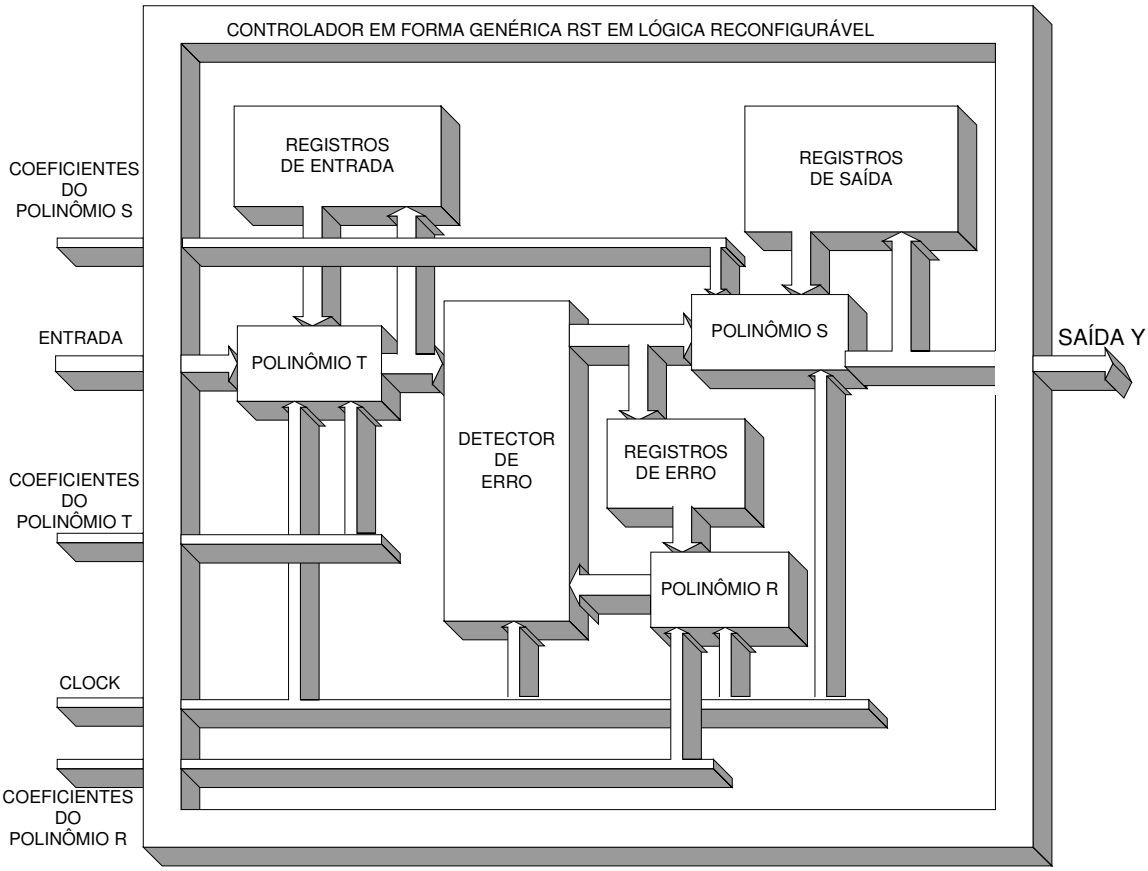


Figura 43 – Diagrama de blocos representando a implementação de um controlador na forma genérica RST usando lógica reconfigurável.

O conjunto de módulos associados ao controlador GPC implementados no nível de controle supervisor e no nível de controle embarcado é chamado de programa do controlador GPC. Assim,

o programa do controlador GPC efetua uma série de tarefas em função de dados de entrada necessários para a operação como controlador. As tarefas principais são:

- Gerar os coeficientes dos polinômios R, S e T do controlador RST em função de um modelo dado, uma função de transferência do sistema a ser controlado, bem como dos horizontes de predição.

- No caso do sistema ser considerado variante no tempo, gerar um modelo segundo um processo de estimação. Esta mudança de parâmetros ao longo do tempo deve ser feita no bloco de geração de modelos, dentro do nível de controle supervisor ou remoto.

- Gerar a saída de controle para cada instante de amostragem, levando em conta o modelo do sistema, bem como a referência e a realimentação do sistema.

- Gerenciar a aquisição de dados do meio externo bem como a geração de dados para o meio externo, fazendo eventuais correções associadas com as limitações das interfaces de *hardware*, a exemplo de saturações e adequações de escala. Estas considerações devem ser implementadas no nível de controle local.

Assim, o programa GPC é dividido em dois módulos principais: um associado aos coeficientes dos polinômios RST (módulo RST) e outro associado aos sinais de controle U, de referência W e de realimentação Y (módulo WUY). Estas estruturas são flexíveis e podem ser acessadas e modificadas ao longo da execução do programa. As saídas do módulo RST são modificadas pelos parâmetros de cálculo do polinômio RST. Esta parte do programa é realizada em linguagem C++ e pode ser vista no apêndice C. Já as saídas do módulo WUY são funções dos coeficientes do polinômio RST passados como parâmetros pelo nível de controle supervisor, dos sinais de controle e dos sinais oriundos dos sensores.

Considerando a flexibilidade dos parâmetros de entrada e do modelo adotado, os módulos a serem desenvolvidos devem levar em conta a necessidade de adaptação do programa a estas

variações. Por exemplo, a variação no horizonte de estimação do sistema pode provocar um aumento no grau de um ou mais dos polinômios R, S e T. Isto leva a modificações na equação a diferenças que calcula a saída de controle a cada instante de tempo e deve ser considerado no programa.

4.5 Conclusão

A implementação de estratégias de controle em lógica reconfigurável apresenta uma relativa dificuldade se comparada com a implementação em lógica convencional. A manipulação de variáveis que devem ser memorizadas, com valores anteriores de entradas ou saídas, é feita de forma rápida em lógica convencional. Já em lógica reconfigurável, cuidados especiais devem ser tomados, como pode ser observado na implementação do controlador PID, onde as variáveis discretas de entrada e saída devem ser tratadas em blocos específicos (registros de erro e registros de saída). Em contrapartida, a velocidade de resolução do algoritmo cresce de forma significativa. Isto permite a implementação de laços rápidos em diversos sistemas de controle. Além disto algumas famílias de PLD permitem a reprogramação em um tempo suficientemente curto para permitir a alteração de uma estratégia de controle durante a execução de uma determinada tarefa, adaptando o controle a mudanças no ambiente, por exemplo.

Algumas considerações devem ser tecidas quanto às implementações de controladores realizadas em lógica reconfigurável:

- Limites de saturação no bloco detector de erro: O contador utilizado para mensurar é um contador circular, logo, necessita de restrições para interromper a contagem quando os limites superior ou inferior são atingidos.

- Erros de precisão do *encoder*: A precisão na velocidade final é afetada pela precisão do *encoder* utilizado.
- Erros de quantização do contador: O número de bits utilizado implica na precisão obtida.

- O número de bits utilizado pelas funções aritméticas implementadas em lógica reconfigurável, a exemplo de somadores e multiplicadores na implementação do PID digital, afetam a precisão dos resultados obtidos. Por outro lado, a utilização de operações com um maior número de bits implica em uma maior demanda de recursos. Uma solução de compromisso entre precisão e recursos utilizados deve ser encontrada.

- Sentido Bidirecional: No projeto do módulo detector de erro, por simplificação, foi considerado que o motor desenvolve rotação apenas em um sentido.

- Controlador proporcional: No bloco detector de erro é implementado um controlador proporcional, onde o sinal de entrada CLOCK altera a constante de proporcionalidade. Por esta razão, um erro em regime permanente na velocidade é esperado.

- Erro de partida no bloco detector de erro: Devido à natureza do controlador é necessário, no caso do motor parado, detectar um sinal de TRAJETÓRIA sem um sinal correspondente de ENCODER e a partir disso, gerar um sinal de partida (estabelecimento de uma condição inicial).

- A utilização da implementação de controladores na forma RST permite propor uma estrutura genérica de implementação de controladores. Isto permite inclusive a modificação dos coeficientes dos polinômios RST de acordo com eventuais modificações nos requisitos do sistemas. A determinação destes coeficientes é feita através no bloco de geração de modelo presente no nível de controle supervisor.

No capítulo 5 são apresentadas três implementações do ambiente proposto aplicadas a sistemas embarcados práticos.

Capítulo 5

Análise Experimental

Este capítulo apresenta a aplicação do ambiente proposto a três sistemas embarcados móveis: um robô móvel experimental, um robô móvel de competição e uma cadeira de rodas automatizada. O ambiente descrito no capítulo 3 é aplicado nestas implementações, sendo adicionados novos blocos de hardware e software dentro do conceito do sistema aberto proposto. A escolha destes sistemas práticos para validação do ambiente é justificada por fatores como facilidade de implementação, capacidade de agregar sensores e atuadores em diferentes tarefas, possibilidade de expansão e demanda social.

5.1 Robô Móvel Experimental

O robô móvel experimental foi desenvolvido como plataforma de testes para diversos sensores e atuadores, bem como para validação a algoritmos de controle que utilizam estes sensores e atuadores. Foi desenvolvido pelo Laboratório de Automação Integrada e Robótica da Faculdade de Engenharia Mecânica da UNICAMP. Apresenta uma configuração descrita pela Figura 44, onde se pode observar que os movimentos de rotação e translação são efetuados por dois motores DC acoplados através de redutores às rodas de tração do robô. Esta configuração é bastante utilizada por robôs móveis em diversas aplicações na literatura especializada [Borenstein et al., 1996]. Este robô móvel foi desenvolvido visando aplicações didáticas e de suporte à

pesquisa em dissertações de mestrado e teses de doutorado.

Dentro dos sistemas embarcados, os robôs móveis apresentam-se como objeto de estudo atual. Alguns pontos justificam o aprofundamento do estudo desta classe de sistemas:

- Possibilidade de aplicação em diversas áreas de conhecimento, permitindo, por exemplo, o estudo de interfaces, sensores, programação, controle, modelagem de sistemas, mecânica e eletrônica embarcadas, entre outros. Por ser um produto da colaboração de diversas áreas de conhecimento é uma ferramenta acadêmica poderosa.

- Mecânica relativamente fácil de ser implementada. Devido às tarefas associadas aos robôs móveis, a margem de erro permitida no projeto mecânico é maior do que em um manipulador robótico, por exemplo.

- Aplicação comercial. Os robôs móveis apresentam um grande número de aplicações comerciais: inspeção, segurança, manutenção, mapeamento, etc.

- Tema atual. É uma área de grande interesse comercial e acadêmico, apresentando ampla divulgação em eventos e congressos.

- Flexibilidade. Possibilidade de adaptação a diferentes tarefas, usando diferentes sensores e atuadores e várias estratégias de controle.

As características deste projeto são:

- Baixo custo. Através de uma mecânica simples e da utilização de sensores e atuadores disponíveis no mercado é possível a implementação do projeto com um custo reduzido. Isto facilita a reprodução do projeto em ambiente acadêmico.
- Controle por RF dos motores envolvidos nos movimentos de rotação e translação do robô móvel.

- Alcance do sistema deve ser limitado às operações dentro dos ambientes internos dos laboratórios.
- Ser uma plataforma para consolidação de conhecimento em diversas áreas de ensino e pesquisa (modelagem, controle, automação, acionamento, sensores, transmissão de dados, e engenharia de software).
- Fazer uso do ambiente desenvolvido, distribuindo as diversas ações de controle nos níveis propostos.
- Possibilidade de usar novas combinações de sensores, tanto em nível qualitativo, quanto em nível quantitativo.
- Possibilidade de utilizar/testar diferentes modelos de supervisão e controle.

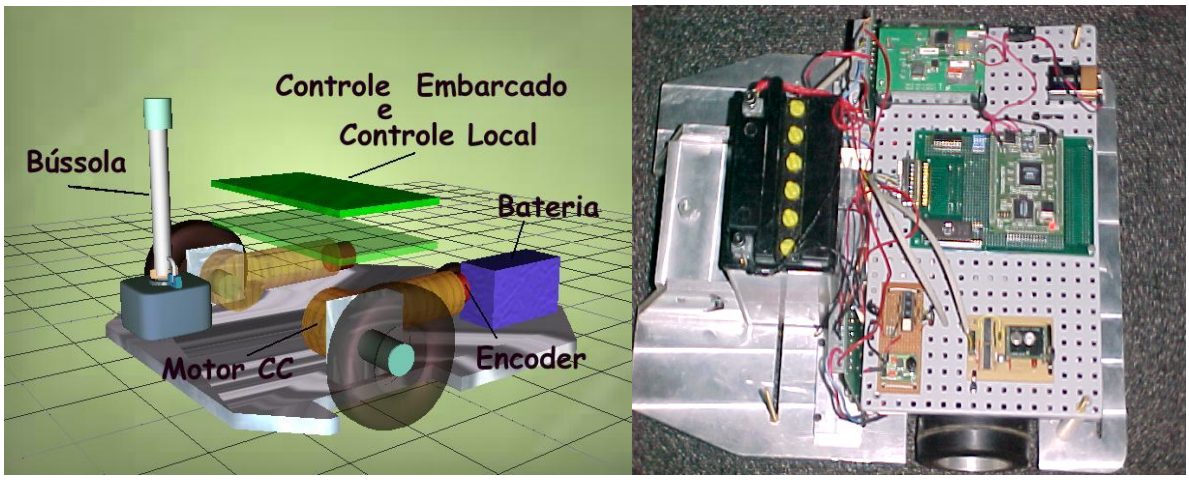


Figura 44 - Representação dos principais componentes do robô móvel experimental e sua implementação.

A Figura 45 apresenta o digrama de blocos da implementação do robô móvel experimental usando a arquitetura do ambiente proposto. Os três níveis de controle estão presentes. O nível de controle supervisor ou embarcado apresenta uma interface com o usuário que permite a entrada de comandos a serem executados no nível de controle embarcado, bem como a visualização de parâmetros de operação do robô móvel. O nível de controle embarcado executa o tratamento dos comandos recebidos do nível de controle supervisor e trata as

informações recebidas do nível de controle local. O nível de controle local apresenta interfaces para a eletrônica de potência dos motores e para os sensores utilizados: *encoders* e bússola. A comunicação entre o nível de controle embarcado e o nível de controle supervisor é realizada através de um enlace bi-direcional de RF. Os blocos de comunicação necessários para esta operação acham-se descritos no capítulo 3.

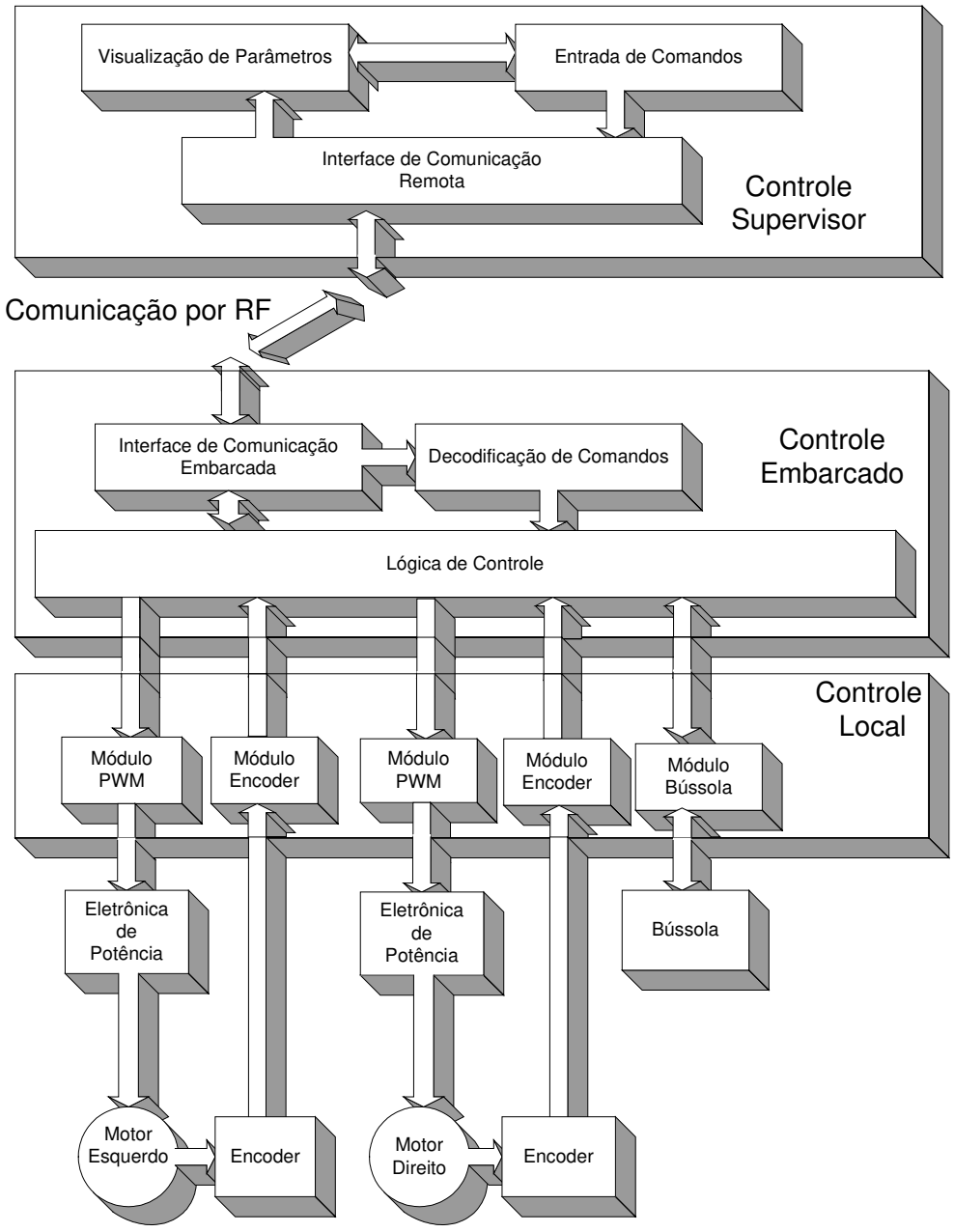


Figura 45 - Diagrama dos blocos usados na implementação do robô móvel experimental distribuídos segundo a arquitetura do ambiente proposto.

5.1.1 Nível de Controle Supervisor

No nível de controle supervisor, além da interface de comunicação por RF, são implementados dois blocos: o bloco de visualização de parâmetros e o bloco de entrada de comandos. Estes blocos são construídos em linguagem Visual Basic e fazem uso das funções de comunicação descritas no capítulo 3. A Figura 46 apresenta a janela de interface com o usuário resultante da implementação destes blocos, onde os parâmetros relevantes à operação do robô podem ser vistos. Nesta mesma janela, através de botões de controle, é possível enviar comandos para o robô móvel experimental.

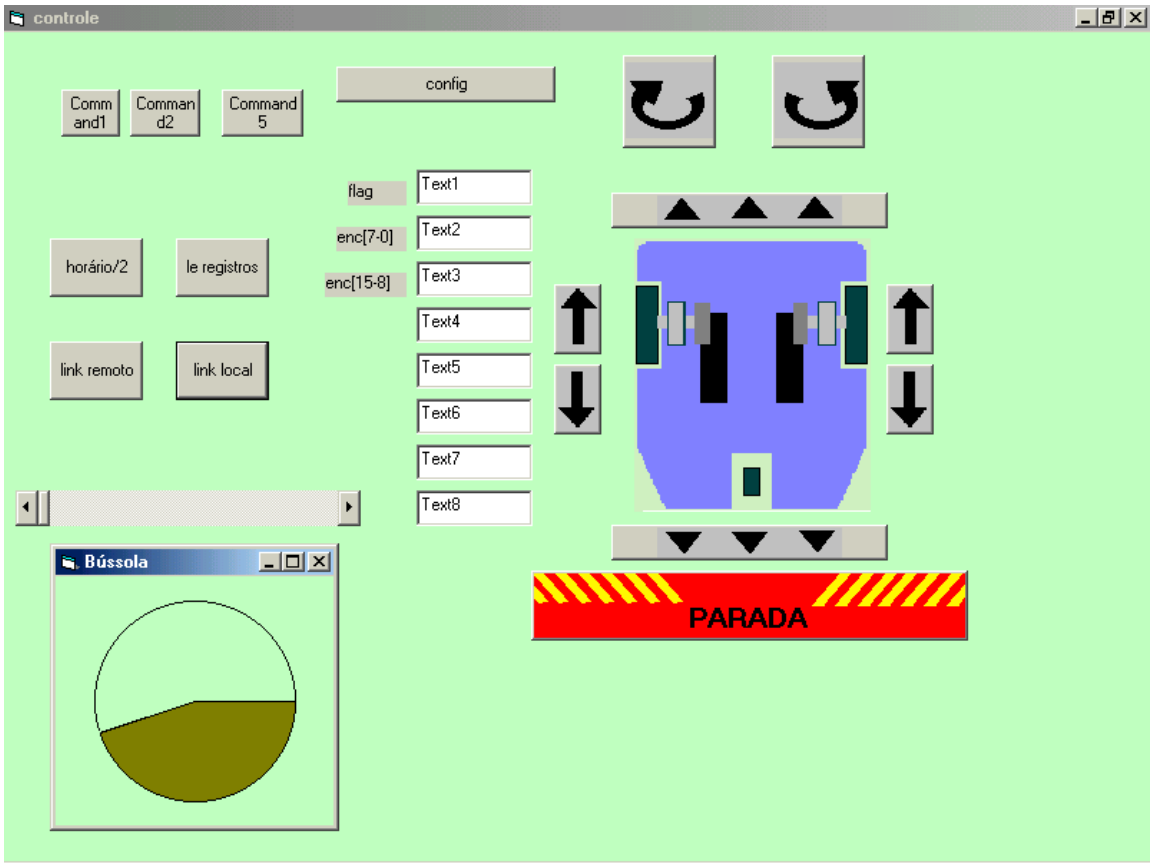


Figura 46 - Interface com o usuário presente no nível de controle supervisor.

5.1.2 Nível de Controle Embarcado

No nível de controle embarcado, além da interface de comunicação embarcada (abordada no Capítulo 3), são implementados dois blocos principais: o bloco de decodificação de comandos e o bloco de lógica de controle.

O bloco de decodificação de comandos decodifica campos de dados recebidos pela interface de comunicação embarcada, realizando diferentes ações em função dos dados recebidos. Já bloco de lógica de controle gera os sinais para os blocos de controle dos motores em função de comandos enviados pelo nível supervisor e das informações de sensores. Eventuais estratégias de controle são implementadas neste bloco.

5.1.2.1 Bloco de Decodificação de Comandos

A transferência de dados e comandos entre o nível de controle supervisor e o nível de controle embarcado utiliza uma estrutura de dados que permite operações de escrita e leitura em uma memória interna do sistema embarcado. Nesta memória interna são armazenados parâmetros de operação e variáveis associadas a sensores e atuadores. Esta memória interna é composta por 256 bytes, do endereço 0 até o endereço 255. O bloco de decodificação de comandos permite traduzir seqüências de bytes recebidos em padrões que são utilizados para realizar três tipos de eventos no nível de controle embarcado: comandos diretos, operação de escrita em memória e operação de leitura em memória. A Figura 47 apresenta as possíveis seqüências de bytes com os respectivos eventos associados. Na Figura 47 é possível observar três tipos de bytes:

- Byte de Comando – Todo evento é iniciado por um byte de comando. Se este comando é igual ao valor decimal 101, então trata-se de comando para uma operação de leitura em memória. Neste caso um campo adicional, o byte de endereço, é requerido. Se este comando é igual ao valor decimal 201, então é o comando para uma operação de escrita em memória e dois campos adicionais são requeridos, o byte de endereço e o byte de dado. Qualquer outro

valor enviado (diferente de 101 e 201) é considerado um comando direto. Neste caso, é iniciado um evento no nível de controle embarcado que não leitura ou escrita em memória.

- Byte de Endereço – No caso do byte de comando ser igual ao valor 101 ou 201 ocorre uma operação de escrita ou leitura em memória, demandando um byte adicional com o endereço de memória.

- Byte de Dado – No caso do byte de comando ser igual ao valor 201 ocorre uma operação de escrita em memória, demandando, além do byte de endereço, um byte adicional de dado.

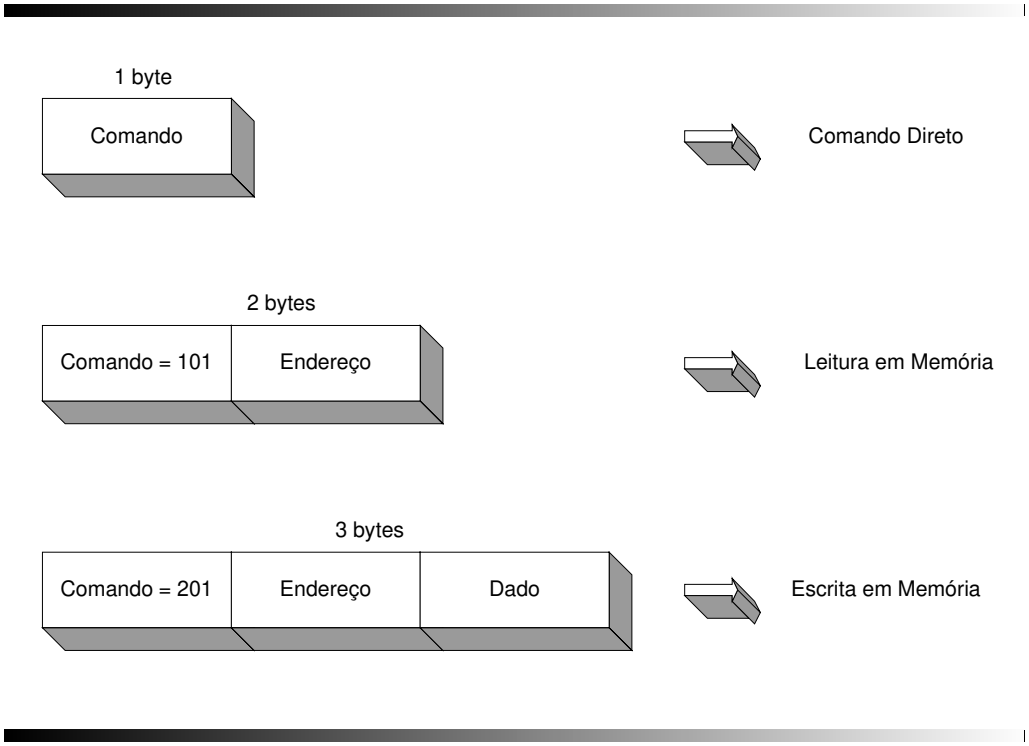


Figura 47 - Possíveis seqüências de bytes que o bloco de decodificação de comandos pode receber.

A Figura 48 apresenta a implementação do bloco de decodificação de comandos, aqui chamdo de “dec_com1” onde se pode observar os campos de comando, endereço e dados como sinais de saída. Além disto, como saída do bloco “dec_com1” pode-se observar sinais de controle

necessários para viabilizar as operações de escrita e leitura em memória (h_escrita e h_leitura). O bloco “dec_com1” está ligado aos blocos de comunicação “TX8” e “dec_rx”.

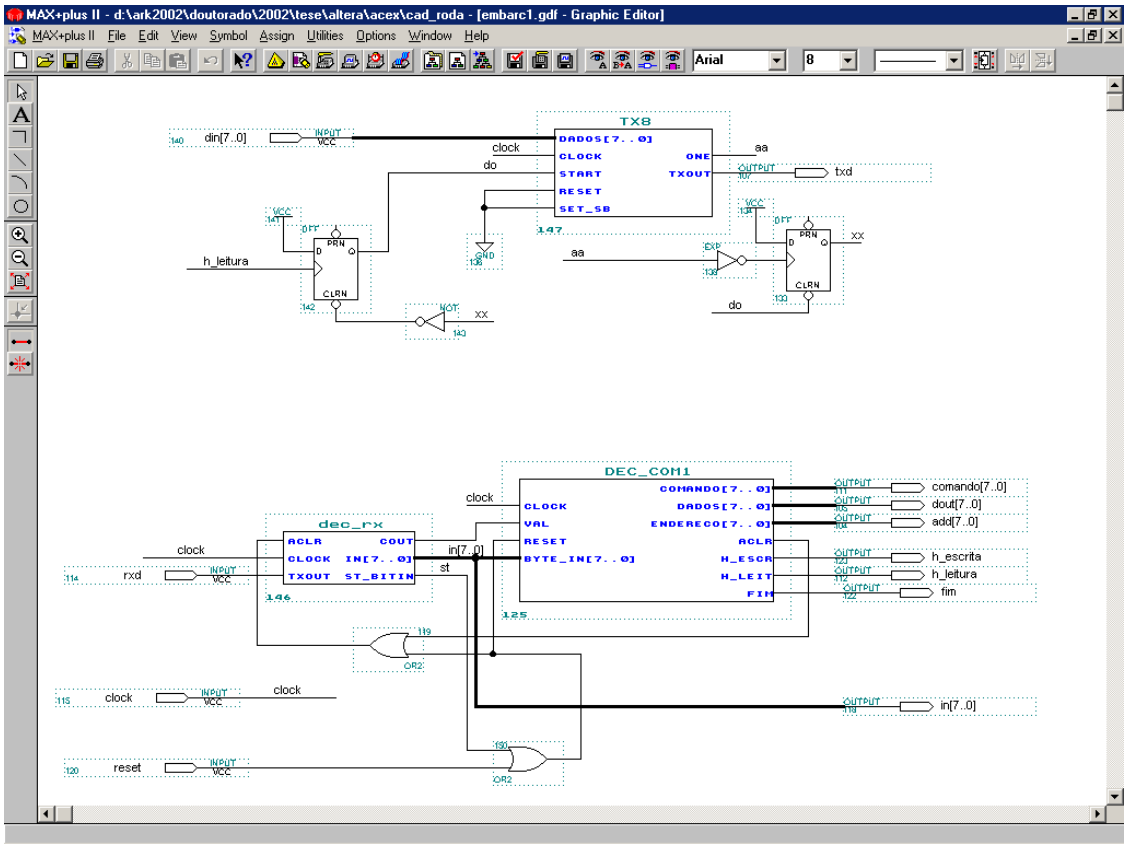


Figura 48 - Implementação em linguagem gráfica do bloco de decodificação de comandos.

5.1.2.2 Bloco de Lógica de Controle

O bloco de lógica de controle é responsável pelo tratamento dos comandos enviados pelo nível de controle supervisor, levando em conta os parâmetros e variáveis armazenados na memória interna. Além disto executa eventuais estratégias de controle do robô móvel experimental. O bloco de lógica de controle atua diretamente sobre os módulos implementados no nível de controle local. Também neste bloco é feita a tradução dos comandos diretos obtidos do bloco de decodificação de comandos.

Vários exemplos de implementação do bloco de lógica de controle podem ser apresentados. Na Figura 49 é apresentado de modo parcial a implementação gráfica de uma operação de escrita em uma memória e conseqüente evento sobre um módulo no nível de controle local. A escrita no endereço de memória zero (decodificado pelo bloco "com0") provoca a alteração nos valores de entrada do bloco "gerador". Outro parâmetro de entrada do bloco "gerador" (ina) é fornecido pelo conteúdo de outro endereço de memória (não representado na Figura 49). Assim, através de simples operações de escrita em memória é possível comandar a velocidade de cada motor do robô móvel individualmente.

Outro evento possível é a utilização de comandos diretos. Os comandos diretos podem iniciar a execução de máquinas de estado pré-programadas que fornecerão os parâmetros de entrada do bloco "gerador", permitindo a execução de tarefas. Um exemplo é a execução de uma trajetória pré-definida ou de uma busca de orientação através do sensor bússola.

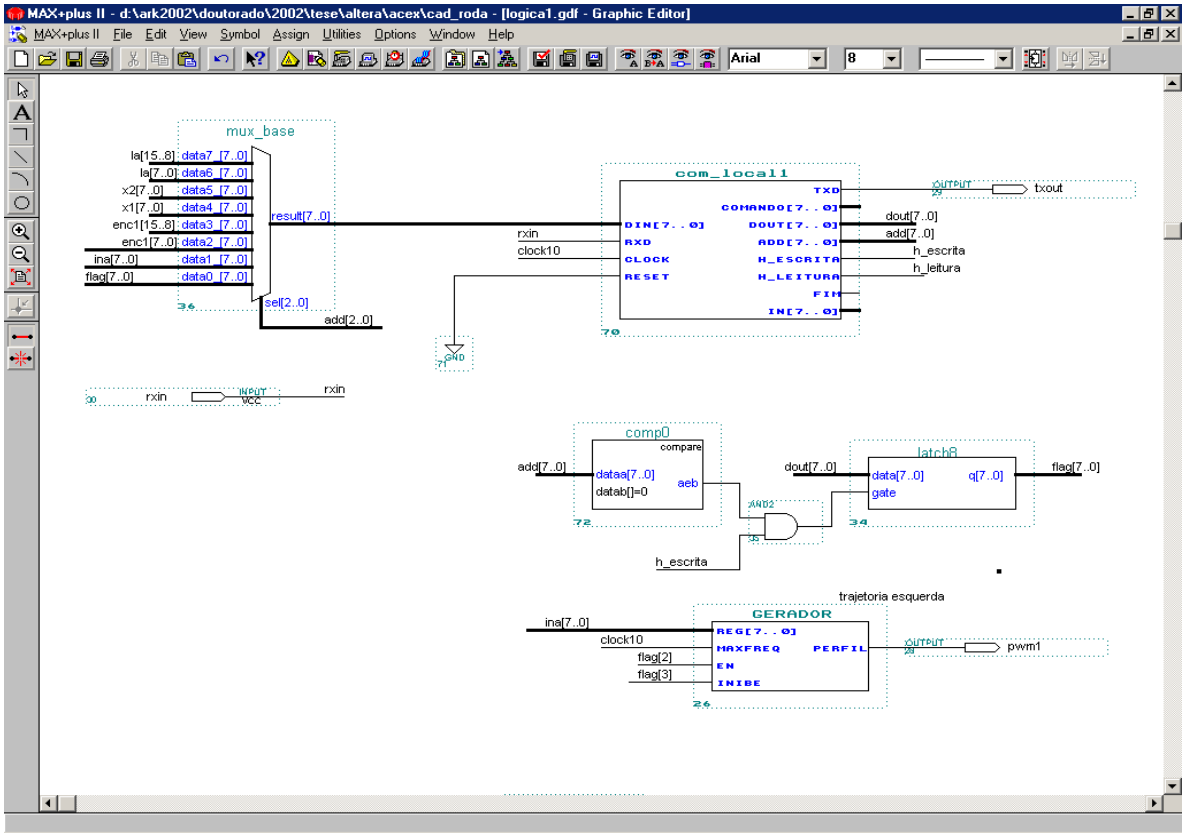


Figura 49 - Representação parcial de implementação em linguagem gráfica do bloco de lógica de controle.

5.1.3 Nível de Controle Local

No nível de controle local são construídas as interfaces com os sensores e atuadores usados no robô móvel experimental. São considerados três módulos distintos: o módulo PWM, o módulo da bússola e o módulo do *encoder*.

5.1.3.1 Módulo de Interface PWM

O módulo de interface PWM ou simplesmente módulo PWM gera diferentes perfis PWM para a eletrônica de potência utilizada. A Figura 50 representa as entradas e saídas deste bloco, implementado em linguagem VHDL . O código completo encontra-se no Apêndice D.

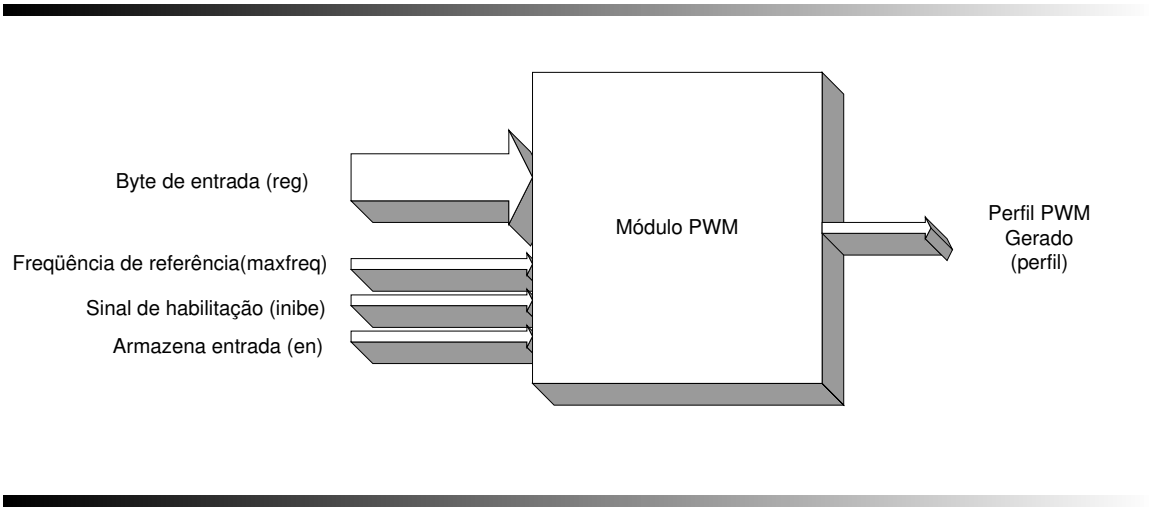


Figura 50 - Representação do módulo PWM com as respectivas entradas e saídas.

Uma vez armazenado o valor do byte de entrada no módulo PWM (sinal ‘en’ = 1) o perfil é gerado indefinidamente com um período proporcional à freqüência de referência e um ciclo de trabalho proporcional ao valor do byte de entrada. Este comportamento pode ser observado na Figura 51, que mostra uma simulação do módulo PWM para uma freqüência de referência de 500

kHz e dois valores diferentes do byte de entrada (reg[7..0] = 200 e reg[7..0] = 100).

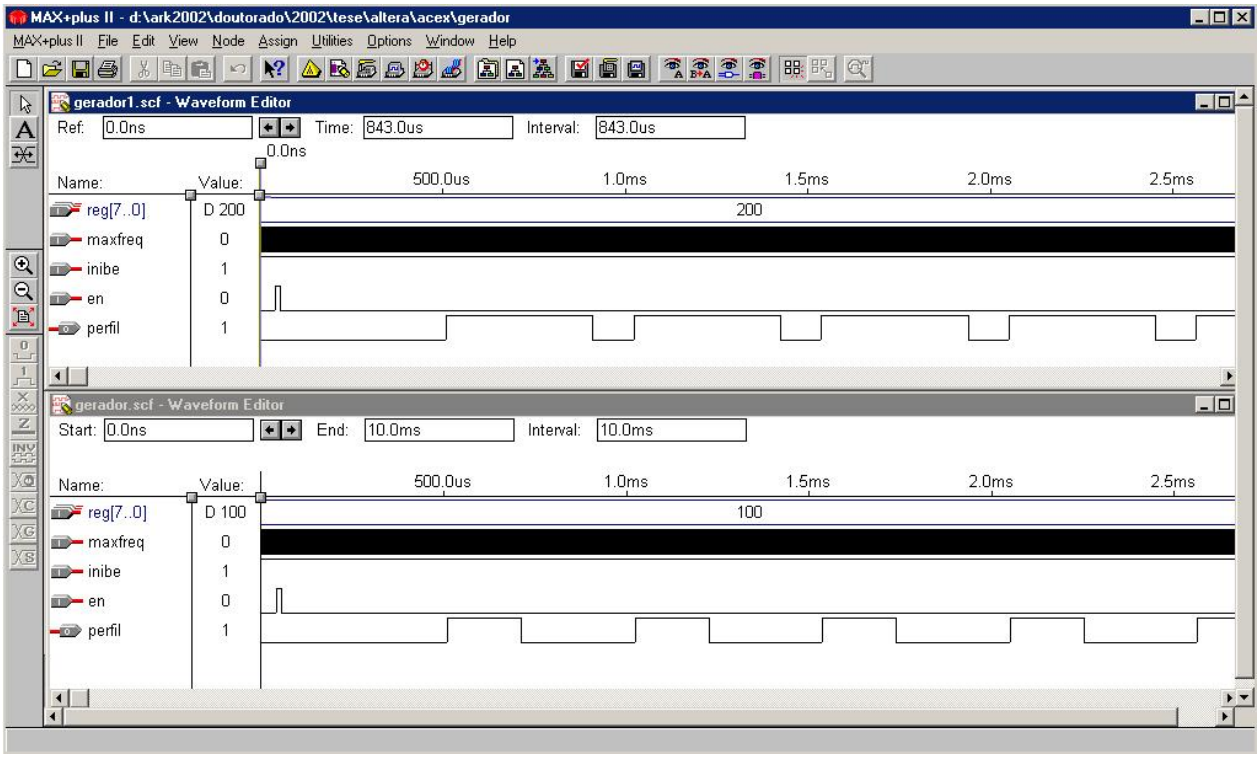


Figura 51 - Simulação do módulo PWM para dois valores da byte de entrada reg[7..0].

O módulo PWM é de grande utilidade no controle de velocidade de motores CC, desde que seja utilizada a freqüência de referência adequada para a constante de tempo de cada motor. Nos motores empregados no robô móvel experimental é usada uma freqüência de referência tal que o período do perfil PWM é de 500 μs.

5.1.3.2 Módulo de Interface da Bússola

A utilização de bússolas digitais como sensores auxiliares na determinação do posicionamento de robôs móveis pode ser vista em diversos trabalhos na literatura, como por exemplo em [Borenstein et al., 1996]. Esta é uma alternativa de implementação de uma bússola digital de baixo custo. Através do melhor aproveitamento de um sensor de campo magnético bastante conhecido e da utilização de dispositivos lógicos programáveis para controle e

tratamento dos dados deste sensor é possível obter o deslocamento de um eixo de referência horizontal do robô móvel em relação à orientação do campo magnético terrestre. São apresentados os diagramas de blocos detalhando a operação do sistema bem como os programas em VHDL e linguagem gráfica. Uma interface com o usuário é implementada em Visual Basic, permitindo a calibração da bússola em relação ao eixo de referência do robô móvel. A Figura 52 apresenta uma representação da implementação da bússola digital.

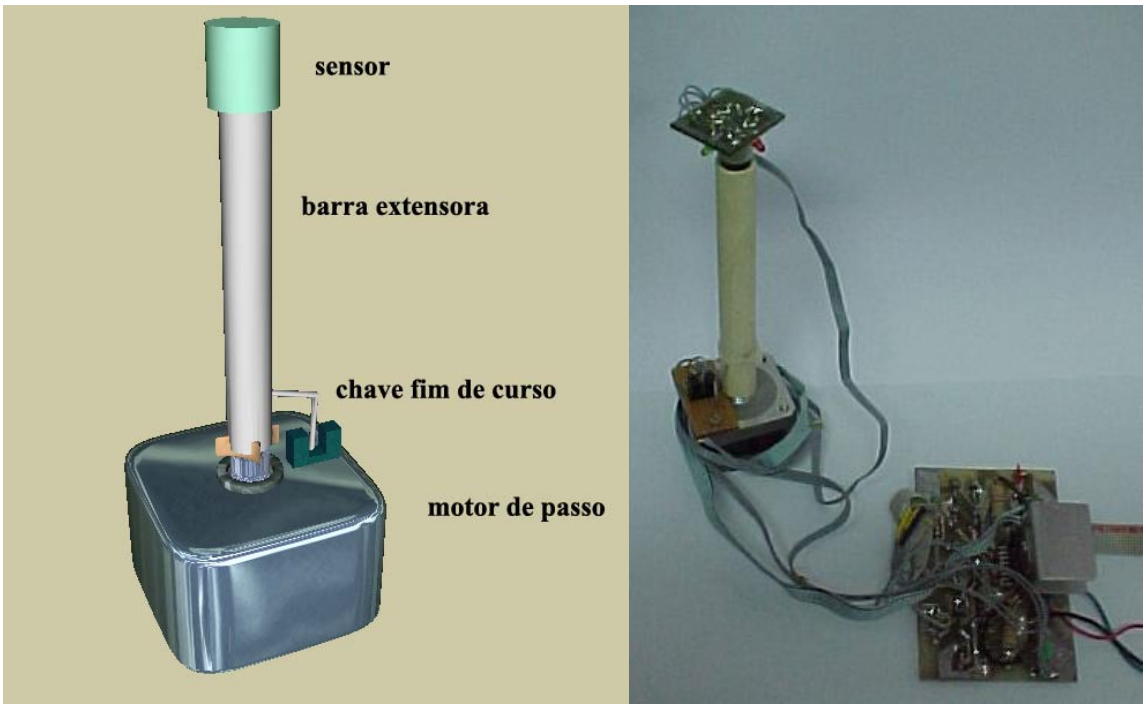


Figura 52 - Descrição da disposição do sensor em relação ao motor de passo utilizado.

Os seguintes pontos são ressaltados:

- Utilização de um motor de passo para obter uma maior resolução na orientação em relação ao norte magnético.

- Utilização de lógica reconfigurável para controle do motor de passo e tratamento dos sinais recebidos do sensor, ao nível do controle local.

O sistema implementado integra o sensor Disnmore 1490, um motor de passo e uma chave

fim de curso para fornecer informação de deslocamento angular de um eixo de referência de um robô móvel. O sensor Dinsmore 1490 [Dinsmore, 2002] é uma solução de baixo custo para implementar esta bússola digital. Contudo, se aplicado como proposto pelo fabricante, este sensor tem apenas a capacidade de informar a orientação de oito pontos cardeais. Esta implementação é uma modificação na aplicação e no tratamento deste sensor.

Como este deslocamento angular é medido em relação a uma chave fim de curso é necessário fazer uma calibração mecânica em relação ao eixo de referência do robô, ou seja, posicionar a chave corretamente em relação a este eixo.

Uma representação simplificada dos blocos implementados pode ser vista no diagrama de blocos da Figura 53.

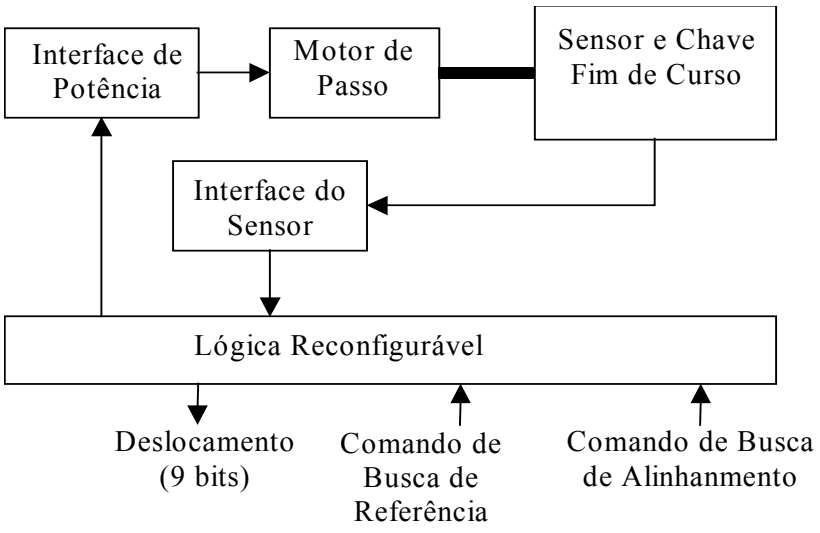


Figura 53 **-** Diagrama de blocos representado o sistema de posicionamento implementado.

O bloco em lógica reconfigurável é responsável pelo tratamento do motor de passo, habilitando o deslocamento do mesmo nos sentidos horário e anti-horário para execução das duas tarefas associadas: a busca da chave fim de curso e a busca do alinhamento com o campo magnético. Quando a chave fim de curso é encontrada um contador interno neste bloco é zerado.

5.1.3.3 Mapa do sensor

O sensor Dinsmore 1490 permite a determinação de oito setores de aproximadamente 45° onde os pontos cardeais podem ser representados segundo a Figura 54. Esta representação é feita através de quatro chaves (chamadas de L1, L2, L3 e L4) que são sensíveis ao alinhamento com o campo magnético terrestre. Em função deste alinhamento pode-se obter a Tabela 6.

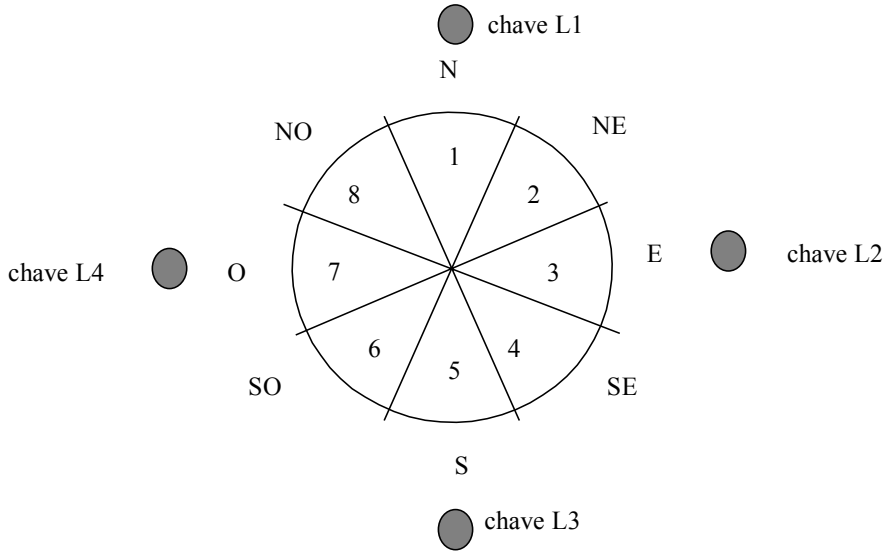


Figura 54 - Mapa do sensor Dinsmore 1490 em função do seu alinhamento com o campo magnético terrestre.

Tabela 6 – Estado de acionamento das chaves do sensor Dinsmore 1490 em função do alinhamento com o campo magnético terrestre segundo o mapa da Figura 54.

setor	L1	L2	L3	L4
1	fechada	aberta	aberta	aberta
2	fechada	fechada	aberta	aberta
3	aberta	fechada	aberta	aberta
4	aberta	fechada	fechada	aberta
5	aberta	aberta	fechada	aberta
6	aberta	aberta	fechada	fechada
7	aberta	aberta	aberta	fechada
8	fechada	aberta	aberta	fechada

5.1.3.4 Comandos do Sensor da Bússola

O algoritmo implementado leva em conta a execução de dois comandos:

- Busca de referência – este comando sempre move o motor em sentido horário até a chave fim de curso ser acionada.

- Busca de alinhamento – este comando move o motor em sentido horário ou anti-horário até encontrar a intersecção de dois setores. Foi escolhida a intersecção entre o setor 1 e o setor 2.

Um registrador de 9 bits (contador) é usado para armazenar o deslocamento angular do motor em relação à referencia. Como uma volta do motor de passo ocorre após 400 passos (0.9°), ou mais precisamente meios-passos, são necessários 9 bits para representar o deslocamento máximo possível. O registrador sofre incremento ou decremento de acordo com o deslocamento em sentido anti-horário ou em sentido horário respectivamente. Este registrador é zerado toda vez que a chave fim de curso é atingida. Além disto, o limite inferior e o limite superior deste registrador, ou seja, 0 e 400, não são ultrapassados, impedindo que o motor execute mais de uma rotação em torno de seu eixo, o que poderia provocar rompimento dos fios conectados ao sensor.

A máquina de estados que define a execução do comando de busca de zero é apresentada na

Figura 55. A chave fim de curso acionada é definida como Z = 1 e enquanto a chave fim de curso não é atingida como Z = 0. A variável comando = 1 implica em um evento de execução de comando de busca de zero. A variável contador reflete o estado do registro de 9 bits que representa o deslocamento do motor em relação ao ponto de referência representado pela chave fim de curso.

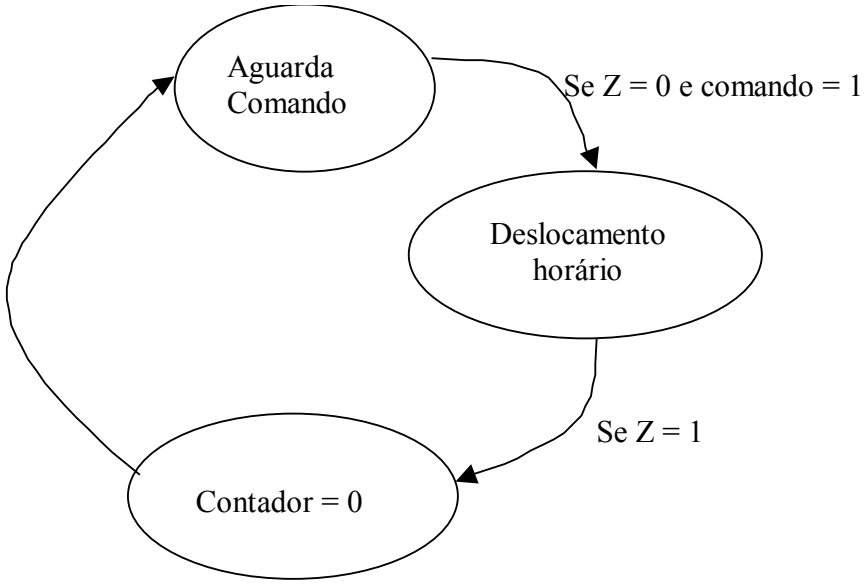


Figura 55 – Máquina de estados representado a execução do comando de busca de zero.

A máquina de estado da Figura 55 pode ser implementada usando linguagem gráfica, segundo a Figura 56. O comando é ignorado se a chave fim de curso já está acionada. Caso contrário, se um comando for recebido, o motor é acionado no sentido horário até a chave fim de curso ser acionada.

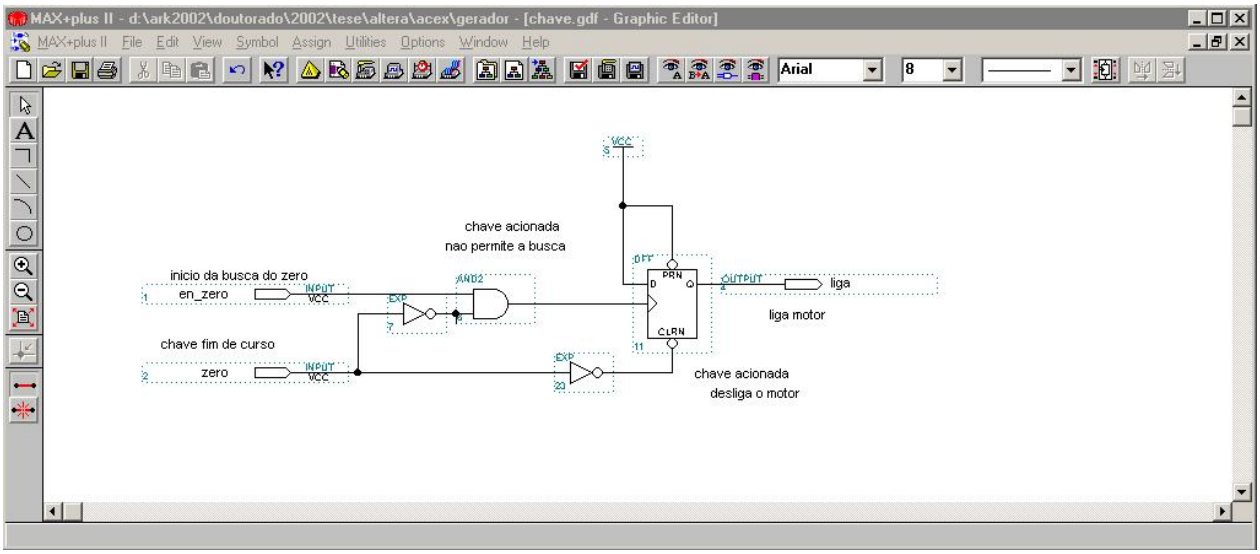


Figura 56 - Diagrama esquemático que implementa o comando de busca de zero.

Muito embora sejam necessárias as quatro chaves L1, L2, L3 e L4 para definir qual das oito áreas encontra-se orientada com o campo magnético terrestre, a transição entre o estado L1 = 1 e L1 = 0, ou vise-versa, pode ser determinada apenas com as chaves L1 e L2. A definição de onde está ocorrendo a transição de estado de L1, se na interface das áreas 2 e 3 ou se na interface das áreas 7 e 8, pode ser feita pelo estado da chave L2. Isto pode ser observado na Figura 57.

Por analogia, a determinação da ocorrência da transição de estado de uma chave (L1, L2, L3 ou L4) pode ser feita pela análise do estado de duas chaves adequadamente escolhidas. A escolha das chaves L1 e L2 e da interface entre as áreas 2 e 3 é aleatória, sendo outras opções perfeitamente possíveis.

Uma vez escolhida esta referência pode-se implementar a máquina de estados que define o comando de busca de alinhamento, a qual é representada na Figura 58.

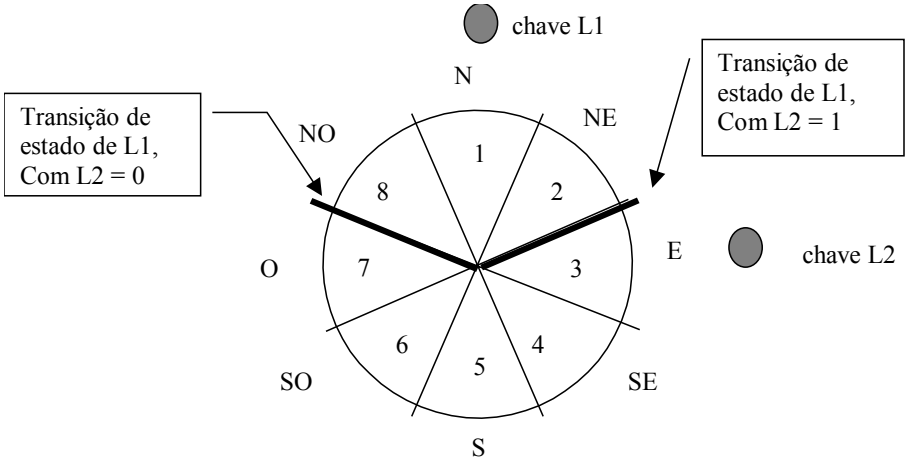


Figura 57 - Mapa do sensor Dinsmore 1490 em função do seu alinhamento com o campo magnético terrestre, enfatizando as duas possíveis transições de estado da chave L1.

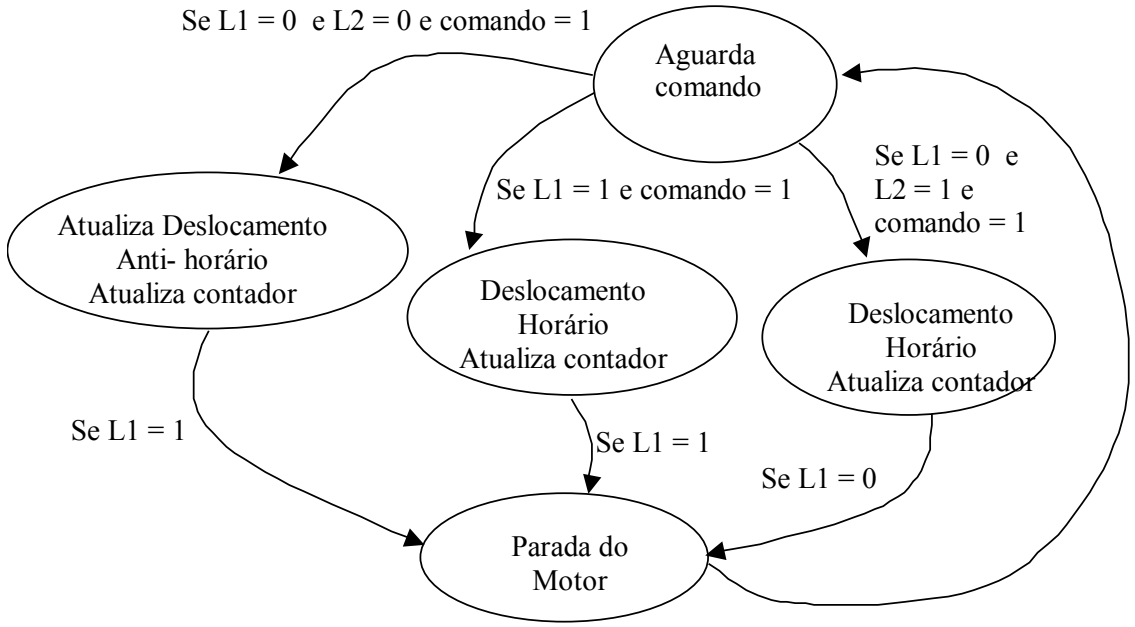


Figura 58 -Máquina de estados representado a execução do comando de busca de orientação.

A implementação da máquina de estado da Figura 58 pode ser feita usando linguagem gráfica, segundo a Figura 59. O comando é ignorado se a chave fim de curso já está acionada. Se um comando é recebido, o motor é acionado até a chave fim de curso ser ativada.

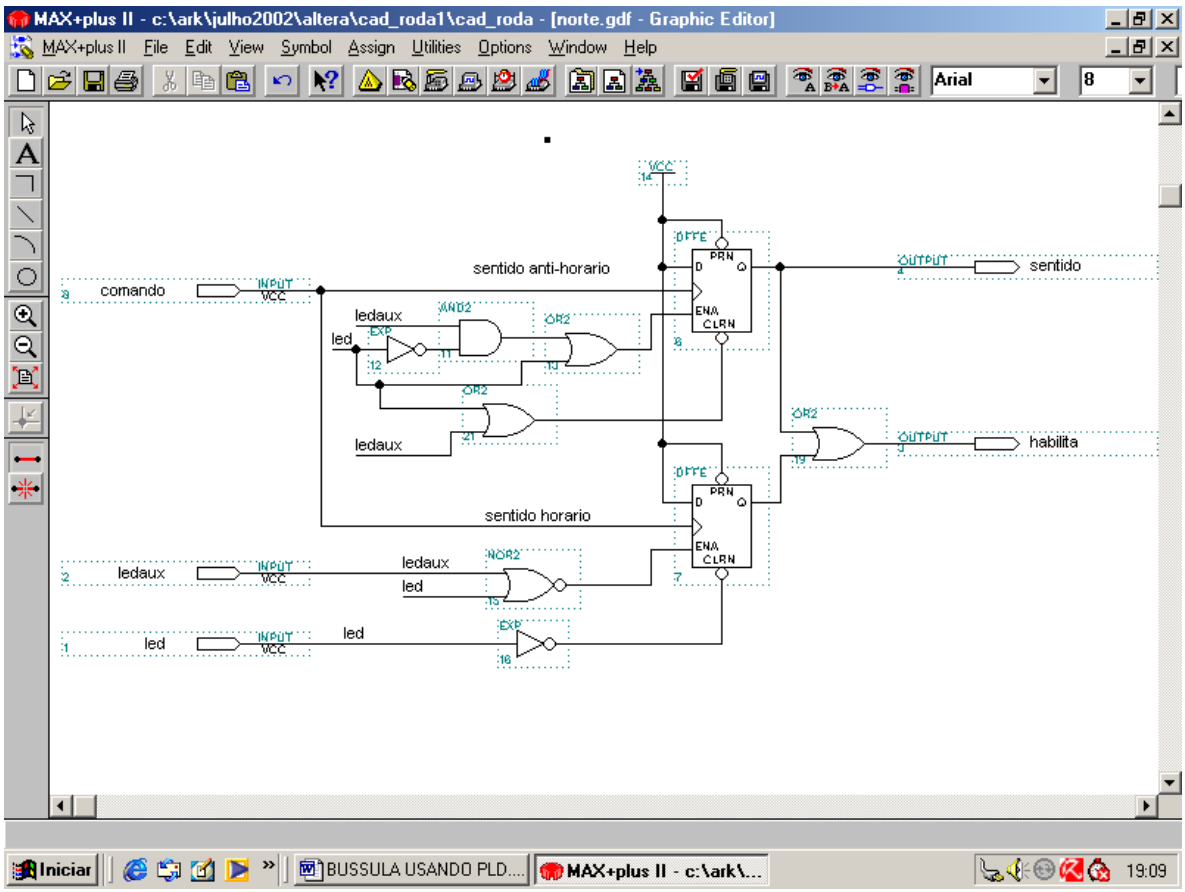


Figura 59 -Diagrama esquemático que implementa o comando de busca de orientação.

O diagrama esquemático da Figura 59 usa o estado da chave L1 ("led") e o estado da chave L2 ("ledaux') para determinar, através dos sinais "sentido" e "habilita", o sentido em que o motor deve ser acionado e quando o mesmo deve parar.

5.1.3.5 Controle do Motor de Passo da Bússola

O controle do motor de passo associado ao sensor da bússola é feito através de um bloco implementado em lógica reconfigurável. Este bloco, representado de forma simplificada na Figura 60, gera os sinais de controle para o circuito de potência que comanda as fases do motor de passo. Os sinais de controle provocam a rotação do motor no sentido horário ou anti-horário

com uma velocidade fixa (180° por segundo). O motor pode operar no modo de passo pleno ou de meio passo.

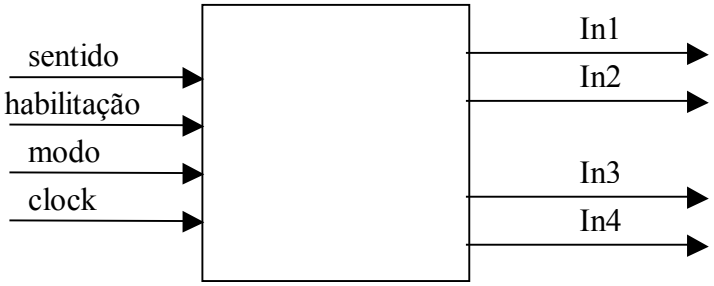


Figura 60 – Representação simplificada do bloco de comando do motor de passo usado na bússola digital.

A Tabela 7 e a Tabela 8 apresentam as seqüências de acionamento dos sinais de controle para os dois possíveis modos de operação, passo pleno e meio-passo respectivamente.

Tabela 7 – Modo de operação em Passo pleno

In1	In2	In3	In4
1	0	0	1
1	0	1	0
0	1	1	0
0	1	0	1

Tabela 8 – Modo de operação Meio-passo

In1	In2	In3	In4
1	0	1	0
1	0	0	0
1	0	0	1
0	0	0	1
0	1	0	1
0	1	0	0
0	1	1	0
0	0	1	0

Os seguintes comandos são considerados:

- Passo ou meio – passo.

- Sentido horário ou anti-horário.

- Habilitação ou não habilitação.

A implementação em linguagem gráfica do bloco da Figura 60 com as seqüências da Tabela 7 e da Tabela 8 pode ser observada na Figura 61.

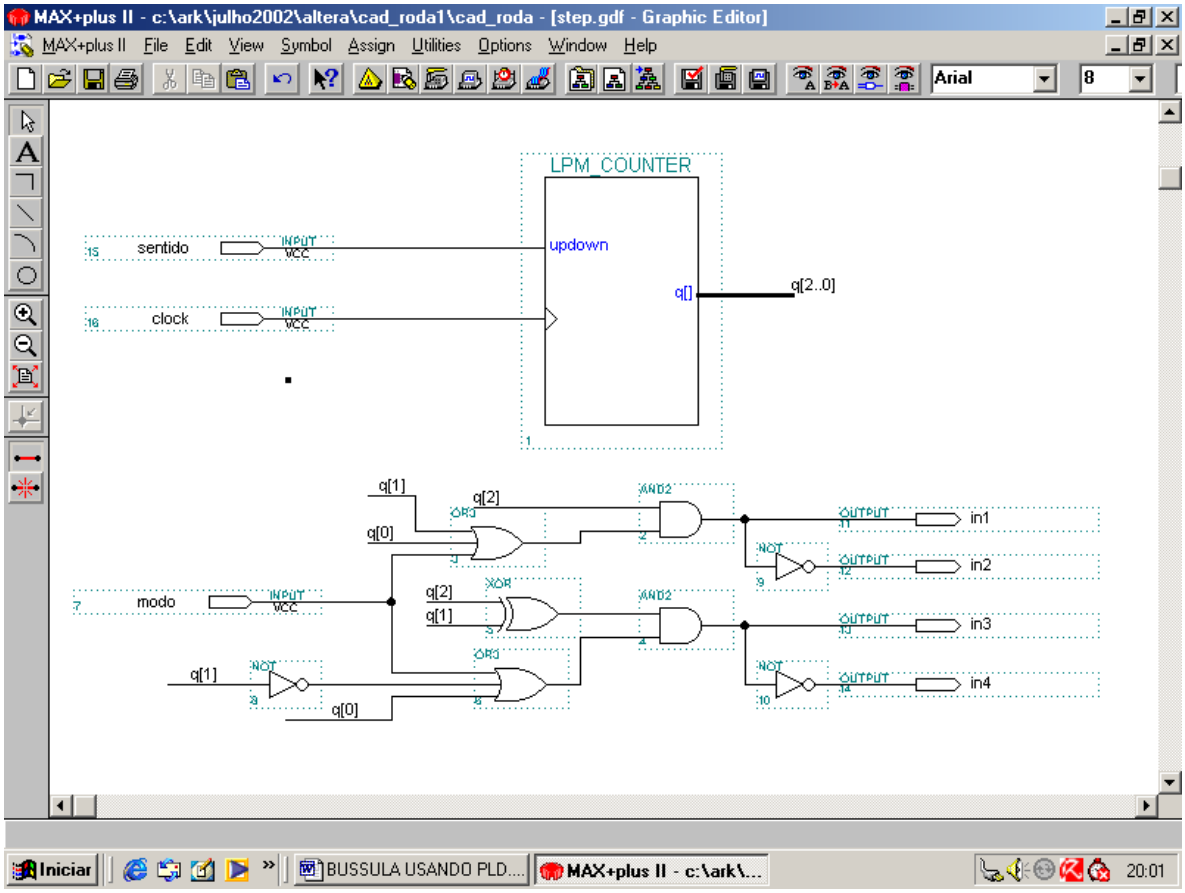


Figura 61 – Implementação em linguagem gráfica do bloco de controle do motor de passo.

5.1.3.6 Tratamento de Comandos da Bússola Digital

A Figura 62 figura mostra a implementação em linguagem gráfica do tratamento de comandos da bússola digital. Este bloco é implementado no nível de controle local.

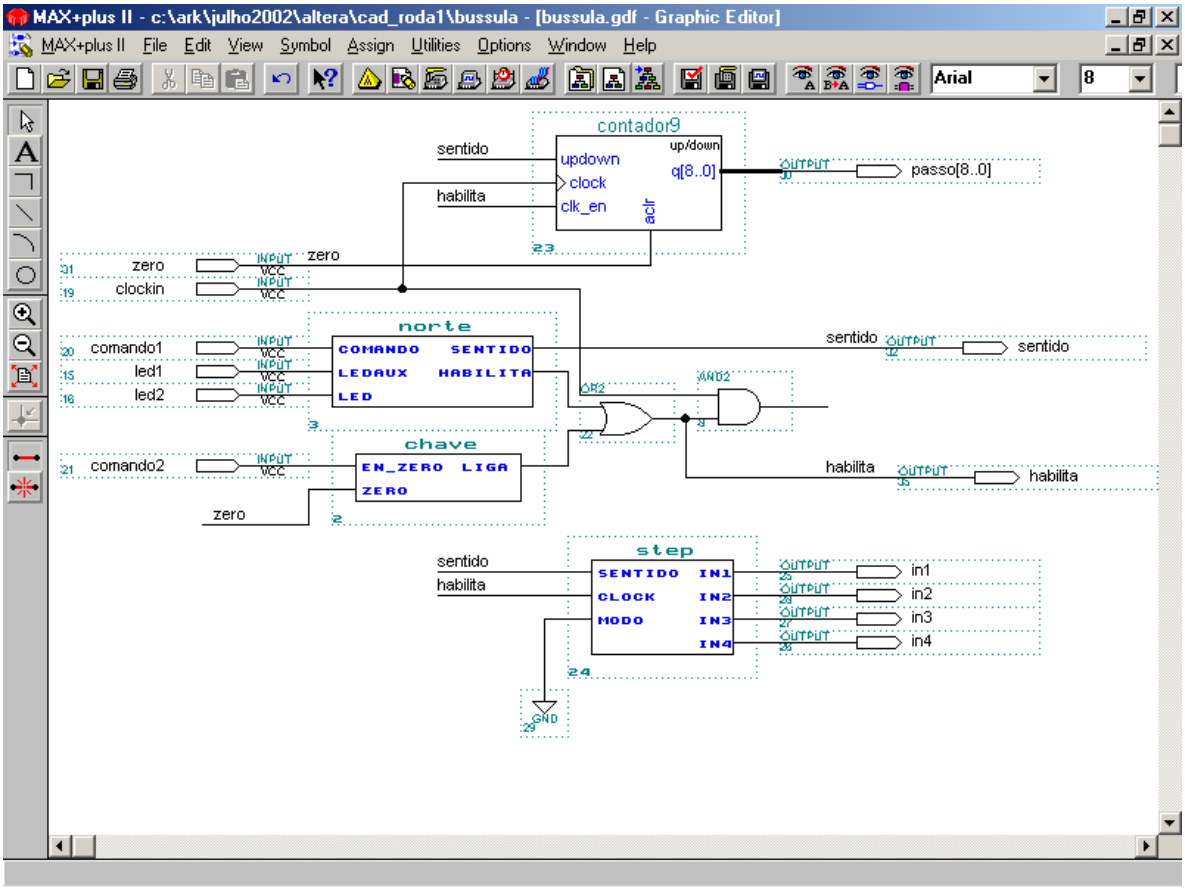


Figura 62 **-** Diagrama esquemático que implementa o controle do motor de passo.

Os seguintes blocos são detalhados:

- Bloco “contador9” - é um registrador de 9 bits que pode ser incrementado ou decrementado em função do sentido de rotação do motor, permitido valores entre 0 e 400. Este registrador é consultado periodicamente para verificação da orientação em relação à referência.

- Bloco “norte” - trata o comando de busca de orientação

- Bloco “chave” - trata o comando de busca de referência

- Bloco “step’ - gera os pulsos de comando para a interface de potência do motor de passo.

5.1.3.7 Simulações no MAX+PLUX II

Diversas simulações são feitas no ambiente de desenvolvimento Altera MAX+PLUX II. Estas simulações tornam possível a depuração dos diversos blocos implementados. Um exemplo é apresentado na Figura 63, que apresenta a simulação do bloco bússola em resposta ao comando de busca de referência. O sinal ´habilita´ em nível alto seleciona o sentido horário de movimento do motor (estando o sinal ´sentido´ em nível alto) ou o sentido anti-horário (estando o sinal ´sentido´ em nível baixo).

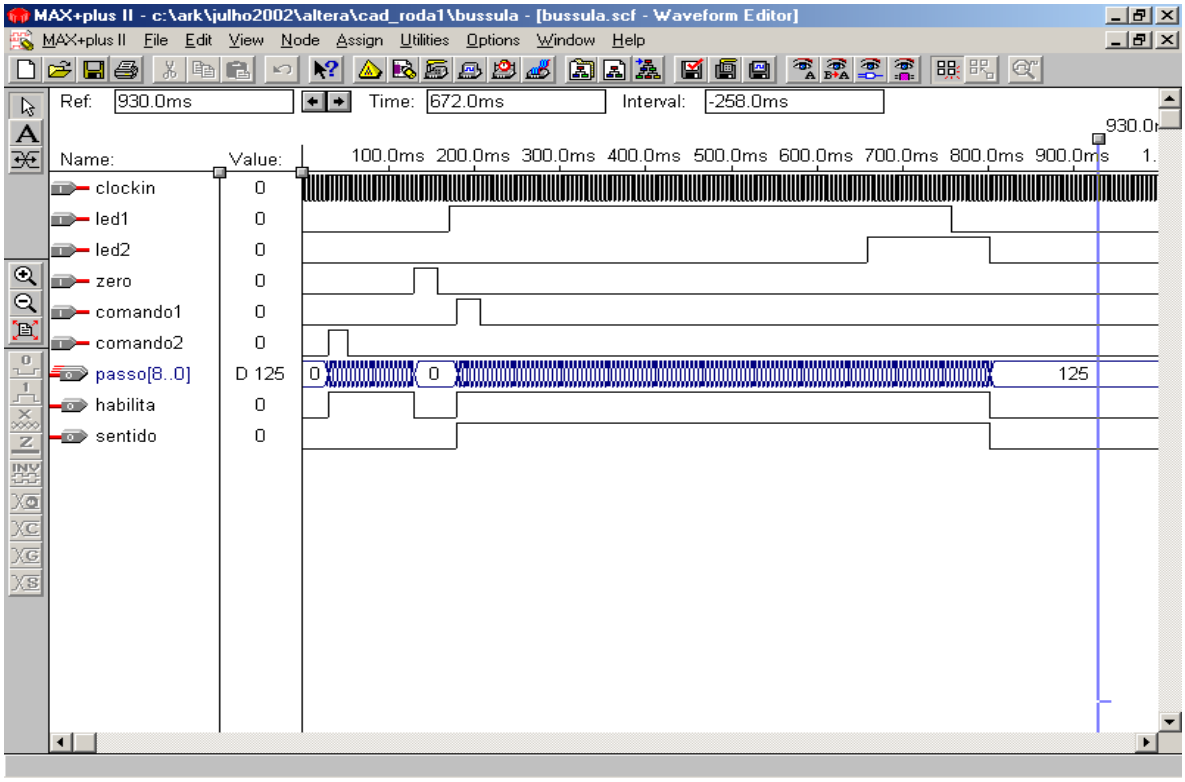


Figura 63 –Simulação da execução do comando de busca de referência no bloco de comando da bússola.

5.1.3.8 Módulo de Interface do Encoder

O módulo de interface do encoder (ou módulo encoder) é responsável por traduzir em uma palavra binária de 16 bits a freqüência de um sinal de entrada fornecido por um trem de pulsos digital oriundo de um encoder incremental acoplado ao eixo de um motor do robô móvel experimental. Através do módulo encoder é possível a realimentação de velocidade do motore, bem como o armazenamento de informações de odometria de uma das rodas.

A Figura 64 apresenta a implementação do módulo encoder em linguagem gráfica. O bloco "end16" conta o número de pulsos do encoder (sinal ´encoder´) durante um período de tempo dado pela saída do bloco "jan_enc".

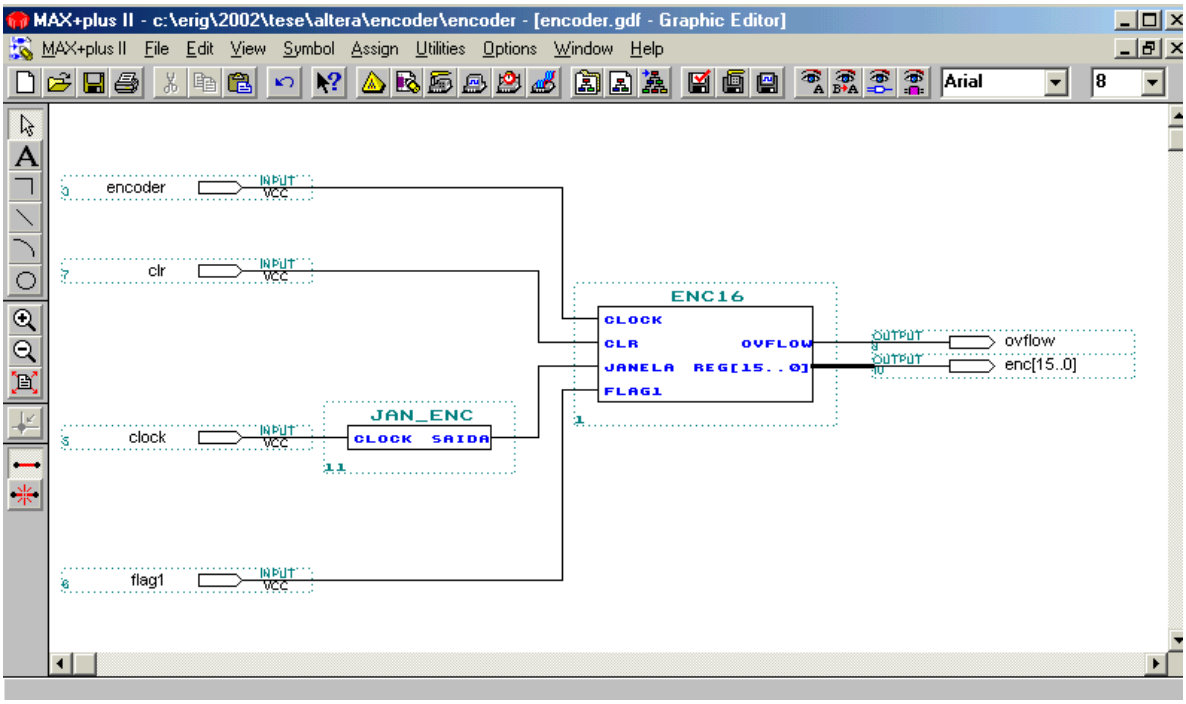


Figura 64 - Implementação em linguagem gráfica do módulo encoder.

A duração desta janela de contagem pode ser alterada pelo sinal ´clock´. Isto permite que janelas maiores sejam usadas quando os pulsos oriundos do encoder apresentarem períodos maiores (motor mais lento). Na Figura 65 é apresentada uma simulação do módulo encoder. Para

um sinal ´encoder´ constante (motor em velocidade constante) e para uma janela de tempo de contagem de aproximadamente de 10,24 ms a palavra de saída é constante e igual a 2560.

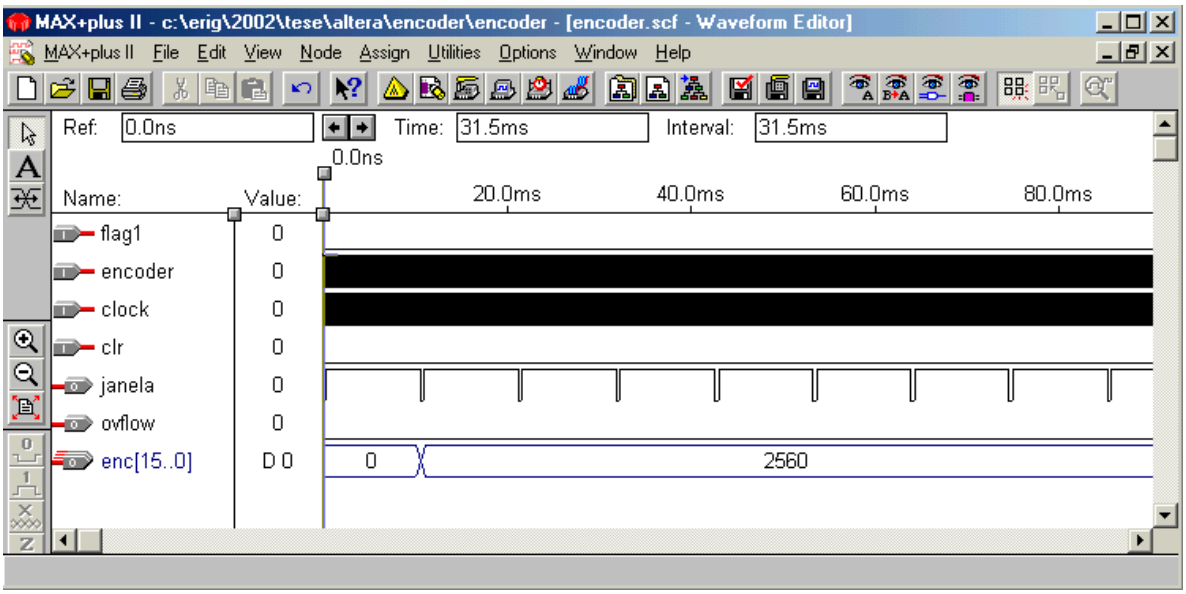


Figura 65 - Simulação do módulo encoder. O sinal ´janela´ foi adicionado, representando a janela de tempo de contagem.

Os demais sinais mostrados na Figura 64 são:

- ´clr´ – Sinal usado para zerar o valor de saída ´enc[15..0]´. Ativo em nível alto.
- ´flag1´ – Sinal usado para manter o último estado de ´enc[15..0]´. Ativo em nível alto.
- ´ovflow´ – Indica que o número de pulsos durante uma janela de contagem excedeu o limite superior de contagem (65535 pulsos).

5.2 Robô de Competição

5.2.1 Introdução

O robô de competição é um projeto desenvolvido pela equipe Marthe, formada por alunos do curso de Engenharia Mecatrônica da Unicamp. Este projeto participou de eventos de competição entre robôs que ocorreram entre diversas instituições de ensino de engenharia no ano de 2001 e 2002. Este eventos apresentam como motivação os seguintes pontos:

- Permitir a competição entre as principais escolas de Engenharia do Brasil, fomentando a transferência de conhecimentos.
- Ser uma atividade extracurricular dos alunos de Engenharia Mecatrônica da Unicamp.
- Estimular trabalho em equipe com alunos de outros cursos da Unicamp.
- Propiciar o contato direto dos alunos com atividades práticas.
- Estimular a utilização de trabalhos de pós-graduação como parte da solução de projeto, permitindo o contato dos alunos dos cursos de graduação com novas tecnologias.

Estes eventos são competições de sobrevivência, onde os participantes buscavam a vitória pela eliminação dos concorrentes segundo regras bem estabelecidas. Em função destas regras é proposto um projeto com as seguintes especificações:

- Peso máximo do robô: 50 kg.
- Capacidade de sofrer impactos.
- Utilização de motores CC alimentados por baterias de 12V.
- Controle remoto independente para cada um dos motores utilizados.

- Controle de velocidade dos motores utilizados.

A Figura 66 apresenta duas versões de implementação do robô de competição. Na primeira versão é utilizado um conjunto de motor, redutor e esteiras para propiciar a tração do robô. Na segunda versão, um outro motor, um outro redutor e rodas são usados para propiciar a tração necessária. Estas versões refletem a tentativa de obter uma melhor relação entre a potência mecânica, o tempo de operação das baterias, o custo da eletrônica de potência utilizada e a disponibilidade dos diversos componentes no mercado. Ambas as versões apresentam os seguintes elementos comuns de projeto:

- Utilização de dois motores CC, com capacidade de rotação nos sentidos horário e anti-horário.

- Controle remoto implementado com rádio-controle comercial utilizado em aeromodelismo.

- Sensores de corrente para proteção da eletrônica de potência.

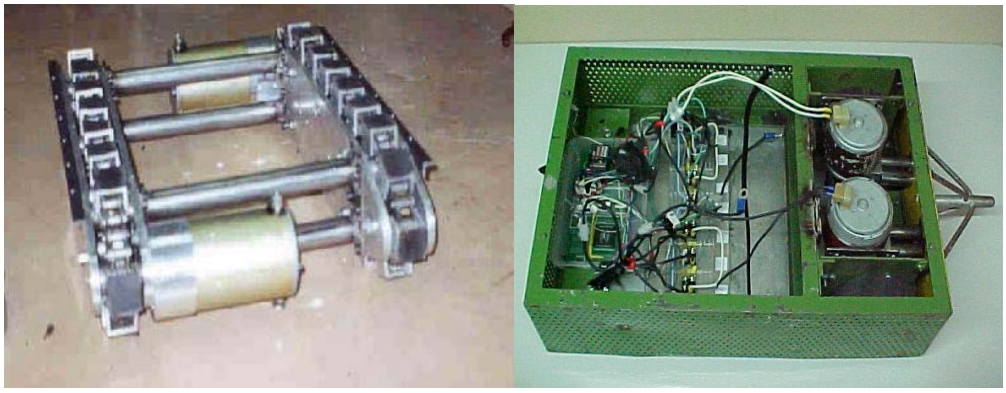


Figura 66 - Duas versões de implementação do robô de competição. A primeira versão utiliza tração por esteiras. A segunda versão utiliza rodas.

5.2.2 Aplicação do Ambiente Proposto ao Robô de Competição

Como os requisitos de projeto podem ser atendidos pelo ambiente proposto, é apresentada

uma solução descrita pelo diagrama de blocos da Figura 67.

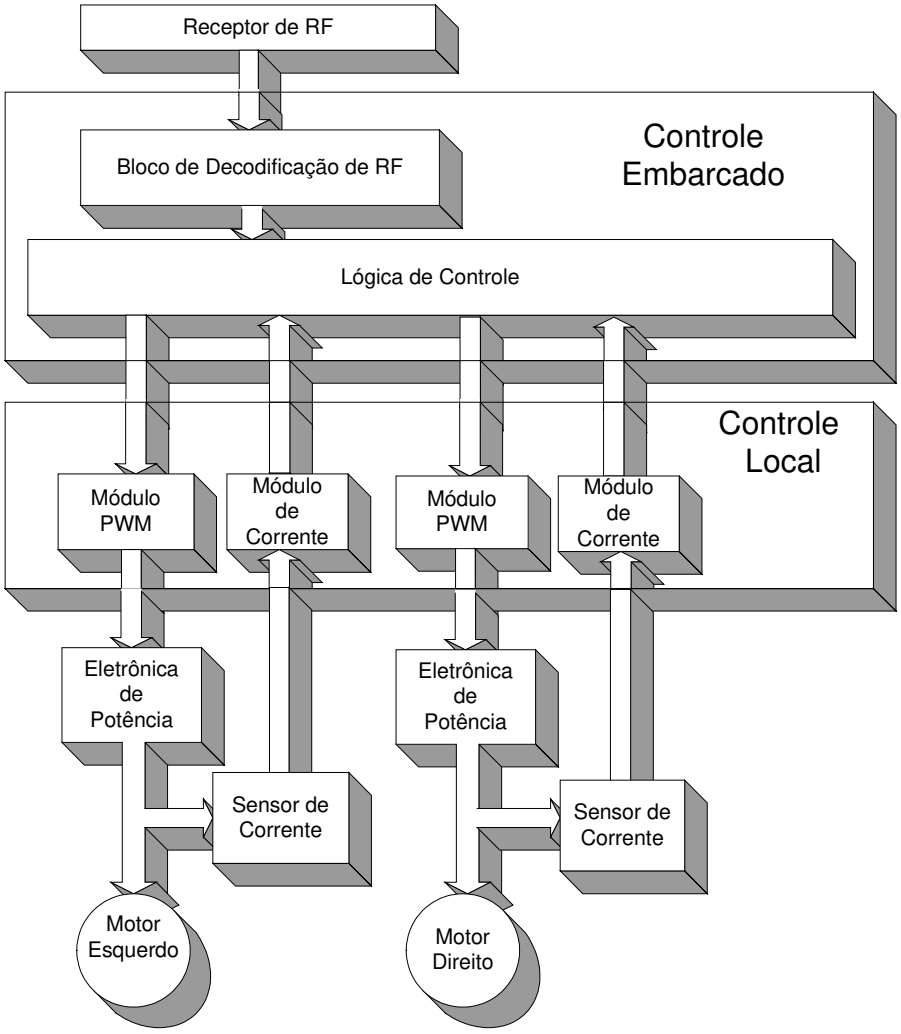


Figura 67 - Diagrama de blocos do ambiente proposto aplicado ao robô de competição.

Na Figura 67 são enfatizados os dois níveis de controle utilizados: o nível de controle embarcado e o nível de controle local. No nível de controle embarcado são decodificados os sinais de RF e gerados comandos individuais para os blocos que controlam os motores no nível de controle local. Estes comandos são influenciados pelas informações das intensidades das correntes nos enrolamentos de ambos os motores. No nível de controle local são gerados os pulsos PWM que comandam a eletrônica de potência de cada motor. Além disto é tratada a informação dos sensores de corrente acoplados aos enrolamentos dos motores.

É importante ressaltar que neste projeto não é implementado o nível de controle supervisor. Esta opção está de acordo com a característica de adaptação à tarefa desejada neste ambiente. Apenas os recursos necessários para atender aos requisitos de projeto são efetivamente utilizados.

5.2.3 Nível de Controle Embarcado

O controle de velocidade em cada motor é feito de maneira individual, permitindo rotação nos sentidos horário e anti-horário, com diferentes velocidades. A Figura 68 apresenta a parte da implementação gráfica relativa no nível de controle embarcado, onde os sinais de saída perfil, sentido e inibe_esq são usados pelos blocos do nível de controle local.

O transmissor escolhido para este projeto foi o transmissor Futaba que opera com quatro canais de FM. Este transmissor é unidirecional, ou seja, apenas comandos são enviados. Apenas dois canais são utilizados, um canal para cada motor. Na Figura 68 só um dos canais (sinal de entrada canal1) é apresentado.

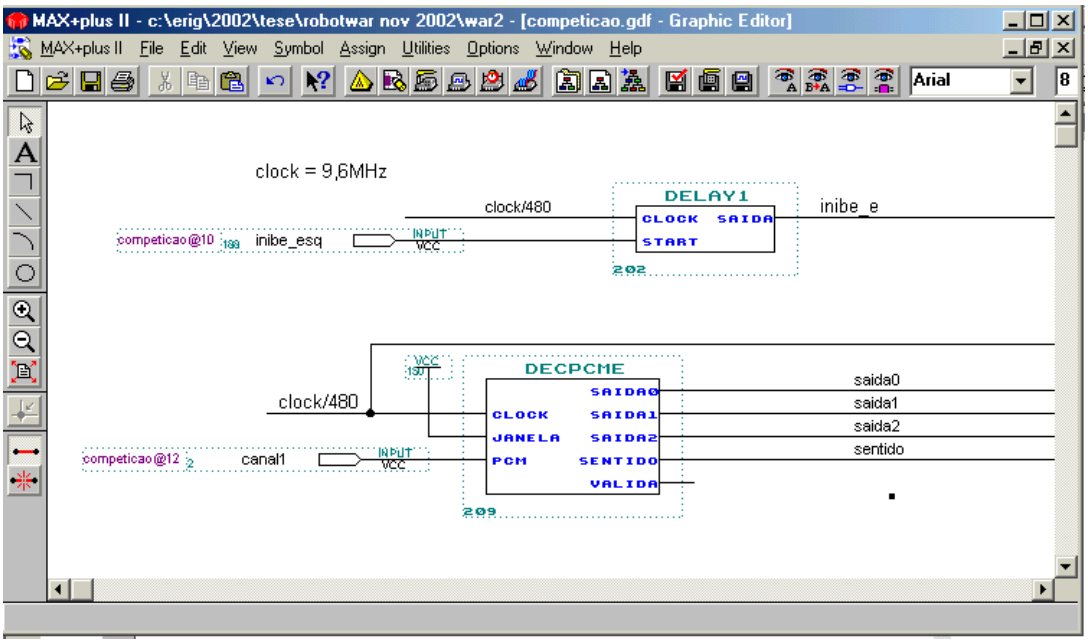


Figura 68 - Parte da implementação em linguagem gráfica representando o nível de controle embarcado do robô de competição.

5.2.3.1 Decodificação dos sinais de RF

Cada canal é decodificado no receptor como um sinal digital com as seguintes características:

- Período constante de 19,2 ms.

- Ciclo de trabalho positivo variando de 1,1 ms até 2 ms.

Em cada canal de RF, o sinal digital recebido apresenta uma relação linear com o deslocamento angular de um potenciômetro contido no transmissor. O objetivo é associar a velocidade e sentido de cada motor ao deslocamento deste potenciômetro, permitindo um controle adequado da trajetória e da velocidade do robô de competição. A Figura 69 representa a conversão do valor do ciclo positivo de um canal de RF

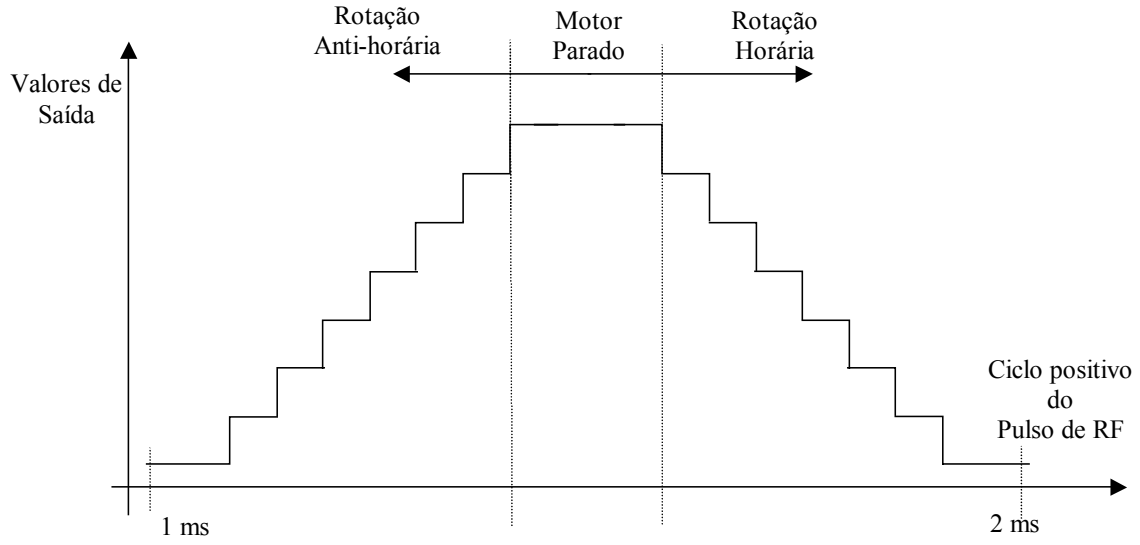


Figura 69 - Representação da operação de conversão do ciclo positivo do sinal de RF em valores de saída que representam a rotação horário e anti-horário do motor em diferentes velocidades.

O bloco “decpcme”, representado na Figura 68 faz exatamente a conversão proposta na

Figura 69. A implementação é feita por uma máquina de estados implementada em VHDL e descrita no Apêndice D.

5.2.3.2 Lógica de Controle

A lógica de controle neste caso é bastante simples. Apenas um laço de corrente é utilizado para manter a corrente nos enrolamentos dos motores dentro de níveis seguros. No caso do primeiro protótipo desenvolvido os motores utilizados chegavam a ter correntes de partida acima de 150 A, implicando em uma eletrônica de potência super dimensionada e em vida útil das baterias reduzida. O bloco “delay1” , na Figura 68, inibe o fornecimento de energia para o motor toda vez que a corrente limite é atingida. No protótipo final, esta corrente máxima é de 50 A.

5.2.4 Nível de Controle Local

Neste projeto o nível de controle local pode ser visto na implementação gráfica parcial do nível de controle local do robô de competição. O bloco ”duty” é semelhante ao módulo PWM tratado no item 5.1.3.1. O bloco “duty” também gera um perfil PWM, mas com um número menor de possíveis perfis. Assim, apenas oito velocidades diferentes podem ser usadas.

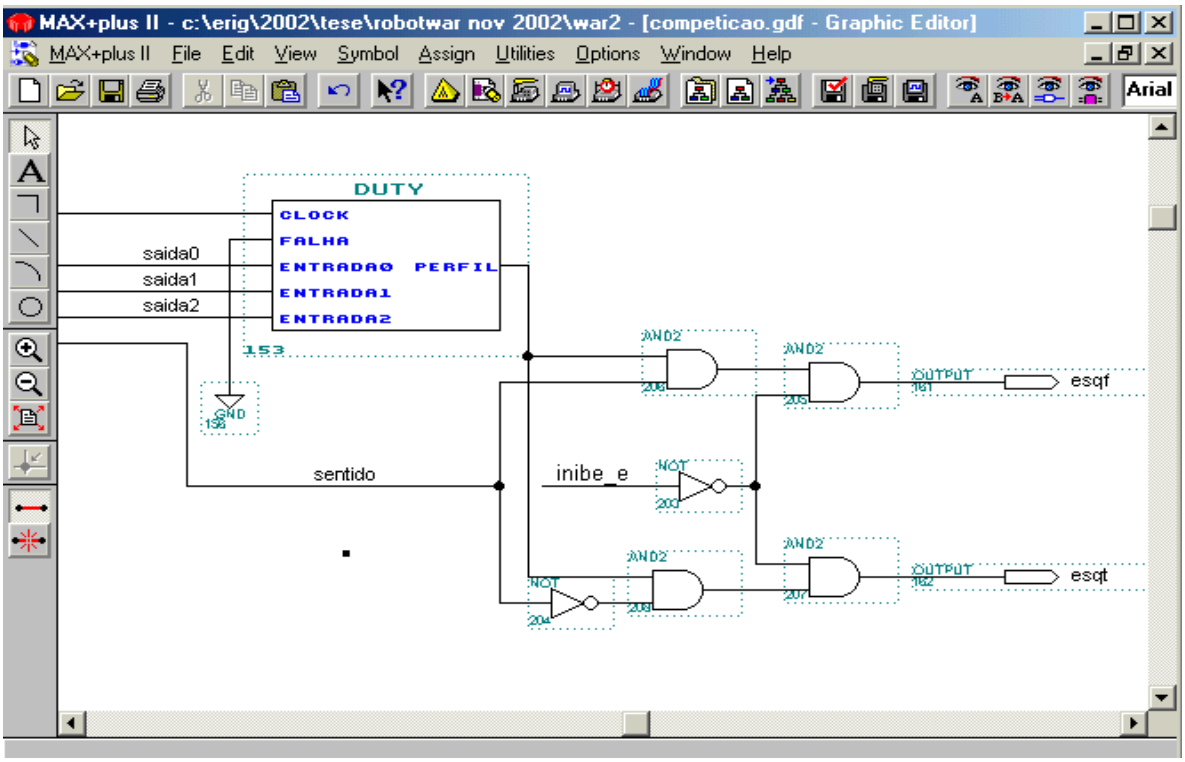


Figura 70 - Implementação gráfica do nível de controle local do robô de competição.

Os sinais esqf e esqt controlam uma ponte H, permitindo o controle de velocidade do motor nos dois sentidos de rotação, usando apenas uma fonte de alimentação (bateria).

5.3 Cadeira de Rodas Automatizada

5.3.1 Introdução

A cadeira de rodas automatizada está inserida dentro de um projeto de acessibilidade às instalações da Faculdade de Engenharia Mecânica da UNICAMP, o qual visa propiciar melhores meios de acesso a deficientes físicos dentro das dependências da Faculdade. Paralelamente existe a proposta de realizar testes com um protótipo desta cadeira dentro do Hospital Universitário da Unicamp, agregando gradativamente funcionalidades que venham a se mostrar necessárias. A idéia central é desenvolver um módulo de automação de cadeiras de rodas. Este módulo permite a

conversão de uma cadeira de rodas convencional (não motorizada) em uma cadeira de rodas movida por motores elétricos e com capacidade de ser controlada pelo paciente ou por um assistente remoto. A Cadeira de Rodas Automatizada participou da 1ª Olimpíada Universitária Altera, em 2002, classificando-se em 3º lugar. São enfatizados os seguintes aspectos no projeto:

- Baixo custo final do protótipo.
- Facilidade de adaptação a diferentes tipos de cadeiras de rodas convencionais.
- Facilidade na adição de novas funcionalidades.
- Capacidade de monitoração remota.

Levando em conta os aspectos de baixo custo e flexibilidade, o projeto é centrado na utilização do ambiente proposto.

O primeiro protótipo funcional é apresentado na Figura 71. Este protótipo participou da 1ª Olimpíada Universitária Altera.

Figura 71 - Primeiro protótipo funcional da cadeira de rodas automatizada.

O segundo protótipo funcional é apresentado na Figura 72.

Figura 72 – Segundo protótipo funcional da cadeira de rodas automatizada. É mostrada a cadeira sem automatização (esquerda) e com automatização (direita).

O segundo protótipo foi desenvolvido visando a padronização nos motores, a padronização dos redutores, a padronização das baterias e a diminuição do peso final.

Os protótipos apresentados na Figura 71 e na Figura 72 utilizam um modelo padrão de cadeira de rodas, de baixo custo e com capacidade de carga de 150 kg. Ambos os protótipos utilizam a mesma eletrônica de potência e a mesma eletrônica de controle, baseada em lógica reconfigurável, segundo o ambiente proposto.

5.3.2 Aplicação do Ambiente Proposto à Cadeira de Rodas Automatizada

As funcionalidades inicialmente propostas no módulo de automatização da cadeira de rodas são:

- Operação remota.

- Interface com usuário através de joystick.

- Interface com sensores para operação.

- Controle de velocidade dos motores.

Até a conclusão deste trabalho nem todas as funcionalidades foram implementadas. A operação remota e a interface com sensores são funcionalidades propostas como continuação futura deste trabalho, já considerando as necessidades de operação em ambiente hospitalar. A Figura 73 apresenta o diagrama de blocos apresentando as funcionalidades efetivamente implementadas.

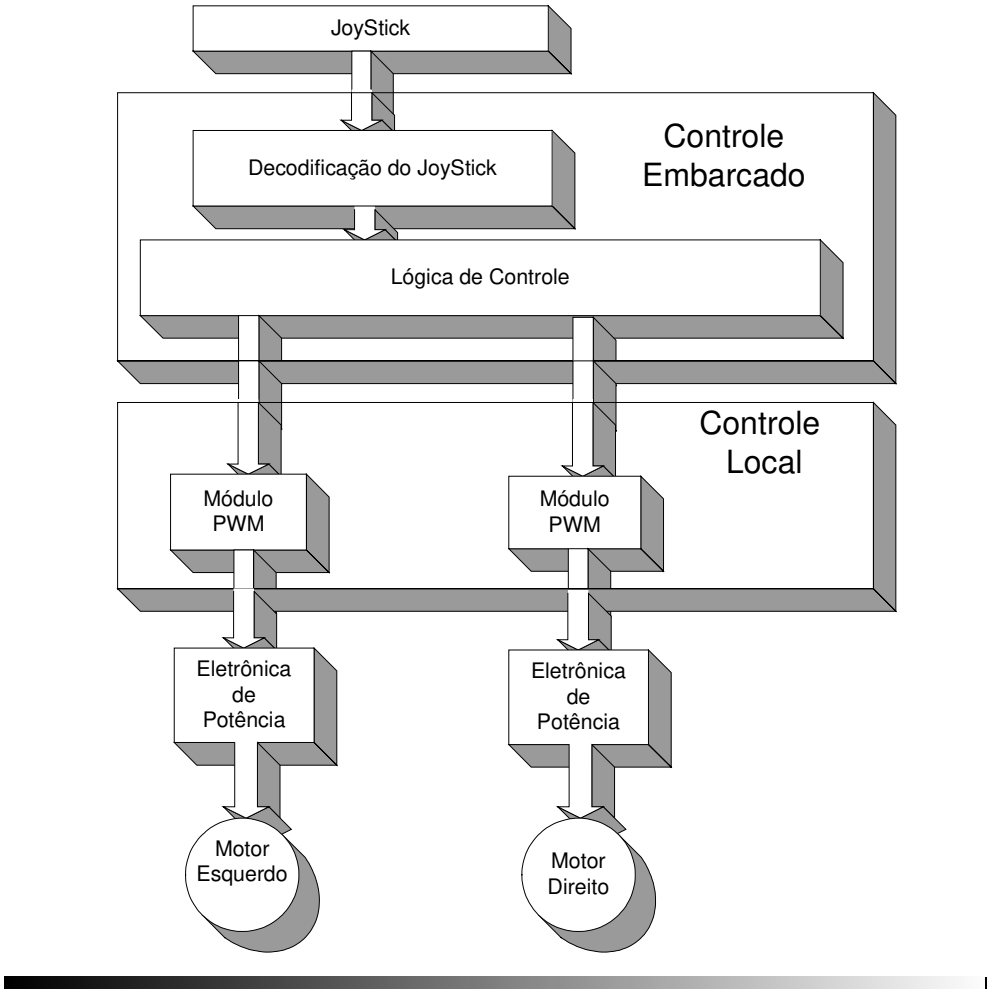


Figura 73 - Diagrama de blocos do ambiente proposto aplicado à cadeira de rodas automatizada.

5.3.3 Nível de Controle Embarcado

O projeto em lógica reconfigurável é visualizado no diagrama de blocos da Figura 74, onde são apresentados todos os blocos relacionados com sensores e com a operação remota. Apenas a interface com o usuário via joystick e o módulo de comunicação remota são efetivamente implementados, sendo que os blocos restantes dependem de implementações adicionais de *hardware*.

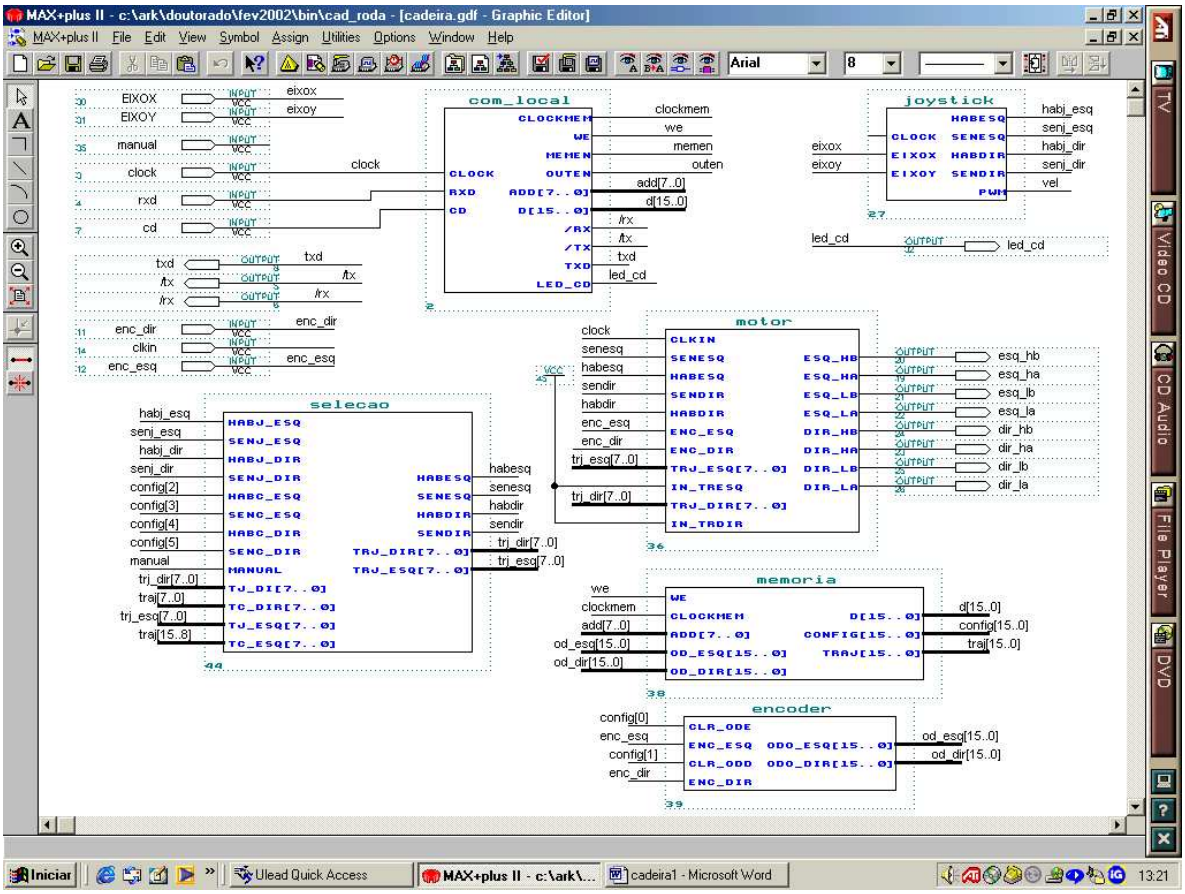


Figura 74– Implementação em linguagem gráfica dos diversos módulos relacionados com o nível de controle embarcado e o nível de controle local aplicados a cadeira de rodas automatizada.

Os blocos não implementados (mas previstos como parte do projeto final) são:

- Operação remota – Através dos blocos "com_local" e "memoria" é implementada a interface de comunicação bidirecional entre o nível de controle embarcado e o nível de controle supervisor, permitindo a monitoração de parâmetros da cadeira, envio de alarmes ou ainda a geração remota de trajetórias.
- Tratamento dos Encoders – Através do bloco "encoder" é realizado o elo de realimentação de velocidade dos motores, bem como o armazenamento de informações de odometria de cada roda da cadeira.

5.3.3.1 Interface com o Usuário – JoyStick

O joystick é a interface principal com o usuário responsável pelo direcionamento da cadeira. Optou-se por um joystick analógico do tipo usado em sistemas PC, para o qual é proposto o mapa de operação da Figura 75. Outras opções de mapas podem ser configuradas facilmente em função de necessidades específicas de cada usuário. Através do bloco "joystick" são decodificas as informações oriundas dos eixos frente – ré e esquerda – direita do joystick.

Figura 75 – Mapa de operação do JoyStick.

5.3.4 Nível de Controle Local

Na Figura 74 um único bloco é associoado ao nível de controle local: o módulo "motor".

5.3.4.1 Módulo de Controle dos Motores

O módulo de controle dos motores, implementado no bloco “motor” é detalhado na Figura 76, onde os blocos “gerador” são responsáveis pela geração dos perfis PWM para os motores da cadeira de rodas automatizada. Estes blocos são semelhantes ao módulo PWM tratado no item 5.1.3.1.

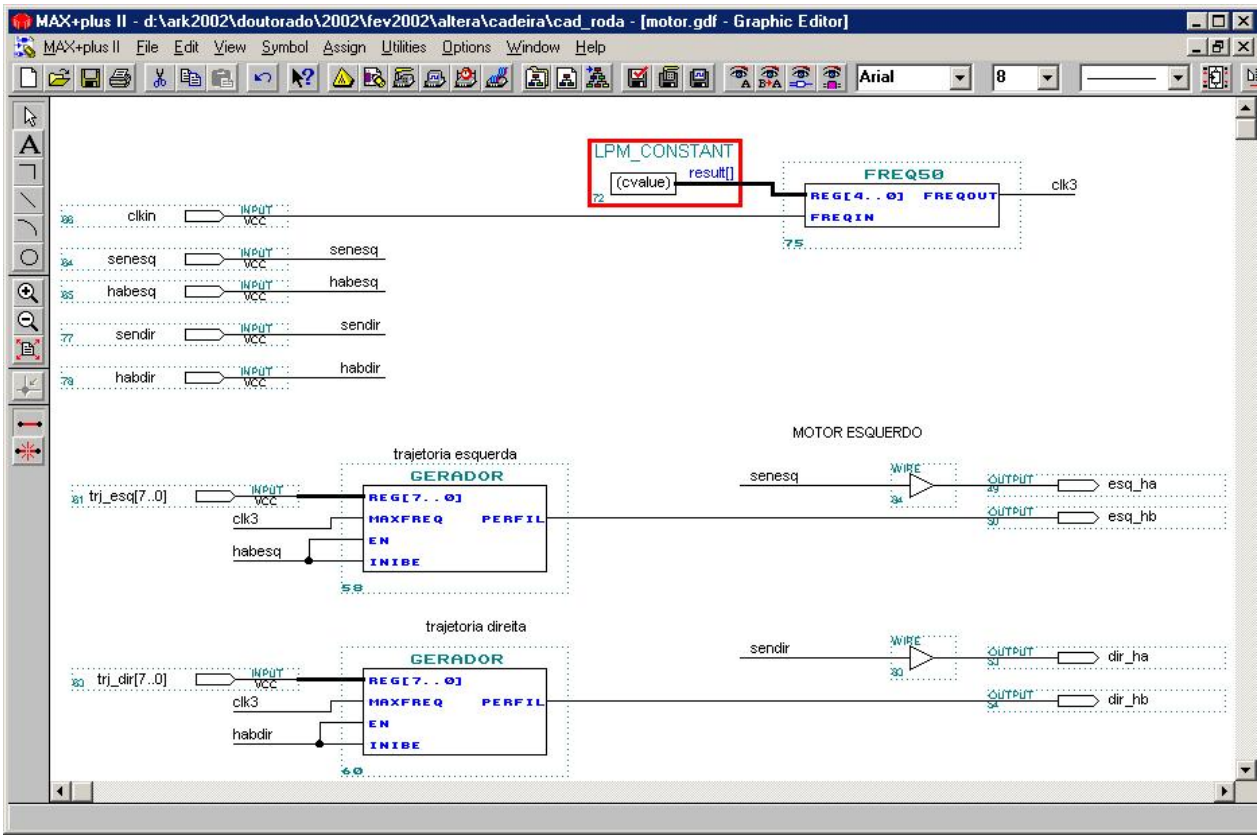


Figura 76- Detalhamento do bloco “motor”, onde são observados os blocos “gerador”, que geram os perfis PWM para os motores da cadeira de rodas automatizada.

5.3.5 Módulos de Potência

Os módulos de potência foram implementados com chaves MOSFET. Optou-se por uma configuração em ponte H, que permite, com sistemas de alimentação por baterias unipolares (no caso 14 V corrente contínua) a operação dos motores CC em sentido horário e anti-horário. Além disto, a configuração em ponte H permite a operação no modo PWM. Isto caracteriza o módulo

de potência como um circuito de alta eficiência. Este fator é importante em sistemas embarcados, onde a energia disponibilizada pelas baterias é limitada. A Figura 77 apresenta o módulo de MOSFET montado sobre uma placa dissipadora. Este módulo permite operações nominais de até 30 A. Na figura é apresentado um módulo sem a proteção de resina. Os módulos são cobertos com resina para prover a resistência mecânica necessária em um sistema sujeito a vibrações, como é o caso da cadeira de rodas.

Para o controle imediato de cada chave da ponte H optou-se por controladores PVI1050N da International Rectifier. Estes circuitos integrados permitem a tradução direta de pulsos de comando digitais em sinais adequados para a comutação das chaves. Além disto, estes componentes implementam uma isolação galvânica entre os módulos de potência e os módulos de controle, condição fundamental para evitar ruídos e sobre-cargas nos circuitos de controle digital.

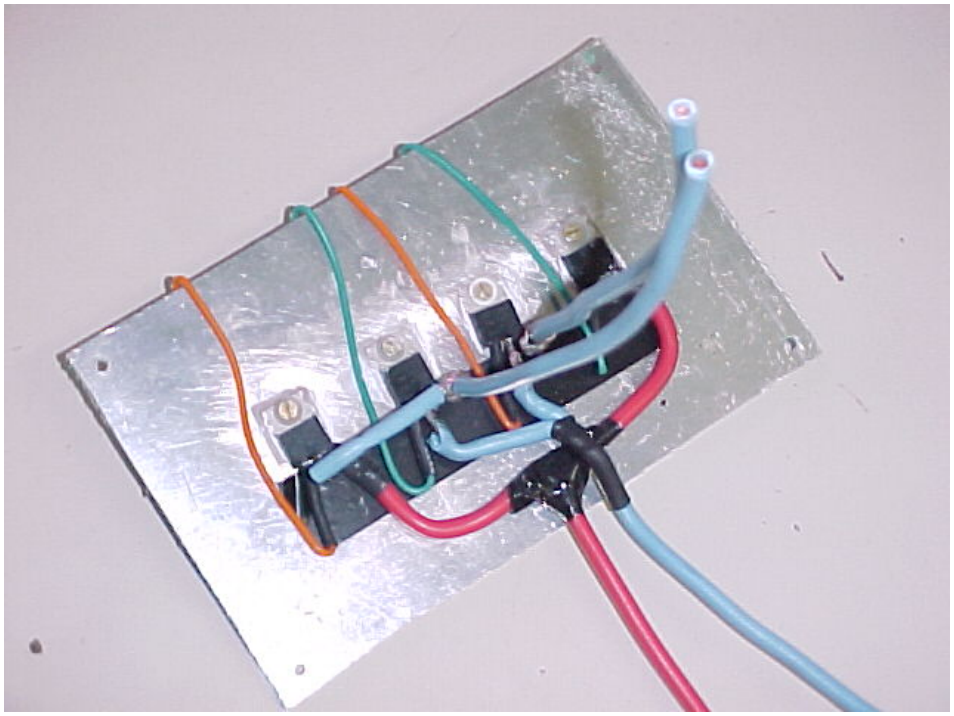


Figura 77 – Módulo de potência implementado com chaves MOSFET.

5.4 Conclusão

Este capítulo descreveu a validação experimental do ambiente proposto aplicado a três sistemas embarcados móveis. Os sistema apresentados têm semelhanças quanto aos atuadores e sensores utilizados. Contudo, cada sistema têm suas particularidades no que se refere ao local de operação, a eletrônica de potência necessária para operá-los, o tipo de interface com o usuário. Estas particularidades inerentes a cada sistema são uma constante no desenvolvimento de protótipos, sejam móveis ou não. A elevada adaptabilidade do sistema proposto aos requisitos de cada projeto é evidente.

O nível de controle supervisor pode ser perfeitamente descartado dependendo da tarefa requerida pelo sistema embarcado. Ao nível do controle embarcado não é necessariamente imprescindível a utilização de microprocessadores. Nos exemplos apresentados não foram utilizados. Evidentemente, em aplicações mais complexas, com tarefas mais complexas, a utilização de microprocessadores associados com lógica reconfigurável ou mesmo microprocessadores reconfiguráveis podem e devem ser utilizados.

Outro aspecto que deve ser observado é o fácil reaproveitamento de blocos de um projeto em outro projeto. Um exemplo pode ser visto nos blocos usados para gerar os perfis PWM de controle dos motores CC. Nos três projetos há o aproveitamento de uma mesma estrutura.

Capítulo 6

Conclusões e Sugestões para Futuros Trabalhos

A eletrônica embarcada apresenta restrições quanto ao consumo de energia e quanto ao volume físico ocupado. Considerando ainda o grande número de tarefas e operações que dispositivos embarcados móveis podem realizar, bem como a variedade de combinações de sensores e atuadores usados na realização destas tarefas, é necessário prever em sua eletrônica, aspectos de flexibilidade e facilidade de alterações no projeto.

Sistemas de desenvolvimento baseados em computação reconfigurável apresentam características adequadas para execução desta classe de projeto. Eles permitem, por exemplo, que o controle dos atuadores seja feito de modo a aproveitar as características de baixo consumo, alta velocidade de operação, capacidade de integração, flexibilidade e facilidade de programação dos dispositivos lógicos reprogramáveis..

Ferramentas que venham a permitir a rápida implementação prática de um projeto de *hardware* não são apenas uma comodidade técnica, mas são também essenciais para a viabilidade econômica deste projeto. Isto é válido para a empresa e para a universidade e centros de pesquisa, cada vez mais sujeitos às competições do mercado. Espera-se com este trabalho contribuir para o esclarecimento dos potenciais destes sistemas, bem como, através de um exemplo prático, demonstrar sua aplicabilidade na área de controle de sistemas embarcados. A utilização de interfaces gráficas e VHDL no projeto do controlador permite descrever um projeto típico em lógica reconfigurável, onde diferentes blocos são projetados separadamente para posterior

integração. O ambiente de simulação permite a visualização de possíveis erros de projeto, antes mesmo da execução física do mesmo. O sistema de desenvolvimento contribuiu ainda para o baixo custo final do produto e para sua facilidade de implementação e de correção de erros. Outra vantagem do sistema de desenvolvimento é a característica de hierarquização, que facilita o trabalho em equipe.

Este trabalho apresenta como contribuição uma alternativa de ambiente para auxílio no desenvolvimento de protótipos de sistemas embarcados. A utilização de lógica reconfigurável contribui para verificar os limites da aplicação desta alternativa de implementação de algoritmos em problemas práticos. Outras contribuições são as interfaces desenvolvidas em *hardware*, disponíveis para reprodução pela comunidade acadêmica. O ambiente proposto é ainda uma ferramenta de apoio didático em laboratórios de graduação de pós-graduação de engenharia, já sendo utilizado com sucesso no Laboratório de Mecatrônica do Curso de Engenharia Mecatrônica da Unicamp. O protótipo de cadeira de rodas automatizada também é uma modesta contribuição para o meio médico, principalmente considerando-se o baixo custo final em relação aos modelos comerciais disponíveis. Finalmente, pode-se citar como contribuição ao meio científico, a geração de artigos publicados em revistas e eventos de circulação e internacional.

Aspectos promissores do ambiente proposto:

- Flexibilidade – Há uma grande variedade de configurações possíveis na implementação de soluções para diversos problemas associados com desenvolvimento de protótipos de sistemas embarcados.

- Adaptação à tarefa – Apenas os recursos necessários à execução de uma tarefa são utilizados. Sistemas mais simples são tratados com versões mais simples do ambiente proposto, evitando-se desperdício de recursos.

- Ambiente aberto – Permite que seus blocos componentes sejam totalmente acessados e eventualmente modificados para agregar novas soluções.

- Facilidade de expansão – Novos recursos podem ser facilmente adicionados a um projeto, permitindo a adição de novas funcionalidades.

- Possibilidade de controle supervisório de vários protótipos simultaneamente.

- Custo relativamente baixo.

O ambiente proposto não está completo e é provável que nunca venha a estar. O objetivo é que este ambiente evolua, seus componentes sendo aperfeiçoados com o tempo, permitindo que soluções progressivamente mais simples, rápidas e baratas sejam usadas no desenvolvimento de protótipos de sistemas embarcados.

Este trabalho abre perspectivas para a realização dos seguintes trabalhos futuros, entre outros:

- Estudo e implementação de códigos corretores de erro nos módulos de comunicação embarcado e remoto.

- Implementação das interfaces de comunicação entre o nível de controle supervisor e o nível de controle embarcado usando fibras óticas e infravermelho.

- Aprofundamento do estudo da implementação de estratégias de controle em lógica reconfigurável, em especial controladores preditivos.

- Estudo e implementação de técnicas de fusão de sensores em lógica reconfigurável.

- Adição de microprocessadores ou de microprocessadores reconfiguráveis ao nível de controle embarcado.

- Completar e expandir o projeto da cadeira de rodas automatizada, permitindo sua implantação em ambiente hospitalar e acrescendo funcionalidades para atender novos requisitos

operacionais.

- Desenvolver interfaces para operação de sistemas embarcados cooperativos, como o caso de vários robôs móveis operando em conjunto para um determinado objetivo em um espaço de tarefa comum.

Referências Bibliográficas

Accolade Design Automation, Inc. Disponível na internet via www, URL: http://www.acc-eda.com, 2002.

Acosta, G.; Tosini, M. "A firmware digital neural network for climate prediction applications". Proceedings of the 2001 IEEE International Symposium on Intelligent Control, ISIC '01, pp. 127 – 131, 2001.

Actel Corporation. Disponível na internet via www, URL: http://www.actel.com, 2002.

Alford, G. D. "A low cost CAD/CAM system with simulation". In: Schrefler, B. A., Lewis, R. W. (Eds.). Engineering software for microcomputers. Swansea: Pineridge, 1984, pp.201-214.

Altera Corporation, "In-System Programmability" in MAX Devices Application Note, 1998.

Altera Corporation, MAX 7000, "Programmable Logic Family Data Sheet", 1998.

Altera Corporation.Disponível na internet via www, URL: http://www.altera.com, 2002.

Aporntewan, C.; Chongstitvatana, P. "A Hardware Implementation of the Compact Genetic Algorithm". Proceedings of the IEEE Congress on Evolutionary Computation, pp. 624 – 629, 2001.

Associação Brasileira de Normas Técnicas, Rio de Janeiro. NBR-6023; referências bibliográficas. Rio de Janeiro, 1989, 19p.

Aström, K.J. and B. Wittenmark. "Computer Controlled Systems". Third Edition, Prentice Hall, New Jersey, 1997.

Atmel Corporation. Disponível na internet via www, URL: http://www .atmel.com, 2002.

Baraza, J. C.; Gracia, j.; Gil, D.; Gil, P. J. “A Prototipe of a VHDL-Based Fault Injection Tool: Description and Application”. Journal of System Archicteture 47, pp. 847-867, published by Elselvier Science Ltd., 2002.

Borenstein, J., 1995, "Control and Kinematic Design of Multi-Degree-of-Freedom Mobile Robots with Compliant Linkage”. IEEE Transactions on Robotics and Automation, Vol. 11, No 1, pp 21-35, February 1995.

Borenstein, J., Everett H. R. and Feng L. “Sensors and Methods for Mobile Robot Positioning”. University of Michigan, 1996.

Boucher P.; Dumur, D. “Survey on Predictive Control”. Orsay: École Supérieure D'Électricité, 1997. 37 p.

Boucher, P; Codron, P. “Multivariable generalized preditícve control with multiple reference model: a flexible arm application”. Sidney: 12th IFAC World Congress, 1993. 4 p.

Cassemiro, Edna Rodrigues. “Metodologia para Desenvolvimento de Dispositivos Biomecânicos para Aplicação em Próteses Antropomórficas”. Tese de Mestrado. Faculdade de Engenharia Mecânica, Unicamp, 103 pp., fevereiro de 2002.

Changuel, A.; Rolland, R.; Jerraya, A.A. “Design of an Adaptive Motors Controller Based on Fuzzy Logic Using Behavioural Synthesis”. IEEE Design Automation Conference, with EURO-VHDL '96 and European Exhibition Proceedings, EURO-DAC '96, pp. 48 – 52, 1996.

Chen, Ruei-Xi; Chen, Liang-Gee; Chen, Lilin. “System Design Consideration for Digital Wheelchair Controller”. IEEE Transactions on Industrial Electronics, pp. 898 – 907, Volume 47, Issue 4, 2000.

Cirstea, M.; Aounis, A.; McCormick, M.; Urwin, P.; Haydock, L. “Induction Motor Drive System Modelled in VHDL”. VHDL International Users Forum Fall Workshop Proceedings, pp. 113 – 117, 2000.

Cirstea, M.; Dinu, A. “A New Neural Networks Approach To Induction Motor Speed Control”. IEEE 32nd Annual Power Electronics Specialists Conference, PESC. 2001 IEEE, pp. 784 – 787, vol.2, 2001.

Clarke, D. W.; Mohtadi, C. “Properties of Generalized Predictive Control”. Automatica, Vol. 25, No. 6, p. 859-875, 1989.

Clarke, D. W.; Mohtadi, C. “Tuffs P. S. Generalized Predictive Control - Part I. The Basic Algoritm”. Automatica, Vol. 23, No. 2, p. 137-148, 1987.

Codron, P.; Boucher, P. “Multivariable generalized preditícve control with multiple reference model: a new choise for the reference model”. Groningen European Control Conference, 1993. 4p.

Codron, Pascal. “Commande Prédictive Multivariable”. Orsay: École Supérieure d'Électricité, 1993. 168p. Tese (Docteur en Science).

Compton, Katherine. “Reconfigurable Computing: A Survey of Systems and Software, Acm Computing Surveys”, pp.171-210, vol 34, n. 2, 2002.

Coric, S.; Leeser, M.; Miller, E. “Parallel-Bean Backprojection: An FPGA Implementation Optimized for Medical Imaging”. FPGA´02, pp 217-226, Monterey, California, USA, fevereiro 2002.

Daly, Alan and Marnane, William. “Efficient Architectures for Implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic”. FPGA´02, pp 40-49, Monterey, California, USA, fevereiro 2002.

De Pablo, S.; Dominguez, J.A.; Lorenzo, S. Vaquero, L.J. “A Flexible Inverter Controller

for Prototypes. Proceedings of the IEEE International Symposium on Industrial Electronics", ISIE '97, pp. 254 – 257, vol.2 , 1997.

Dechechi, Eduardo Cesar. "Controle avançado preditivo adaptativo "DMC" multivariável". Campinas: Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas , 1998. 168 p. Tese (Doutorado).

Dido, J.; Geraudie, N.; Loiseau, L.; Payeur, O.; Savaria, Y. "A Flexible Floating-Point Format for Optimizing Data-Paths and Operators in FPGA-Based DSPs". FPGA´02, pp 50-55, Monterey, California, USA, fevereiro 2002.

Diniz, Pedro C; Park, Joonseok. "Data Reorganization Engines for the Next Generation of System-On-a-Chip FPGAs". FPGA´s 2002 Proceedings, pp. 237-244, 2002.

Dinsmore Sensors. Disponível na Internet via www. URL: http://www.dinsmoresensors.com, 2002.

Dumur, D.; Daumüller, S.; Boucher, P. "Popcorn: CAD for Predictive Polynomial Control. Application to Motor Drives". The First IEEE Conference on Control Applications, 1992. 5p.

Dumur, Didier. "Commande Prédictive et Machine-Outil". Orsay: École Supérieure d'Électricité, 1993. 156p. Tese (Docteur en Science).

Empringham, L.; Wheeler, P.; Clare, J. "A Matrix Converter Induction Motor Drive Using Intelligent Gate Drive Level Current Commutation Techniques". Conference Record of the IEEE Industry Applications Conference, pp.1936 – 1941, vol.3, 2000.

Endemano, Aitor. "System Level Simulation of a Double Stator Wobble Electrostatic Motor. Sensors and Actuator A 99", pp. 312-320, published by Elselvier Science Ltd.,2002.

Fanucci, L. "FAST: FFT ASIC Automated Synthesis". Integration, the VLSI journal, pp. 1-

15, published by Elselvier Science Ltd., 2002.

Franco, Denis T.; Carro, Luigi. “A FPGA Version of a Non-Linear Adaptative Filter”. IEEE.

Franklin, G. F., Powel, J. D., Workman, M. L. “Digital control of dynamic systems”. 2.ed. Reading: Addison-Wesley, 1992, Cap. 11: “Nonlinear Control”, pp.516-640.

Garbergs, B.; Sohlberg, B. “Implementation of a State Space Controller in a FPGA”. 9th Mediterranean Electrotechnical Conference, MELECON 98., pp. 566 – 569, vol.1, 1998.

Garbergs, B.; Sohlberg, B. “Specialised Hardware for State Space Control of a Dynamic Process”. IEEE Digital Signal Processing Applications Proceedings, TENCON '96, pp. 895 – 899, vol.2, 1996.

Garcia, Carlos E.; Prett, David M.; Morari, Manfred. “Model Predictive Control: Theory and Practice - a Survey”. Automatica, Vol. 25, No. 3 , p. 335-348, 1989.

Harkin, J.; McGinnity, T. M.; Maguire, L. P. “Genetic Algorithm Driven Hardware-Software Partioning for Dynamically Reconfigurable Systems”. Microprocessors and Microsystems 25, pp. 263-274, published by Elselvier Science Ltd., 2001.

Hasuko, K.; Fukunaga, C.; Ichimiya, R.; Ikeno, M.; Ishida, Y.; Kano, H.; Kurashige, H.; Mizouchi, K.; Nakamura, Y.; Sakamoto, H.; Sasaki, O.; Tanaka, K.; “A Remote Control System for FPGA-Embedded Modules in Radiation Environments”. IEEE Transactions on Nuclear Science, pp. 501 – 506, 2001.

Hu, Bao-Sheng; Li, Jing. “The Fuzzy PID Gain Conditioner: Algorithm, Architecture and FPGA Implementation”. Proceedings of The IEEE International Conference on Industrial Technology, ICIT '96, pp. 621 – 624, 1996.

Hwang, Yin-Tsung; Han, Jih-Cheng. “A Novel FPGA Design of a High Throughput Rate

Adaptive Prediction Error Filter". The First IEEE Asia Pacific Conference on ASICs, AP-ASIC '99, pp. 202 – 205, 1999.

Ito, S.A.; Carro, L. "A comparison of microcontrollers targeted to FPGA-based embedded applications". IEEE 13th Symposium on Integrated Circuits and Systems Design Proceedings, pp. 397 – 402, 2000.

Jeon, Jae Wook; Kin, Yun-Ki. "FPGA Based Acceleration and Deceleration Circuit for Industrial Robots and CNC Machine Tools". Mechatronics 12, pp. 635-642, published by Elselvier Science Ltd., 2002.

Kalra, Punit. "Reconfigurable FPGAs Help Build Upgradable Systems". Embede Developers Journal, pp. 16-21, September 2001.

Kean, Tom. "Cryptographic Rights Management of FPGA Intellectual Property Cores". FPGA´s 2000 Proceedings, pp. 113-118, 2000.

Kharrat, M.W. Masmoudi; N. Ghariani, M. Kamoun, L. "A Digital Control System Based on Codesign Technology". Proceedings of the Tenth International Conference on Microelectronics, ICM '98, pp. 257 – 261, 1998.

Khriji, Lazhar; Benacchia, Giuseppe; Gabbouj, Moncef; Sicuranza, Giovanni. "A Dedicated Hardware System for a Class of NonLinear Order Statistics Rational Hybrid Filters with Applications to Image Processing". IEEE 1999.

Klotchkov, I. V.; Pedersen, S. "A Codesign Case Study: Implementing Arithmetic Functions in FPGA´s". IEEE 1996.

Kobayashi, F. Haratsu; M.. "A Digital Pll with Finite Impulse Responses". IEEE International Symposium on Circuits and Systems, ISCAS '95, pp. 191 – 194, vol. 1, 1995.

Kollig, P.; Al-Hashimi, B.; Abbott, K. M. "Design and Implementation of Digital Systems

for Automatic Control Based on Behavioural Descriptions". IEEE 1996.

Kollig, P.; Al-Hashimi, B.M.; Abbott, K.M. "Efficient Scheduling of Behavioural Descriptions in High-Level Synthesis". IEEE Proceedings- Computers and Digital Techniques, pp. 75 – 82, 1997.

Köster, Markus; Teich, Jürgen. "(Self-)Reconfigurable Finite State Machines: Theory and Implementation". Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE´2002). IEEE 2002.

Kozlowski, T.; Dagless, E.L.; Saul, J.M.; Adamski, M.; Szajna, J. "Parallel Controller Synthesis Using Petri Nets". IEEE Proceedings - Computers and Digital Techniques, IEE Proceedings, pp.263 – 271, 1995. Reconfigurable systems: a survey

Lima, Carlos R. E.; Rosário, João M., "Utilização de Circuitos Lógicos Programáveis para Controle de Atuadores em Robôs Móveis", Revista Robótica, Portugal, 2001.

Lima, R. C. E, Silva; N. C., Rosário, J. M., 2000, "A Proposal of Flexible Architecture for Mobile Robotics", In Mechatronics 2000 – The 7th Mechatronics Forum International Conferênce and Mechatronics Education Workshop, 6 a 8 de setembro de 2000. Atlanta, 6p.

Maia, Maria de Lourdes Oliveira. "Controle Preditivo de uma Coluna de Absorção". Campinas: Faculdade de Engenharia Química, Universidade Estadual de Campinas, 1994. 94 p. Tese (Mestrado).

Mentor Graphics Corporation. Disponível na internet via www, URL: http://www.mentor.com, 2002.

Miyazaki, T. "Reconfigurable Systems: a Survey". IEEE Proceedings of the Design Automation Conference 1998, ASP-DAC '98, pp. 447 - 452 , 1998.

Monteiro, Fabrice; Dandache, Abbas; M´Sir, Amine; Lepley, Bernard. “A Fast CRC Implementation on FPGA Using a Pipelined Architecture for the Polynomial Division”. IEEE 2001.

Navabi, Z.; Day, S. “Tutorial on Use of VHDL for Description of Digital Systems”. Fourth Annual IEEE International ASIC Conference and Exhibit Proceedings, pp. T12 - 1/1-8, 1991.

Nomadic Technologies, “XR4000 Mobile Robot”, Disponível na Internet via www, URL: http://www.robots.com/products.htm, 1999.

Oliveira, Gustavo Henrique da Costa. “Controle Preditivo para Processos com Incertezas Estruturadas Baseado em Séries de Funções Ortonormais”. Campinas: Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, 1997. 151 p. Tese (Doutorado).

Page, Ian. “Reconfigurable Processor Archictetures”. Microprocessors and Microsystems 20”, pp. 185-196, published by Elselvier Science Ltd., 1996.

Pape, David A. “A modal analysis approach to flaw detection in ceramic insulators. In: International Modal Analysis Conference”, 11, 1993, Kissimmee. Proceedings. Bethel: Society for Experimental Mechanics, 1993, v.1, pp.35-40.

Peiró, Marcos M.; Valls, Javier; Sansaloni, T.; Pascual, A. P.; Boemo E. I. “A Comparation Between Lattice, Cascade and Direct-Form FIR Filter Structures by Using a FPGA Bit-Serial Distributed Arithmetic Implementation”. IEEE 1999.

Pimentel, Júlio C. G.; Le-Huy, Hoang. “A VHDL-Basead Methodology to Develop High Performance Servo Drivers”. Industry Applications Conference. Conference Record of the 2000 IEEE, pp. 1505 – 1512, vol.3, 2000.

Radiometrix Ltd, “Low Power UHF Data Tranceiver Module”. Disponível na Internet via www, URL: http://www.radiometrix.co.uk, 2002.

Rathna, G. N.; Nandy, S. K.; Parthasarathy, K. “A Methodology for Architecture Synthesis of Cascaded IIR Filters on TLU FPGAs”. 7th International Conference on VLSI Design - IEEE, pp. 225-228, January, 1994.

Renner, F.M.; Hoffmann, K. J.; Markert, R.; Glesner, M. “Desing Methodology of Application Specific Integrated Circuit for Mechatronic Systems”. Microprocessors and Microsystems, pp. 95-102, published by Elselvier Science Ltd., 2002.

Richalet, Jacques. “Commande predictive”. Orsay: École Supérieure D'Électricité, 1993. 140p.

Richalet, Jacques. “Initiation à la commande préditive”. Orsay: École Supérieure D'Électricité, 1997. 42p.

Rutenbar, R.A. “(when) will FPGAs kill asics?” IEE Design Automation Conference Proceedings, 2001. Proceedings, pp. 321 - 322 , 2001.

Samet, L. Masmoudi; N. Kharrat, M.W. Kamoun, L. “A digital PID Controller for Real Time and Multi Loop Control: a Comparative Study”. IEEE International Conference on Electronics, Circuits and Systems, pp. 291 – 296, vol.1, 1998.

Saramago, Marcos Antonio Porta. “Elaboração de Dispositivos Inteligentes Utilizando

Conceitos de Domótica Direcionados a Automação Hospitalar”. Faculdade de Engenharia Mecânica, Unicamp, 224pp, agosto de 2002.

Silva, Jose Edson Lima e. “Simulação e Controle Preditivo Linear (com modelo de convolução) e Não-linear (com modelo baseado em redes neurais artificiais) de Colunas Recheadas de Absorção com Reação Química”. Campinas Faculdade de Engenharia Química, Universidade Estadual de Campinas, 1997. 169 p. Tese (Doutorado).

Silva, N. C., Rosário, J. M., Lima, C. E., 2000e, “Actuator Selection for Antropomorphic

Robots Using Power and Heating Rate", In Mechatronics 2000 – The 7th Mechatronics Forum International Conferênce and Mechatronics Education Workshop, 6 a 8 de setembro de 2000. Atlanta, 5 p.

Silva, N. C., Rosário, J. M., Silva; Lima, C. E., 2000d, "Computer Aided Choice for DC Motors of Antromorphics Robots", Mechatronics 2000 – 1st IFAC-Conferênce Mechatronic System, 18 a 20 de setembro de 2000. Darmstadt – Germany, 5 p. 2000. 519 a 522 pre prints.

Snider, Greg; Shackleford, Barry; Carter, Richard J. "Attacking the Semantic Gap Between Application Programming Languages and Configurable Hardware". FPGA´s 2001 Proceedings, pp. 115-124, 2001.

Soeterboek, R. "Predictive Control: A Unified Approach". Cambridge: Prentice Hall International, 1992. 352 p.

Souza, J. P. et al. "Simulation and Experimental Result on Predictive Control of Robotic Joints". Orsay : École Supérieure D'Électricité, 2000

Souza, Jocarly Patrocínio de., "Implementação de Algoritmos Preditivos para Controle de Juntas Robóticas". Tese de Doutorado. Faculdade de Engenharia Mecânica, Unicamp, 130 pp., março de 2001.

Swaminathan, S.; Tessier, R.; Goeckel, D.; Burleson, W. "A Dynamically Reconfigurable Adaptative Viterbi Decoder". FPGA´02, pp 227-236, Monterey, California, USA, fevereiro 2002.

Synopsys, Inc. Disponível na internet via www, URL: http://www.synopysis.com, 2002.

Thor, Jonas; Akos, Denis M. "A Direct RF Sampling Multifrequency GPS Receiver". IEEE, 2002.

Triscend Corporation., Disponível na internet via www, URL: http://www.triscend.com/,

2002.

Valdes, M.D.; Nogueiras-Melendez, A.A.; Moure, M.J.; Mandado, E. “An Alternative Solution for the Implementation of Configurable Interfaces Oriented to Microprocessor-Based Control Systems”. IEEE International Symposium on Industrial Electronics Proceedings, ISIE '98, pp. 643 – 647, vol.2, 1998.

Valls, J.; Peiro, M.M. ;Sansaloni, T. ;Boemo, E. “A Study About FPGA-Based Digital Filters”. IEEE Workshop on Signal Processing Systems, SIPS 98, pp. 192 – 201, 1998.

Vargas, F.L.; Fagundes, R.D.R. Junior, D.B. “A FPGA-Based Viterbi Algorithm Implementation for Speech Recognition Systems”. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 1217 – 1220, vol.2 2001.

Wegrzyn, Marek; Adamski, Marian A., Monteito, Joao. L. “The Application of Reconfigurable Logic to Controller Design”. Control Enginnering Practice 6, pp. 879-887, published by Elselvier Science Ltd., 1998.

Woerner, D.F.; Lehman, D.H. “Faster, better, cheaper technologies used in the attitude and information management subsystem for the Mars Pathfinder mission”. Aerospace Applications Conference, 1995. Proceedings., IEEE , pp. 155 - 167 vol.2 , 1995.

Xlink Technology, Inc. Disponível na internet via www, URL: http://www.xilinx.com, 2002.

Yamaguchi, T.; Hashiyama, T.; Okuma, S. “A Study on Reconfigurable Computing System for Cryptography”. IEEE International Conference on Systems, Man, and Cybernetics, pp. 2965 – 2968, vol.4, 2000.

Yasunaga, M.; Kin, J. H.; Yoshihara, I. “The Application of Genetic Algorithms to the Desing of Reconfigurable Reasoning VLSI Chips”. FPGA´s 2000 Proceedings, pp.

118-125, 2000.

Ye, Zhi Alex; Sehnoy, Nagaraj; Banerjee, Prithviraj. “A C Compiler for a Processor with a Reconfigurable Functional Unit”. FPGA´s 2000 Proceedings, pp. 95-100, 2000.

Zelenovsky, Ricardo; Mendonça Alexandre. “PC: Um Guia Prático de Hardware e Interfaceamento”. 762 pp. Editora Mz Ltda., 2ª Edição, 1999.

Apêndice A – Controladores Preditivos

A.1 Aspectos gerais

A grande flexibilidade dos sistemas digitais tem facilitado o desenvolvimento de novas estratégias de controle digital. Entre estas novas estratégias encontram-se os controladores preditivos. Os controladores preditivos foram introduzidos por [Richarlet, 1978], [Cutler e Ramaker, 1979] e [Garcia et al., 1989]. Os trabalhos subseqüentes sobre este tema são muitos, sendo que alguns apresentam resultados de aplicações práticas em modelos industriais ou de laboratório. Entre as áreas de aplicação destes controladores encontram-se o controle de processos químicos e o controle de elementos eletro-mecânicos, desde simples motores até manipuladores robóticos e robôs móveis.

Os controladores preditivos trabalham com o princípio de antecipação da resposta do sistema. Devido a sua natureza, os controladores preditivos podem ser utilizados em processos simples ou em processos mais complexos, com atrasos de transporte, de fase não mínima, não lineares e ainda com limitações de resposta. Em função da complexidade do processo, estão presentes na literatura comparações entre o controlador preditivo e outros controladores como PID (controlador proporcional, integral e derivativo), controladores por alocação de pólos e controle linear quadrático. Dependendo do processo os controladores preditivos são equivalentes aos controladores comparados, mas existem duas características únicas que os qualificam como os mais adequados para o controle de determinados processos:

Em contraste com controle linear quadrático e controle por alocação de pólos podem também ser derivados para processos não lineares.

É a única metodologia que pode manipular restrições do processo durante o projeto do controlador. Uma vez que restrições do processo são uma prática comum, isto se torna um ponto importante. Esta é considerada uma das mais importantes características dos controladores preditivos.

Outras características dos controladores preditivos são:

O conceito de controle preditivo não está restrito a processos SISO. Controladores preditivos podem ser usados em processos MIMO.

Controle preditivo é uma metodologia aberta. Isto é, existem muitas maneiras de se projetar o controlador preditivo. Alguns controladores preditivos conhecidos são: GPC (*Generalized Predictive Controller*), DMC (*Dynamic Matrix Control*), EPSAC (*Extended Prediction Self-Adaptive Control*), PFC (*Predictive Functional Control*), EHAC (*Extended Horizon Adaptive Control*) e UPC (*Unified Predictive Control*) (Clarke et al., 1989) (Soeterboek, 1992).

Uma vez que controladores preditivos pertencem à classe de métodos de projeto de controladores baseados em modelos, um modelo do processo deve estar disponível.

Os controladores preditivos são classificados dentro da categoria dos controladores baseados em modelos. Um modelo do processo é usado para projetar o controlador. A Figura 78 ilustra este processo de projeto.

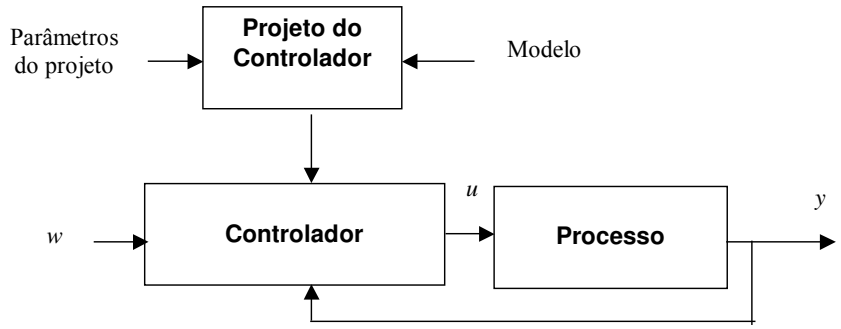


Figura 78- Controle baseado em modelo de processo.

Usualmente, controladores preditivos são projetados para atuar em tempo discreto. Contudo é possível projetar controladores preditivos para uso em tempo contínuo. O modo como controladores preditivos operam em sistemas SISO é ilustrado na Figura 76, onde as

escalas de tempo nos gráficos são relativas ao tempo discreto k. As escalas de tempo mostradas ao pé da Figura 79 são escalas de tempo absolutas. Além disto, u(k), y(k) e w(k) denotam a saída do controlador, a saída do processo e a saída desejada do processo na amostra k, respectivamente.

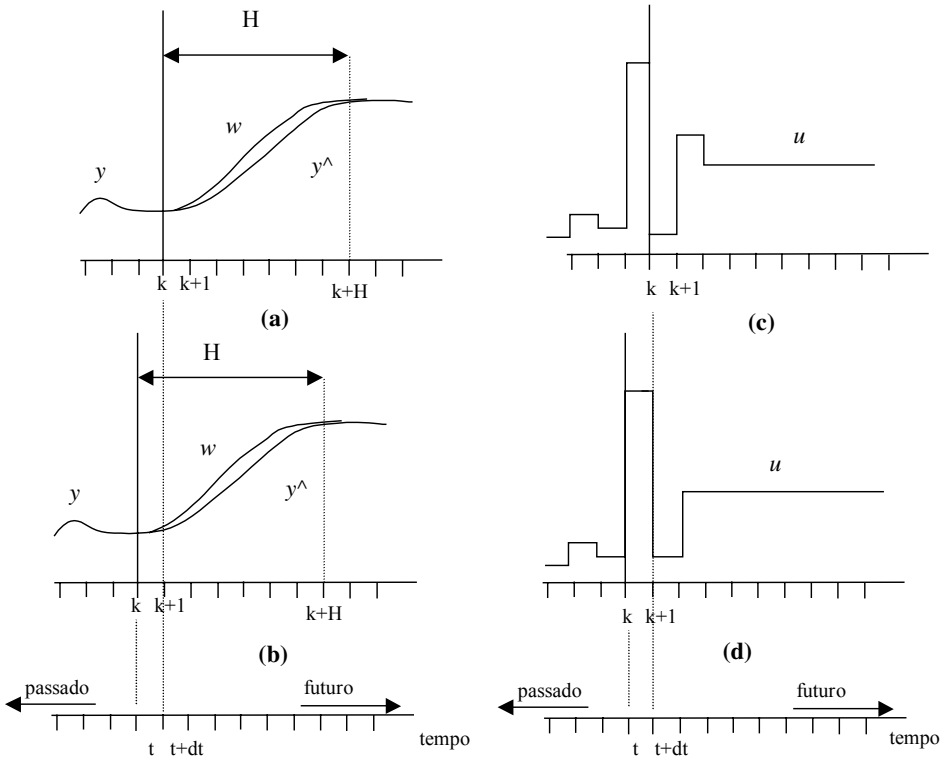


Figura 79 - Representação do princípio de operação dos controladores preditivos.

Agora são definidos:

$$\underline{u} = [u(k), u(k+1)\ldots,u(k+H-1)], \tag{A.1}$$

$$\underline{\hat{y}} = [\hat{y}(k+1),\ldots,\hat{y}(k+H)], \tag{A.2}$$

$$\underline{w} = [w(k+1),\ldots,w(k+H)], \tag{A.3}$$

onde H é o horizonte de predição e o símbolo $\hat{}$ denota estimação. Então o controlador preditivo calcula uma seqüência $\underline{u}$ de saídas futuras do controlador (Figura 79d) tal que a saída

preditiva do processo $\underline{\hat{y}}$ é próxima a saída desejada do processo $\underline{w}$ (Figura 79b). Esta saída desejada do processo é usualmente chamada de trajetória de referência e pode ser uma seqüência arbitrária de pontos.

Em lugar de usar a seqüência de saída do controlador determinada sobre o método de controlar o processo nas próximas H amostras, somente o primeiro elemento desta seqüência de saída do controlador u(k) é usado para controlar o processo. Na próxima amostra (em t+1), o procedimento é todo repetido, usando a última informação medida. Isto é chamado princípio do retrocesso do horizonte [Soeterboek, 1992] e é ilustrado na Figura 79a e Figura 79c, as quais mostram o que acontece no tempo $t+\Delta t$.

Assumindo que não há perturbações e que não existem erros de modelamento do processo a saída do processo preditivo $\hat{y}(k+1)$, predito no tempo t, é exatamente igual à saída do processo y(k), medida em $t+\Delta t$. Em geral, a seqüência de saída é diferente de uma seqüência obtida em uma amostra prévia, como ilustrado na Figura 79c. A razão para utilizar o princípio do retrocesso do horizonte é que isto permite compensar futuras perturbações ou erros de modelamento. Como resultado do princípio de retrocesso do horizonte, o horizonte sobre qual a saída do processo é predita é deslocado uma amostra no futuro a cada instante de amostragem.

A saída do processo é predita pelo uso do modelo do processo a ser controlado. Qualquer modelo que descreve um relacionamento entre entrada e saída do processo pode ser usado: modelos de função transferência, modelos de resposta ao degrau, modelos de espaço de estados e modelos não-lineares [Boucher et al., 1993] [Maia, 1994][Oliveira, 1997]. Se o processo é sujeito a perturbações, um modelo de perturbação ou ruído pode ser adicionado ao modelo do processo, permitindo que o efeito destas perturbações seja levado em conta.

De modo a definir quão bem a saída do processo preditivo segue a trajetória de referência, uma função de custo é usada. Por exemplo, uma função de custo simples é [Maia, 1994]:

$$J = \sum_{i=0}^{N} (y\hat{}(k+1) - w(k+1))^2 \ . \qquad \text{(A.4)}$$

O controlador miniminiza esta função de custo em relação à saída u do controlador, obtendo valores ótimos para esta saída. O erro é minimizado dentro do horizonte de predição.

Assim, calcular a seqüência de saída do controlador é transformado em um problema de otimização, ou mais especificamente, um problema de minimização. Usualmente, resolver um problema de minimização requer um procedimento interativo. Porém, uma solução analítica está disponível quando o critério é quadrático, o modelo é linear e invariante no tempo e não existem restrições. Idealmente, uma função de custo é baseada em especificações de projeto. Porém, pelo uso de uma função de custo qualquer, o problema de otimização pode ser difícil de resolver. Esta é a razão pela qual uma função de custo quadrática é usada em todos os controladores preditivos. Em função do controlador utilizado, adaptações são feitas nesta função de custo quadrática.

As especificações de projeto devem ser transladadas em parâmetros da função de custo quadrático, de modo que, quando esta função é minimizada, as especificações originais do projeto são atendidas (Soeterboek, 1992).

A.2 Relacionamento com Outros Métodos

Os controladores preditivos, controladores lineares quadráticos e controladores por alocação de pólos pertencem à classe de controladores baseados em modelo. Mais exatamente, controladores preditivos e lineares quadráticos são baseados na minimização de uma função de custo.

A.2. 1 Controle Linear Quadrático

Uma discussão com respeito ao controlador quadrático discreto pode ser encontrada em [Soeterboek, 1992]. Alguns dos métodos lineares quadráticos são baseados em realimentação de estados, enquanto outros são baseados em realimentação de saída.

Usando o enfoque de realimentação de estados, em controladores lineares quadráticos discretos o processo é dado por:

$$x(k+1) = \mathbf{A}x(k) + \mathbf{b}u(k) \tag{A.5}$$

$$y(k) = \mathbf{C}^T x(k), \tag{A.6}$$

onde x(k) denota o estado do processo e **A**, **b**, **C** são os parâmetros do processo.

Similar ao projeto de controladores preditivos, uma função de custo é minimizada em relação a um horizonte. Contudo, em contraste com controladores preditivos, o princípio do retrocesso do horizonte não é empregado. Esta é a diferença fundamental. A função de custo é minimizada apenas uma vez (em k = 0), resultando uma saída ótima para o controlador a partir da qual o controle de processo é tomado a cada instante de amostra. Assim, futuras perturbações e erros de modelamento não são levados em conta.

Em conclusão pode ser dito que, para um horizonte finito de predição, a diferença fundamental entre controle preditivo e o controle linear quadrático é o princípio do retrocesso do horizonte, com um horizonte de tamanho fixo empregado pelo controle preditivo em contraste com o horizonte de predição de tamanho decrescente do controlador linear quadrático. Como resultado disto, existem as sequintes diferenças fundamentais entre os dois controladores:

- Devido ao fato da estrutura do controlador linear quadrático ser determinada previamente,

eventuais restrições na saída do controlador não podem ser consideradas.

- Os projetos de controladores lineares quadráticos consideram apenas processos linearizados, enquanto que controladores preditivos permitem o uso de modelos não lineares para predição da saída.

- Uma desvantagem importante do controle linear quadrático é que, em geral, é bastante difícil traduzir as especificações de projeto em matrizes de peso na função de custo. Isto ocorre porque os estados de um sistema discreto são artificiais e não estão diretamente relacionados com os estados reais do processo. Regras práticas para a escolha dos parâmetros de projeto não estão prontamente disponíveis. Uma vantagem importante dos controladores lineares quadráticos de horizonte finito é que, sob condições quase gerais, o sistema em malha fechada é sempre estável. Esta alegação não pode ser usualmente feita sobre os controladores preditivos de horizonte finito [Soeterboek, 1992].

A.2.2 Projeto por Alocação de Pólos

Outro projeto de controlador baseado em modelos que é freqüentemente usado para o projeto de controladores discretos é o método de projeto por alocação de pólos. Uma discussão com respeito ao projeto por alocação de pólos pode ser encontrada em [Soeterboek, 1992], [Oliveira, 1997] e [Souza, 2000]. Neste método a saída do controlador é generalizada pela lei de controle linear:

$$Ru(k) = - S\, y(k) + T\, Sp, \qquad (A.7)$$

Onde R, S e T são polinômios do controlador no operador q^{-1} e Sp denota o *set-point*. Os controladores polinomiais R e S são selecionados pelos pólos em malha fechada desejados e T é selecionado em função do comportamento do sistema em malha fechada a variações no

set-point.

A maior dificuldade na aplicação deste método é decidir o posicionamento dos pólos em malha fechada. Ou seja, como transladar as especificações de projeto (geralmente fornecidas no domínio do tempo) em localização de pólos e zeros em malha fechada. Somente em casos de sistemas de baixa ordem sem zeros é possível fazer isto facilmente. Além disto, unido ao fato de que as saídas do controlador são geradas pela equação (A.7), restrições a estas mesmas saídas e uma trajetória de referência arbitrária não podem ser adicionadas ao sistema. Em contraste com o controle preditivo, este método somente pode ser aplicado a processos lineares (ou linearizados). Apesar disto, em muitas publicações o projeto por alocação de pólos também é discutido junto com o projeto de controladores preditivos [Clarke et al., 1987] [Soeterboek, 1992] [Oliveira, 1997].

O artigo de [Clarke et al., 1987] apresenta, pela primeira vez, uma descrição do projeto de controladores preditivos generalizados (GPC). É feita uma comparação (por simulação de um sistema SISO) com o controlador por alocação por pólos e o controlador linear quadrático. Este trabalho é freqüentemente referenciado por trabalhos posteriores. O trabalho de [Clarke et al., 1989] discute detalhadamente as propriedades dos controladores preditivos generalizados. São apresentados exemplos práticos do uso do GPC, como em manipuladores robóticos, em sistemas de aquecimento e em secadores, entre outros.

Outro artigo muito referenciado é [García et al., 1989]. Trata-se de um resumo comparativo de diversos métodos de controle preditivo e outros métodos tradicionais. Apresenta um interessante resumo histórico da matéria, chegando a discutir controladores preditivos não lineares.

A proposta de uma teoria unificada no projeto de controladores preditivos é feita em um livro escrito por [Soeterboek, 1992]. Devido ao fato do conceito de controle preditivo ser uma metodologia aberta, muitos controladores preditivos podem ser obtidos durante um projeto, cada um com características próprias. Embora, a primeira vista, a diferença entre estes

controladores tenda a parecer pequena, a mesma pode provocar comportamentos muitos diferentes do sistema em malha fechada. Como resultado, pode ser um tanto difícil determinar qual dos controladores preditivos pode ou deve ser usado. Visando resolver este problema, uma abordagem unificada do projeto de controladores preditivos é apresentada, permitindo o tratamento de cada problema dentro de uma mesma estrutura, reduzindo significativamente os custos de projeto. O controlador preditivo unificado unifica conhecidos controladores preditivos, como GPC, DMC, EPSA e EHAC. Conseqüentemente, outra vantagem da abordagem unificada proposta no livro é que, uma vez que este controlador preditivo unificado é analisado, conclusões podem ser tiradas a respeito dos controladores preditivos conhecidos mencionados.

O trabalho de [Richalet, 1993] explora detalhadamente o conceito do controlador preditivo funcional (PFC - Prédictive Functional Control). Em um trabalho posterior [Richalet, 1997], é apresentado um resumo, onde podem ser observados os princípios básicos do controle preditivo.

O trabalho de [Boucher et al., 1997] apresenta uma introdução e idéias gerais do controle preditivo. São considerados as linhas de trabalho de Clark, com o controle preditivo generalizado [Clark et al. , 1987] [Clark et al., 1989] e de Richalet [Richalet, 1993]. São apresentados resultados práticos do controle de uma planta de destilação de conhaque.

O trabalho de [Maia, 1994] é um exemplo de aplicação de controle preditivo a um processo químico. No caso foi apresentado um controlador DMC (*Dynamic Matrix Control*) controlando uma coluna de absorção. O trabalho utiliza a função de custo simplificada:

$$J = \sum_{i=0}^{N} (y\hat{}(k+1) - w(k+1))^2 \qquad \text{(A.8)}$$

O modelo do processo é gerado segundo a utilização do modelo de convolução. A identificação dos coeficientes do modelo gerado é feita *off-line.*

Ainda dentro da industria química, foi apresentado o trabalho de [Maia, 1994][Silva, 1997], no qual o modelo linear do processo também foi gerado por modelo de convolução.

Adicionalmente, foi gerado um modelo não linear através da utilização de redes neurais.

O trabalho intitulado Controle Preditivo Generalizado Aplicado a Sistemas Flexíveis foi apresentado por [Oliveira, 1997], sendo um trabalho de simulação sobre uma junta isolada de um manipulador robótico. São apresentados os resultados de simulação de controladores por alocação por pólos e do controlador preditivo GPC. A função de custo utilizada para o GPC é:

$$J = \sum_{i=N1}^{NY} [P(q^{-1})\hat{y}(k+i)] - D(q^{-1})w(k+i)]^2 + \sum_{i=1}^{NU} \rho \Delta u(k+i-1)^2 \qquad \text{(A.9)}$$

Onde são considerados parâmetros adicionais em relação à (A.9)como:

N1 - horizonte inicial de predição.

NY - horizonte final de predição.

NU - horizonte de controle.

ρ - constante de ponderação do sinal de controle.

$P(q^{-1})$ e $Q(q^{-1})$ são polinômios de ponderação que filtram os sinais de saída e da referência.

É apresentado um estudo sobre a robustez do sistema utilizando um controlador preditivo generalizado. Uma contribuição sobre a análise dos parâmetros de projeto do GPC para o controle de um sistema flexível é dada. Parâmetros como horizonte de predição e período de amostragem são considerados. Os trabalhos de [Clarke et al., 1987] e de [Soeterboek, 1992] são importantes na compreensão do texto.

Em [Dechechi, 1998] é proposto um trabalho com o controlador DMC aplicado a sistemas multivariáveis. O modelo do processo sofre um enfoque adaptativo.

Outro trabalho abordando a utilização do controle preditivo para sistemas multivariáveis, mas com uma nova abordagem para o modelo de referência é apresentado em [Codron, 1993]. Este trabalho descreve um algoritmo do controlador GPC com modelos de múltipla referência, seguido de análise da robustez destes algoritmos. São apresentados estudos

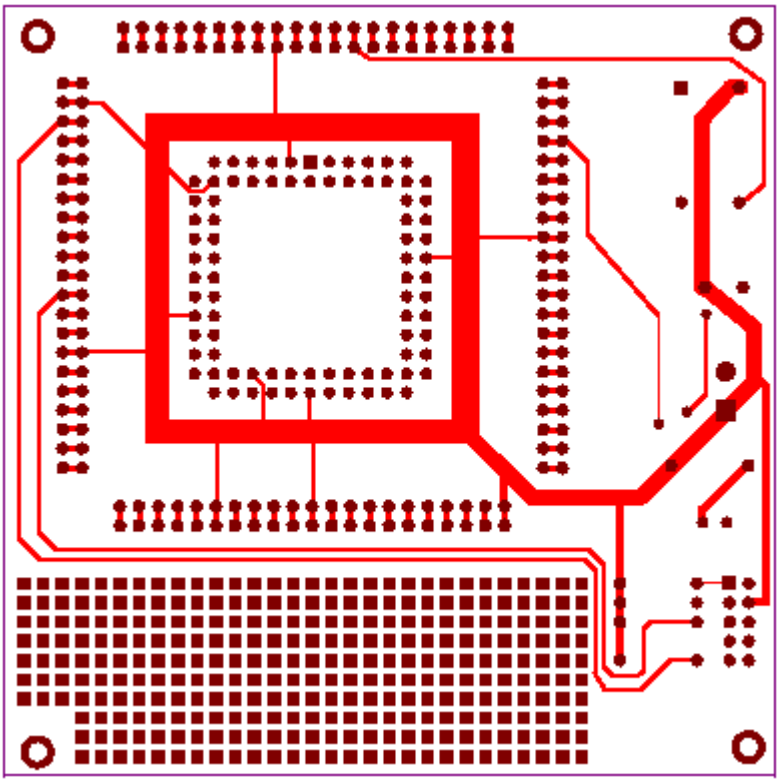
da atuação destes controladores em modelos dinâmicos de um helicóptero e de um braço flexível. Dois trabalhos diretamente associados ao tema de escolha de modelos de referência para controladores preditivos são apresentado em [Boucher et al., 1993] e [Codron et al., 1993].

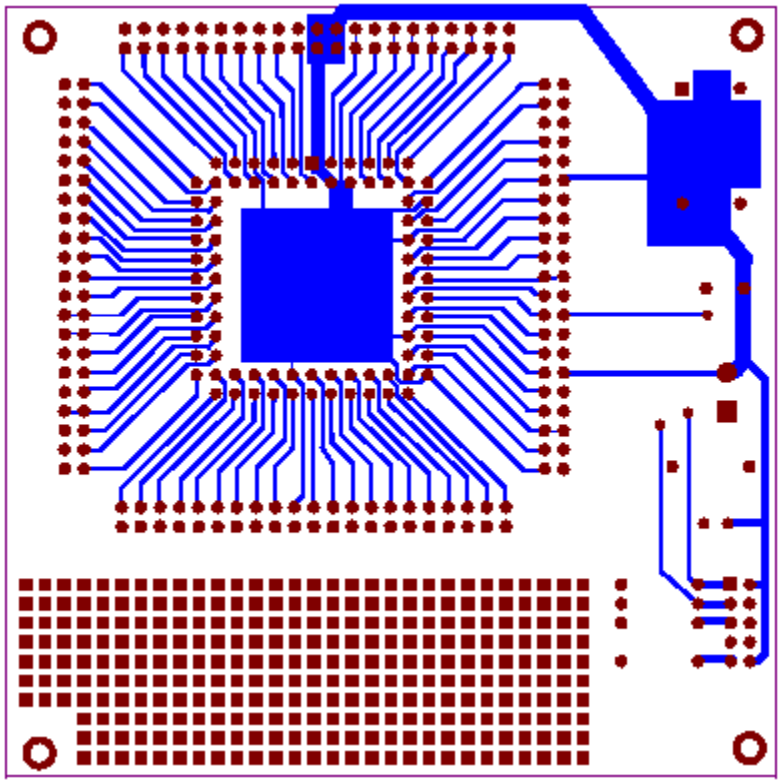
O controlador preditivo generalizado aplicado em máquinas ferramentas por comando numérico é apresentado em [Dumur, 1993]. Neste trabalho é detalhada a síntese do GPC segundo um controlador equivalente do tipo R, S e T. Esta estrutura polinomial facilita o tratamento numérico do controlador e é também explorada nos trabalhos de [Soeterboek, 1992], [Oliveira, 1997], [Dumur et al., 1992] e [Souza, 2000].

O trabalho de [Souza, 2000] compara o desempenho do controlador GPC com métodos de controle clássico para sistema SISO, a exemplo de controladores PID. Usa para tanto um modelo de uma junta robótica, analisando a resposta do sistema para diferentes padrões de excitação.

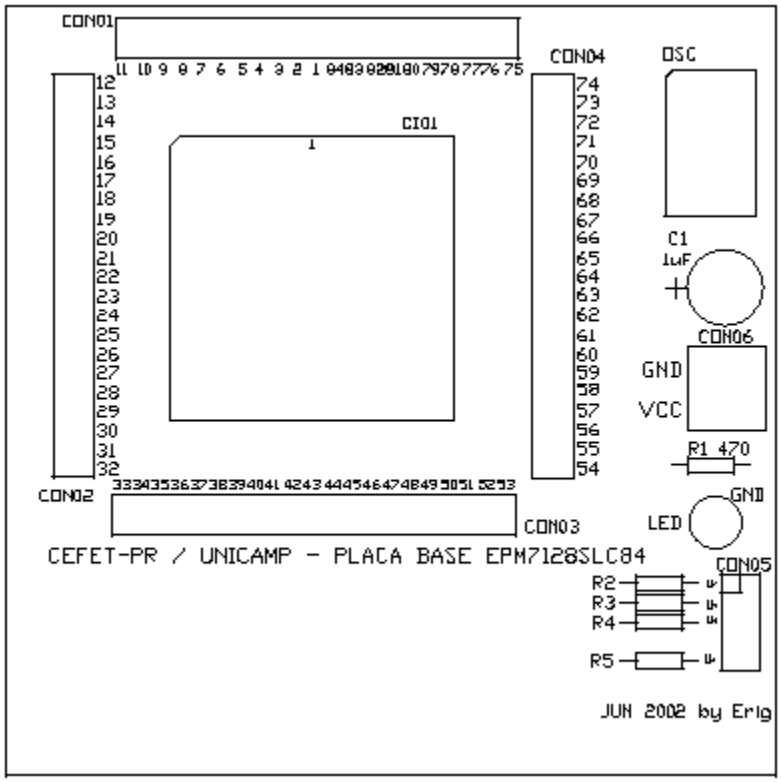
Apêndice B - Hardware

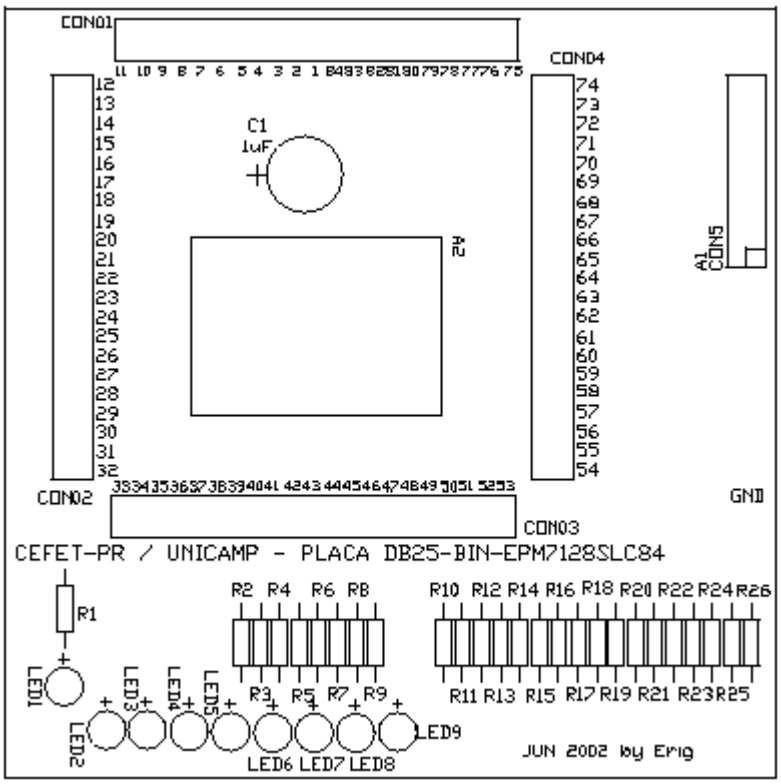
B.1 Placa Base

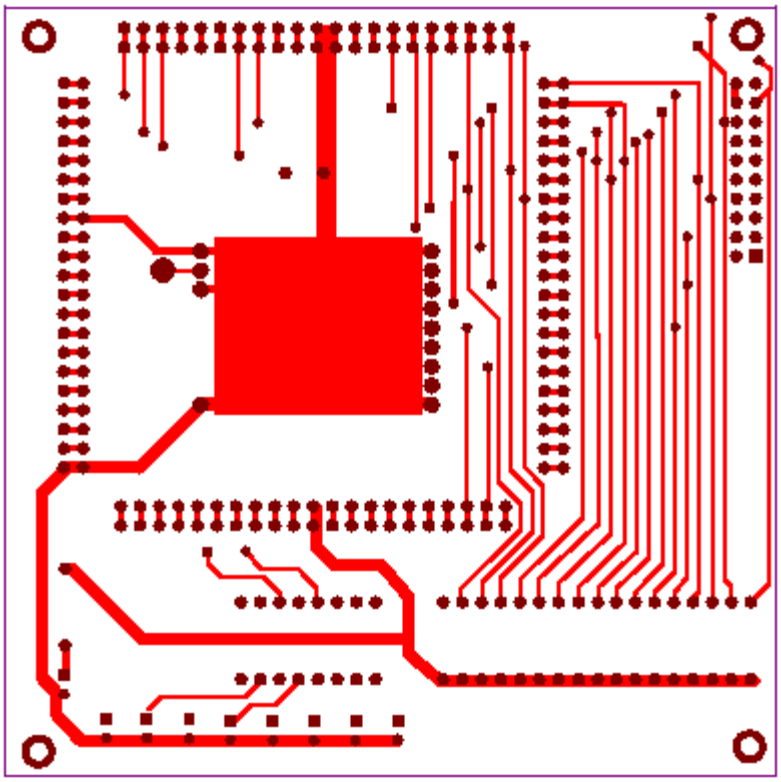


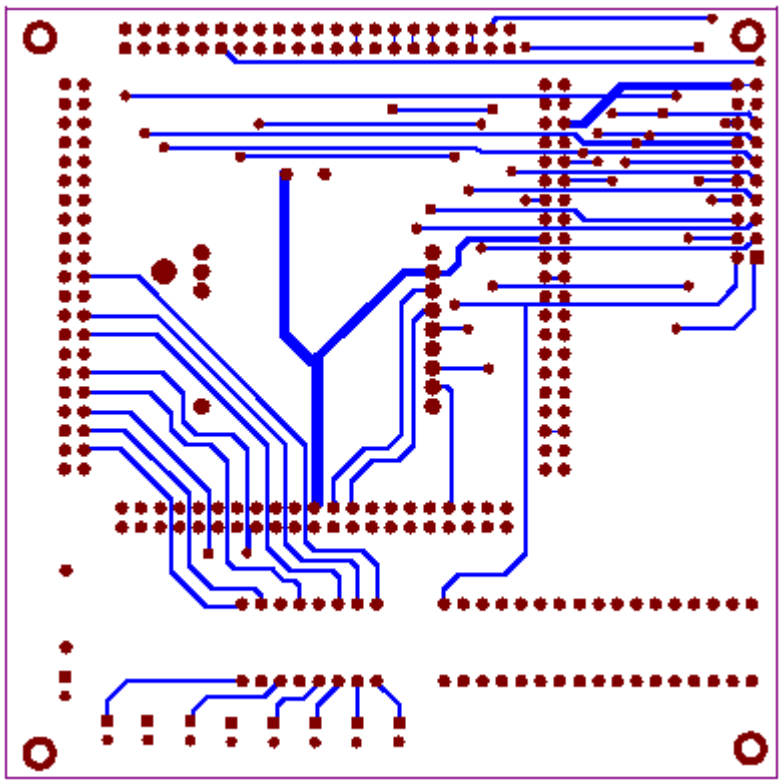


B.2 Placa de Interface de Rádio e da Paralela

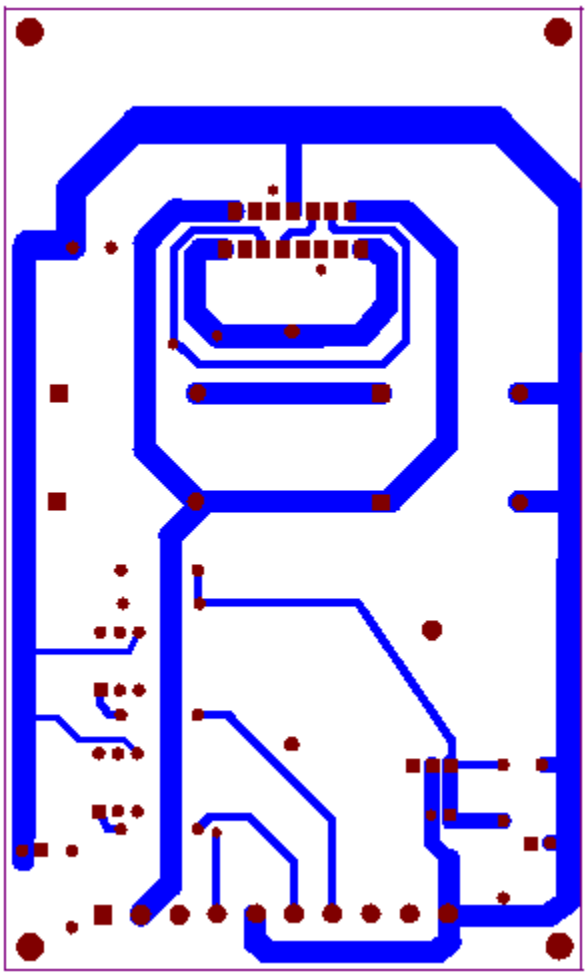


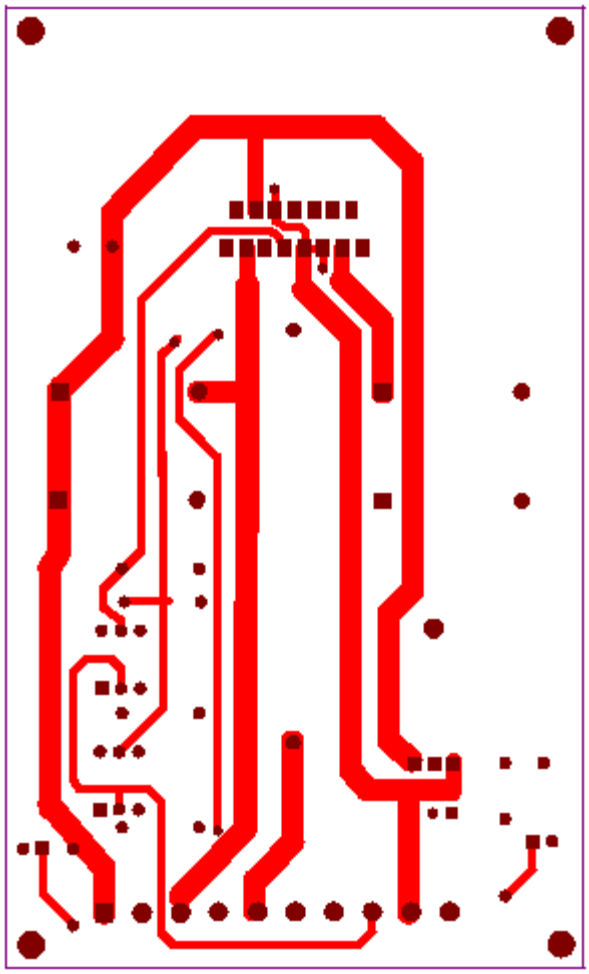

CON01
CON04
C1
1uF
A2
A1
CON5
CON02
CON03
GND
CEFET-PR / UNICAMP - PLACA DB25-BIN-EPM7128SLC84
R1
R2 R4 R6 R8
R10 R12 R14 R16 R18 R20 R22 R24 R26
R3 R5 R7 R9
R11 R13 R15 R17 R19 R21 R23 R25
LED1
LED2
LED3
LED4
LED5
LED6 LED7 LED8
LED9
JUN 2002 by Erig

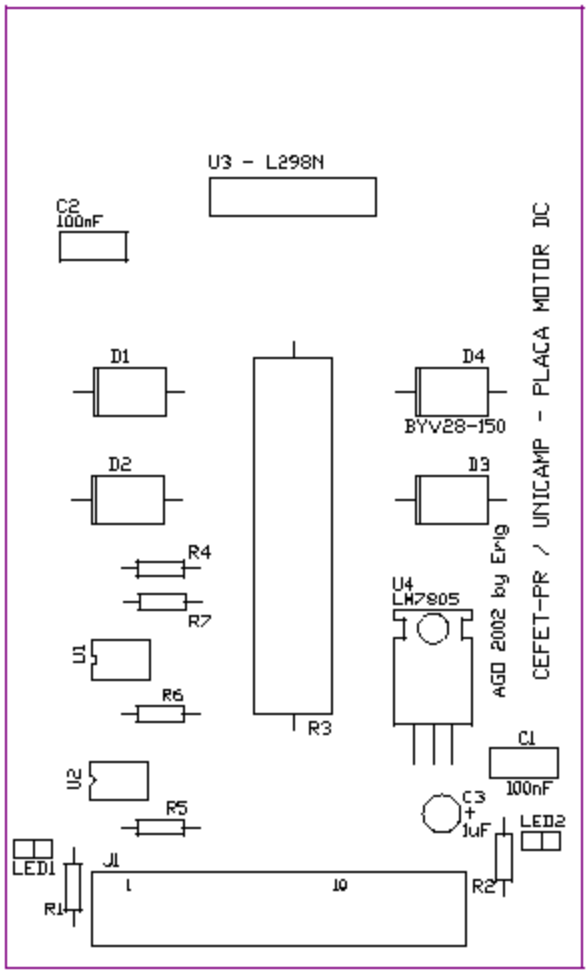




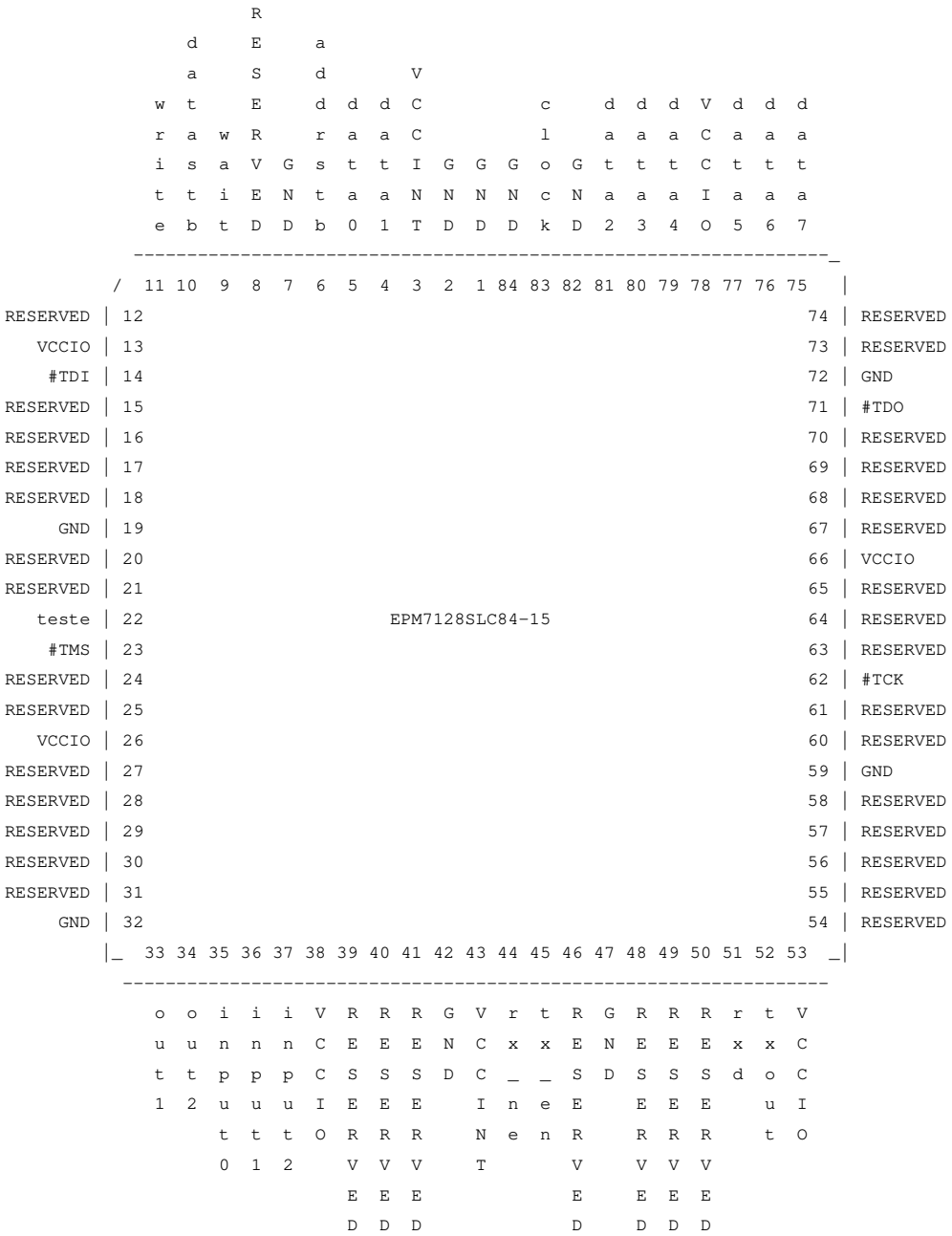
B.3 Placa de Potência

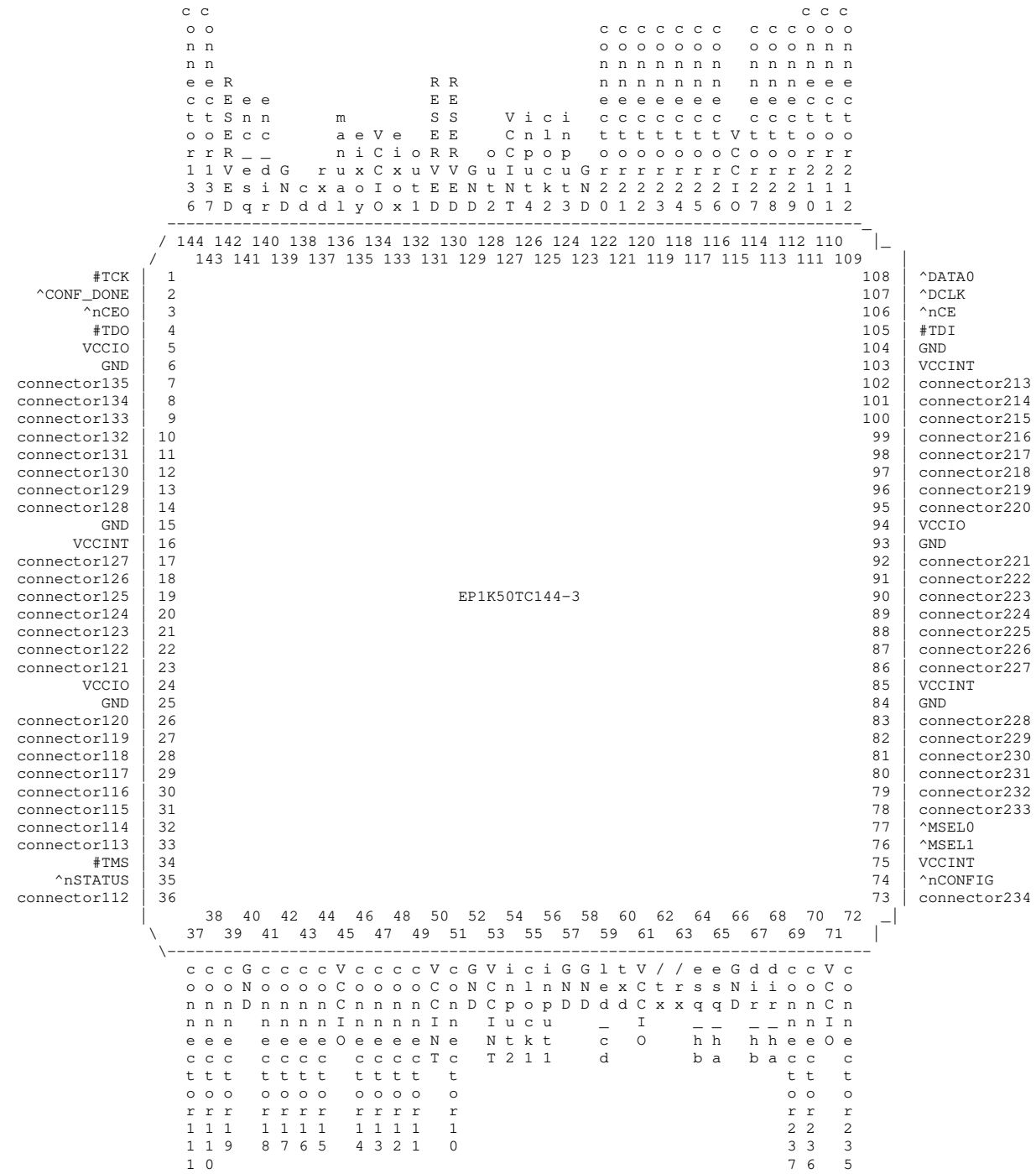





U3 - L298N
C2
100nF
D1
D2
D4
BYV28-150
D3
R4
R7
U1
R6
R3
U4
LM7805
AGD 2002 by Erig
CEFET-PR / UNICAMP - PLACA MOTOR DC
C1
100nF
U2
R5
C3
1uF
LED2
LED1
J1
1
10
R2
R1

B.2 Designação de Pinos das Epld Usadas




EP1K50TC144-3
#TCK 1
^CONF_DONE 2
^nCEO 3
#TDO 4
VCCIO 5
GND 6
connector135 7
connector134 8
connector133 9
connector132 10
connector131 11
connector130 12
connector129 13
connector128 14
GND 15
VCCINT 16
connector127 17
connector126 18
connector125 19
connector124 20
connector123 21
connector122 22
connector121 23
VCCIO 24
GND 25
connector120 26
connector119 27
connector118 28
connector117 29
connector116 30
connector115 31
connector114 32
connector113 33
#TMS 34
^nSTATUS 35
connector112 36
108 ^DATA0
107 ^DCLK
106 ^nCE
105 #TDI
104 GND
103 VCCINT
102 connector213
101 connector214
100 connector215
99 connector216
98 connector217
97 connector218
96 connector219
95 connector220
94 VCCIO
93 GND
92 connector221
91 connector222
90 connector223
89 connector224
88 connector225
87 connector226
86 connector227
85 VCCINT
84 GND
83 connector228
82 connector229
81 connector230
80 connector231
79 connector232
78 connector233
77 ^MSEL0
76 ^MSEL1
75 VCCINT
74 ^nCONFIG
73 connector234

Apêndice C - Software

C.1 Gerador de Coeficientes de Polinômio RST

```
////////////////////////////////////////////////////
// File: gpc.cpp
// Instituica: Laboratório de Automação é Robotíca  - UNICAMP - CEFET-PR
// Objetivo: Cálculo dos polinômios R, S e T para o controlador preditivo GPC
// Autor: Carlos Raimundo Erig Lima
// Email: erig@fem.unicamp.br
// Histórico:  10/08/2000 - tradução do algoritmo Matlab originário da SUPELEC

#include <conio.h>
#include <time.h>
#include "matrix.h"  // Copyright (c) 1997-2000 Techsoft Pvt. Ltd. (See License.Txt file.)
#ifndef _NO_NAMESPACE
using namespace std;
using namespace math;
#define STD std
#else
#define STD
#endif
#ifndef _NO_TEMPLATE
typedef matrix<double> Matrix;
#else
typedef matrix Matrix;
#endif
#ifndef _NO_EXCEPTION
#  define TRYBEGIN()     try {
#  define CATCHERROR()          } catch (const STD::exception& e) { \
                                        cerr << "Error: " << e.what() << endl; }
```

```
#else
#  define TRYBEGIN()
#  define CATCHERROR()
#endif

int main ()
{
//******limitado a polinômios de nono grau
  int             na,nb,i,j,k, cont;
  float aux1, lamda;
  int sizenum;
  int sizeden;
  int deltasize;
  int N1;
  int N2;
  int Nmax ;
  int Nu;
  /*********************************************************/
  /* entrada de dados
  /*********************************************************/
  printf("\nUNICAMP - CEFET - Laboratorio de Automacao e Robotica\n\n");

  printf("\nPara sistema discreto G(z) definido por:\n");
  printf("\n                        B(q^-1)");
  printf("\n                G(q^-1)=-------- ");
  printf("\n                        A(q^-1)\n");
  printf("\nEntre com o grau do polinomio A:");
  scanf("%d", &na);
      printf("\nEntre com o grau do polinomio B:");
  scanf("%d", &nb);
  clrscr();
  //cria as matrizes dos polinômios A e B
  Matrix A(1,na+1);
  Matrix B(1,nb+1);
  clrscr();
```

```
printf("\nUNICAMP - CEFET - Laboratorio de Automacao e Robotica\n\n");
    printf("\nPara A = 1 + a1*q^-1 +a2*q^-2 + ... + ana*q^-na\n");
A(0,0) = 1;
    printf("\na0 = 1\n");
    for (i = 0; i < na; i++)
{
    printf("\na%d = ",i+1);
    scanf("%f", &aux1);
    A(0,i+1)=aux1;
 }
clrscr();
printf("\nUNICAMP - CEFET - Laboratorio de Automacao e Robotica\n\n");
    printf("\nPara  B = b0 + b1*q^-1 + b2*q^-2 + ... + bnb*q^-nb\n");
for (i = 0; i < nb+1; i++)
{
    printf("\nb%d = ",i);
    scanf("%f", &aux1);
  B(0,i)=aux1;
}
clrscr();
printf("\nUNICAMP - CEFET - Laboratorio de Automacao e Robotica\n\n");
printf("\nEntre com o horizonte de predicao inicial da saida (N1): ");
scanf("%d", &N1);
printf("\nEntre com o horizonte de predicao final da saida (N2): ");
scanf("%d", &N2);
printf("\nEntre com o horizonte de predicao de controle (Nu): ");
scanf("%d", &Nu);
printf("\nEntre com o fator de ponderacao de controle (lamda): ");
scanf("%f", &lamda);
Nmax = max(Nu, N2);
printf("N1 = %d N2 = %d Nu = %d lamda = %f Nmax = %d", N1, N2, Nu , lamda, Nmax);
clrscr();
printf("\nUNICAMP - CEFET - Laboratorio de Automacao e Robotica\n\n");
```

```
/*********************************************************/
/*   cálculo do produto polinomial A*[1 -1]
/*********************************************************/

Matrix deltaA(1,na+2);
deltaA(0,0) = 1 ; // é sempre 1, pois é igual à a0
for (i = 0; i < na; i++)deltaA(0,i+1)= A(0,i+1)- A(0,i);
deltaA(0,na+1) = - A(0,na); //sempre
 /*********************************************************/
/*      inicio do loop principal
/*********************************************************/

//definição das matrizes usadas no processo
Matrix IF(N2-N1+1,na+1);
Matrix bigIF(Nmax+1,na+1);
Matrix G (N2-N1+1, Nu);
Matrix M (Nu, N2-N1+1);
Matrix U (Nu,Nu);
U.Unit();
Matrix bigG (Nmax+1, Nmax+1);
Matrix bigIH (Nmax+1,nb);
Matrix IH (N2-N1+1, nb);
bigG.Null();
bigIH.Null();
Matrix R (1,na+1);
R.Null();
Matrix S (1, nb+1);
S.Null();
Matrix T (1, N2+1);
T.Null();

    for (cont=N1; cont<Nmax+1; cont++)
{
    //define a primeira linha de uma matriz unitária de dimensão adequada
  Matrix qpoly (1, na+cont+1);
```

```
    qpoly.Null();
      qpoly(0,0) = 1;
    sizeden = na+1;
      sizenum = na+cont+1;
      deltasize = sizenum - sizeden;
              //divisão de dois polinômios ( qpoly/delta_A)

    Matrix quot(1,cont);
    Matrix g (1,cont);
    Matrix gh (1,cont+nb);
    gh.Null();

//     cout << "qpoly" << qpoly << endl;
 //   cout << "\quot" << quot << endl;
  //   cout << "\deltaA" << deltaA << endl;

      for (i = 0; i<deltasize; i++)
      {
                      quot(0,i) =( qpoly(0,i)/deltaA(0,0));
      for (j = 0; j < sizeden+1; j++)
      {
              k = j+i;
              qpoly(0,k) = qpoly(0,k)-quot(0,i)*deltaA(0,j);
      }
      }

              //multiplicação de dois polinômios (B * quot)

      for (i = 0; i<nb+1; i++)
      {
      for(j = 0; j <cont; j++)
                      {
              k = j+i;
              gh(0,k) = gh(0,k)+B(0,i)*quot(0,j);
       }
```

```
    }
    for (i = 0; i < cont;i++) bigG(cont,i)=gh(0,cont-1-i);
    for (i = 0; i <nb ;i++) bigIH(cont,i)=gh(0,cont+i);

    for(i=0; i<na+1;i++) bigIF(cont,i)= qpoly(0,cont+i);

     }//fim do laço principal

  for (j = 0; j<N2-N1+1; j++)
  {
     for(i=0; i<na+1;i++) IF(j,i)=bigIF(j+N1,i);
    for(i=0; i<Nu; i++)  G(j,i)=bigG(j+N1,i);
    for(i=0; i<nb; i++)  IH(j,i)=bigIH(j+N1,i);
  }

  for (j = 0; j<Nu; j++) U(j,j) = lamda*U(j,j);
  M = (~G*G+U).Solve(~G); //      M = (G'*G + lamda*eye(Nu))\G';
  S(0,0) = 1;  //sempre

  for (j = 0; j<N2-N1+1; j++)
  {
    for(i=0; i<na+1 ; i++) R(0,i) = R(0,i)+M(0,j)*IF(j,i);
    for(i=1; i<nb+1 ; i++) S(0,i) = S(0,i)+M(0,j)*IH(j,i-1);
  }
  for(i=N1; i<N2-N1+2;i++) T(0,i) = M(0,i-1);
     cout << "\nR = " << R << endl;
     cout << "\nS = " << S << endl;
     cout << "\nT = " << T << endl;
  return 0;
}
```

Apêndice D - Módulos VHDL

D.1 Aplicação Geral

```
---------------------------------------------------------------------------
--      UNICAMP - CEFET-PR
--      Laboratório de Automação Integrada e Robótica
---------------------------------------------------------------------------
-- Módulo: tx8
-- Descrição: codifica para transmissão serial por RF uma palavra de 8 bits.
-- O clock de referência é de 1Mz ou 1us.
-- O dado de entrada deve permanecer estável durante a codificação.
--
-- Arquivo: tx8.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 21-MAI-2001
-- Histórico:
---------------------------------------------------------------------------
ENTITY tx8 IS
        PORT(
                dados           : IN     BIT_VECTOR (7 DOWNTO 0);
                clock           : IN     BIT;       -- Pulso indicando um bit "0" válido
                start           : IN    BIT;
                reset           : IN    BIT;
                one         : OUT BIT;
                --two          : OUT BIT;
                txout    : OUT  BIT);                    -- sinal codificado
END tx8;

ARCHITECTURE tx8 OF tx8 IS
        ---------------------------------------------------
        -- SINAIS INTERNOS
        ---------------------------------------------------
```

```
    TYPE ESTADOS IS (espera,um,zero);
    SIGNAL estado : ESTADOS;
    SIGNAL aux,stant : BIT :='0';
    SIGNAL I : INTEGER RANGE 0 TO 7;
  SIGNAL contador : INTEGER RANGE 0 TO 31;
BEGIN
    PROCESS (clock,dados,start)
    BEGIN
                IF ((clock'EVENT) AND (clock='1')) THEN
CASE estado IS

                WHEN espera =>

                IF (start = '1' AND stant = '0') THEN -- ocorre um comando de inicio de
conversão
                        stant <= '1';
                        contador <= 0;
                        aux <= '0';
                        I <= 0;
                        estado <= um;
                ELSIF (start = '0') THEN -- prepara para escrita em memória
                        stant <= '0';
                END IF;

                WHEN um =>
                                        contador <= contador + 1;
                                        IF (reset = '1') THEN
                                                contador <= 0;
                                                aux <= '0';
                                                I <= 0;
                                                estado <= espera;
                                        END IF;
                                        IF (aux = '0' AND dados(I) = '1' AND contador = 24)
THEN
                                                aux <= '1';
```

```
                    contador <= 0;
                ELSIF (contador >= 24) THEN
                    contador <= 0;

                    aux <= '0';
                    estado <= zero;
                END IF;

            WHEN zero =>
                IF (reset = '1') THEN
                    contador <= 0;
                    aux <= '0';
                    I <= 0;
                    estado <= espera;
                END IF;
                contador <= contador + 1;
                IF (aux = '0' AND dados(I) = '1' AND contador = 24)
THEN
                    aux <= '1';
                    contador <= 0;
                ELSIF (contador >= 24) THEN
                        contador <= 0;
                        aux <= '0';
                    IF(I = 7) THEN
                        I <= 0;
                        estado <= espera;
                    END IF;
                    IF (I < 7 ) THEN
                        I <= I + 1;
                        estado <= um;
                    END IF;
                END IF;
```

```
END CASE;
                        END IF;

        END PROCESS;
                txout <= '1' WHEN (estado = um)  ELSE '0';
                one <= '0' WHEN (estado = espera) ELSE '1';

END tx8;
--------------------------------------------------------------------------------
-- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
--
--------------------------------------------------------------------------------
-- Módulo: freq50
-- Descrição: gera uma frequencia submultipla da frequencia de entrada.
-- A frequencia de saída é a frequencia de entrada divida pelo duas vezes reg.
-- O sinal maxfreq deve ser um clock com o dobro da frequencia desejada.
-- Arquivo: gerador.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 05 de janeiro de 2002
-- Histórico:
--------------------------------------------------------------------------------
ENTITY freq50 IS
        PORT(
                reg                             : IN     INTEGER RANGE 0 TO 31;                 --
valor de controle do ciclo de trabalho
                freqin                  : IN     BIT;                                    -- Sinal de clock
para sincronização
                freqout                 : OUT BIT);                                      --    pulso
com o perfil desejado
END freq50;
```

```
ARCHITECTURE freq50 OF freq50 IS
      ---------------------------------------------------
      -- SINAIS INTERNOS
      ---------------------------------------------------
      TYPE ESTADOS IS (zero, um);
      SIGNAL estado : ESTADOS;
      SIGNAL contador : INTEGER RANGE 0 TO 31;
      SIGNAL aux : INTEGER RANGE 0 TO 31;

BEGIN
      PROCESS (freqin, reg)
      BEGIN

   IF ((freqin'EVENT) AND (freqin ='1')) THEN
                        aux <= reg - 1;
CASE estado IS

                        WHEN zero =>
                        contador <= contador - 1;
                        IF (contador = 0) THEN
                                contador <= aux;
                                estado <= um ;
                        END IF;

                        WHEN um =>
                        contador <= contador - 1;
                        IF (contador = 0) THEN
                                contador <= aux;
                                estado <= zero ;
                        END IF;
END CASE;
END IF;
END PROCESS;
                freqout <= '1' WHEN estado = zero   ELSE '0';
END freq50;
```

```
--------------------------------------------------------------------------------
-- UNICAMP _ CEFET-PR  -  Doutorado
-- Projeto  Cadeira de Rodas
--------------------------------------------------------------------------------
-- Módulo: janela
-- Descrição: detecta janela de tempo de 1ms (+/- 0.05 ms)
-- Arquivo: janela1.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 21-MAI-2001
-- Histórico:
--------------------------------------------------------------------------------
ENTITY janela IS
	PORT(
		clock	: IN	BIT;	-- Sinal de clock de 0,01ms - 100kHz		reset	: IN	BIT;		-- Sinal de reset
		sinal	: IN	BIT;		-- Pulsos de largura variável - busca-se valores entre 1ms +/- 0,05ms
		tipo	: OUT	BIT;	-- define o nível lógico
		saida	: OUT BIT);		-- Gera um pulso na rampa de descida do sinal se o mesmo atende exigências
END janela;

ARCHITECTURE janela OF janela IS
	-------------------------------------------------------
	-- SINAIS INTERNOS
	-------------------------------------------------------
	TYPE ESTADOS IS (high,low,valid);
	SIGNAL estado : ESTADOS;
	SIGNAL contador : INTEGER RANGE 0 TO 63;
BEGIN
	PROCESS (clock,sinal)
	BEGIN
   IF (clock'EVENT) AND (clock='1')  THEN
```

```
CASE estado IS
                WHEN high =>
            contador <= contador + 1;
                IF (contador = 63) THEN estado <= low; -- longo demais
                END IF;
                IF (sinal = '0')  THEN -- fim de pulso
                        IF ( contador <60) AND (contador >40) THEN
                        contador <= 0;
                        tipo <= '1';
                        estado <= valid;
                        END IF;
                        IF ( contador <35) AND (contador >15) THEN
                        contador <= 0;
                        tipo <= '0';
                        estado <= valid;
                        END IF;
                        IF ( contador >=60) OR ((contador <= 40) AND ( contador >=35))
OR (contador <= 15) THEN
                        contador <= 0;
                        estado <= low;
                        END IF;
                END IF;

                WHEN low =>
                IF (sinal = '1') THEN
            estado <= high;
                END IF;

                WHEN valid =>
            contador <= contador + 1;
            IF (contador= 10) THEN
                contador <= 0;
                estado <= low;
                END IF;
```

```
END CASE;

   END IF;

      END PROCESS;
            saida <= '1' WHEN (estado = valid) ELSE '0';
END janela;
--------------------------------------------------------------------------------
-- UNICAMP _ CEFET-PR  -  Doutorado
--
-- Projeto robô móvel - controle por RF - julho 2000
--
--------------------------------------------------------------------------------
-- Módulo: delay20
-- Descrição: gera uma janela de tempo de 204,8 us
-- Arquivo: delay20.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 21-MAI-2001
-- Histórico:
--------------------------------------------------------------------------------
ENTITY delay20 IS
      PORT(
            clock   : IN    BIT;      -- Sinal de clock de 10MHz      /128   reset   :      IN
      BIT;            -- Sinal de reset
            start   : IN    BIT;            -- Pulsos de largura variável - busca-se valores entre
1ms +/- 0,05ms
            saida   : OUT BIT:='0');            -- Gera um pulso na rampa de descida do
sinal se o mesmo atende exigências
END delay20;

ARCHITECTURE delay20 OF delay20 IS
```

```
	-------------------------------------------------------
	-- SINAIS INTERNOS
	-------------------------------------------------------
	TYPE ESTADOS IS (high,low);
	SIGNAL estado : ESTADOS := low;
	SIGNAL contador : INTEGER RANGE 0 TO 7 :=0;
BEGIN
	PROCESS (clock,start)
	BEGIN
   IF (clock'EVENT) AND (clock='1')  THEN

CASE estado IS

			WHEN low =>
			IF (start = '1') THEN
		estado <= high;  -- espera um pouco antes de subir
			END IF;

			WHEN high =>
		contador <= contador + 1;
			IF (contador = 7) THEN
			contador <= 0;  -- zera contador
			estado <= low; -- fim da janela
			END IF;

END CASE;

   END IF;

	END PROCESS;
		saida <= '1' WHEN (estado = high) ELSE '0';
END delay20;
```

D.2 Cadeira de Rodas Automatizada

```
---------------------------------------------------------------------------
-- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
--
-- Projeto Cadeira de Rodas
--
---------------------------------------------------------------------------
-- Módulo: deceixox
-- Descrição: converte um sinal digital  Tx de período variável
-- (1.5 ms < Tx < 8.8 ms ) em palavra binária. Tx representa o eixo de
-- deslocamento frente-ré de um joystick. Em Tx, o menor período está associado
-- ao deslocamento para frente.
-- A palavras binária é usada para definir as velocidades dos motores que tracionam a
-- cadeira de rodas.
-- A seguinte tabela é implementada:
-- Tx < 3 ms                                => bitx0 = 0 e bitx1 = 0
-- 3 <= Tx < 4,5 ms                 => bitx0 = 1 e bitx1 = 0
-- 4,5 ms <= Tx < 7,5 ms   => bitx0 = 0 e bitx1 = 1
-- 7,5 <= Tx                                => bitx0 = 1 e bitx1 = 1
-- Arquivo: deceixox.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 05 de novembro de 2001
-- Histórico:
---------------------------------------------------------------------------
ENTITY deceixox IS
      PORT(
            clock          : IN     BIT;      -- Sinal de clock para sincronização 20kHz
                                                                        -- (20 pulsos por ms)
            eixox          : IN    BIT;            -- Sinal do eixo x do joystick
            bitx0          : OUT  BIT;            -- bit de seleção
            bitx1          : OUT  BIT;            -- bit de seleção
            validax        : OUT  BIT);           -- pulso de valor valido na saída
```

```
END deceixox;

ARCHITECTURE deceixox OF deceixox IS
      -------------------------------------------------------
      -- SINAIS INTERNOS
      -------------------------------------------------------
      TYPE ESTADOS IS (zero, um, dois,tres);
      SIGNAL estado : ESTADOS;
      SIGNAL contador : INTEGER RANGE 0 TO 127;
      SIGNAL eixoxa :BIT; -- estado anterior de eixo x

BEGIN
      PROCESS (clock, eixox)
      BEGIN
   IF ((clock'EVENT) AND (clock='1')) THEN

CASE estado IS
                        -- o estado zero espera o inicio de uma nova contagem
                        WHEN zero =>
                        -- detecta subida de eixox
                        IF eixox = '1' AND eixoxa = '0' THEN -- bit 0 = 1
                        eixoxa <= '1';
                        estado <= um;
                        END IF;
                        IF eixox = '0' AND eixoxa = '1' THEN -- bit 0 = 1
                        eixoxa <= '0';
                        END IF;

                        -- o estado um está contado a espera de um evento de fim de contagem
                        -- , ou seja, eixox vai para zero.
                        WHEN um =>
                        IF eixox = '1'  AND contador < 120 THEN contador <= contador +1;
                        END IF;
                        IF eixox = '0' THEN
                        eixoxa <= '0';
```

```
estado <= dois;
END IF;

-- o estado dois caracteriza o fim de contagem de um possível valor válido
WHEN dois =>

IF contador >= 10 AND contador <=100 THEN  -- é um valor válido (
estado <= tres;
END IF;
IF contador < 10 OR contador >100 THEN  -- é um valor não válido
                                                                                --
possível ruído
contador <= 0;
estado <= zero;
END IF;

-- são valores possíveis de serem convertidos
WHEN tres =>
-- tabela
IF contador <30 THEN
bitx0 <= '0';
bitx1 <= '0';
END IF;
IF contador >= 30 AND contador <45 THEN
bitx0 <= '1';
bitx1 <= '0';
END IF;
IF contador >= 45 AND contador <70 THEN
bitx0 <= '0';
bitx1 <= '1';
END IF;
IF contador >= 70 THEN
bitx0 <= '1';
bitx1 <= '1';
END IF;
```

```
                    contador <= 0;
                    estado <= zero;


END CASE;


   END IF;

      END PROCESS;
            validax <= '1' WHEN (estado = tres)  ELSE '0';
END deceixox;

--------------------------------------------------------------------------------
-- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
--
-- Projeto Cadeira de Rodas
--
--------------------------------------------------------------------------------
-- Módulo: deceixoy
-- Descrição: converte um sinal digital  Ty de período variável
-- (2.2 ms < Tx < 8.6 ms ) em palavra binária. Ty representa o eixo de
-- deslocamento esquerda-direita de um joystick. Em Ty, o menor período está associado
-- ao deslocamento para esquerda.
-- A palavra binária é usada para definir as velocidades dos motores que tracionam a
-- cadeira de rodas.
-- A seguinte tabela é implementada:
-- Ty < 4 ms                                          => bity0 = 0 e bity1 = 0
-- 4 ms <= Ty < 7 ms                                  => bity0 = 1 e bity1 = 0
-- 7 ms <= Ty                                         => bity0 = 0 e bity1 = 1
--
-- Arquivo: deceixoy.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
```

```
-- Data de criação: 05 de novembro de 2001
-- Histórico:
--------------------------------------------------------------------------------
ENTITY deceixoy IS
	PORT(
		clock		: IN	BIT;	-- Sinal de clock para sincronização 20kHz
									-- (20 pulsos por ms)
		eixoy		: IN	BIT;		-- Sinal do eixo y do joystick
		bity0		: OUT	BIT;		-- bit de seleção
		bity1		: OUT	BIT;		-- bit de seleção
		validay		: OUT	BIT);		-- pulso de valor valido na saída
END deceixoy;

ARCHITECTURE deceixoy OF deceixoy IS
	-------------------------------------------------------
	-- SINAIS INTERNOS
	-------------------------------------------------------
	TYPE ESTADOS IS (zero, um, dois,tres);
	SIGNAL estado : ESTADOS;
	SIGNAL contador : INTEGER RANGE 0 TO 127;
	SIGNAL eixoya :BIT; -- estado anterior de eixo x

BEGIN
	PROCESS (clock, eixoy)
	BEGIN
   IF ((clock'EVENT) AND (clock='1')) THEN

CASE estado IS
			-- o estado zero espera o inicio de uma nova contagem
			WHEN zero =>
			-- detecta subida de eixoy
			IF eixoy = '1' AND eixoya = '0' THEN -- bit 0 = 1
			eixoya <= '1';
			estado <= um;
			END IF;
```

```
IF eixoy = '0' AND eixoya = '1' THEN -- bit 0 = 1
eixoya <= '0';
END IF;


-- o estado um está contado a espera de um evento de fim de contagem
-- , ou seja, eixox vai para zero.
WHEN um =>
IF eixoy = '1'  AND contador < 120 THEN contador <= contador +1;
END IF;
IF eixoy = '0' THEN
eixoya <= '0';
estado <= dois;
END IF;


-- o estado dois caracteriza o fim de contagem de um possível valor válido
WHEN dois =>

IF contador >= 10 AND contador <= 100 THEN  -- é um valor válido
estado <= tres;
END IF;
IF contador < 10 OR contador >100 THEN  -- é um valor não válido
                                                                                --
possível ruído
contador <= 0;
estado <= zero;
END IF;

-- são valores possíveis de serem convertidos
WHEN tres =>
-- tabela
IF contador  <30 THEN
bity0 <= '0';
bity1 <= '0';
```

```
                    END IF;
                    IF contador >= 30 AND contador <70 THEN
                    bity0 <= '1';
                    bity1 <= '0';
                    END IF;
                    IF contador >= 70  THEN
                    bity0 <= '0';
                    bity1 <= '1';
                    END IF;
                    contador <= 0;
                    estado <= zero;


END CASE;


   END IF;

      END PROCESS;
            validay <= '1' WHEN (estado = tres)  ELSE '0';
END deceixoy;
--------------------------------------------------------------------------------
-- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
--
-- Projeto Cadeira de Rodas
--
--------------------------------------------------------------------------------
-- Módulo: mapa
-- Descrição: converte as informações binárias dos eixos do joystick
-- em comandos para os drivers de potência dos motores da cadeira de rodas.
-- Estes comandos permitem definir qual motor é acionado, em qual sentido e
-- com que velocidade.
-- a sequinte tabela é implementada:
--  |bitx1 | bit x0 | bity1 | bity0 | habesq | habdir | sendir| senesq| PWM |
-- | 0  |   0  |   0 |  0  |   0  |  1   |  1  |  1  | 0 |
```

```
-- | 0 |  0 |  0 | 1 |  1 | 1 |  1 | 1 | 0 |
-- | 0 |  0 |  1 | 0 |  1 | 0 |  1 | 1 | 0 |
-- | 0 |  0 |  1 | 1 |  0 | 0 |  1 | 1 | 0 |
-- | 0 |  1 |  0 | 0 |  0 | 1 |  1 | 1 | 1 |
-- | 0 |  1 |  0 | 1 |  1 | 1 |  1 | 1 | 1 |
-- | 0 |  1 |  1 | 0 |  1 | 0 |  1 | 1 | 1 |
-- | 0 |  1 |  1 | 1 |  0 | 0 |  1 | 1 | 1 |
-- | 1 |  0 |  0 | 0 |  0 | 0 |  0 | 0 | 1 |
-- | 1 |  0 |  0 | 1 |  0 | 0 |  0 | 0 | 1 |
-- | 1 |  0 |  1 | 0 |  0 | 0 |  0 | 0 | 1 |
-- | 1 |  0 |  1 | 1 |  0 | 0 |  0 | 0 | 1 |
-- | 1 |  1 |  0 | 0 |  1 | 0 |  0 | 0 | 0 |
-- | 1 |  1 |  0 | 1 |  1 | 1 |  0 | 0 | 0 |
-- | 1 |  1 |  1 | 0 |  0 | 1 |  0 | 0 | 0 |
-- | 1 |  1 |  1 | 1 |  0 | 0 |  0 | 0 | 0 |

-- Arquivo: mapa.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 05 de novembro de 2001
-- Histórico:
--------------------------------------------------------------------------------
ENTITY mapa IS
      PORT(
            clock                : IN     BIT;       -- Sinal de clock para sincronização
            bitx0                : IN    BIT;     -- Sinal do eixo x do joystick
            bitx1                : IN    BIT;     -- Sinal do eixo x do joystick
            bity0                : IN    BIT;     -- Sinal do eixo y do joystick
            bity1                : IN    BIT;     -- Sinal do eixo y do joystick
            habesq               : OUT  BIT;    -- habilita motor esquerdo
            senesq               : OUT   BIT;    -- sentido do motor esquerdo
            habdir               : OUT  BIT;    -- habilita motor direito
            sendir               : OUT   BIT;    -- sentido do motor direito
            pwm                  : OUT  BIT);   -- seleciona velocidade alta ou baixa
```

```
END mapa;

ARCHITECTURE mapa OF mapa IS
    ---------------------------------------------------
    -- SINAIS INTERNOS
    ---------------------------------------------------

BEGIN
    PROCESS (clock,bitx0, bitx1, bity0, bity1)
    BEGIN
  IF ((clock'EVENT) AND (clock='1')) THEN

                IF bitx1 = '0' AND bitx0 = '0' AND bity1 = '0' AND bity0 = '0'THEN
                habesq <= '0';
                habdir <= '1';
                senesq <= '1';
                sendir <= '1';
                pwm <= '0';
                END IF;
                IF bitx1 = '0' AND bitx0 = '0' AND bity1 = '0' AND bity0 = '1'THEN
                habesq <= '1';
                habdir <= '1';
                senesq <= '1';
                sendir <= '1';
                pwm <= '0';
                END IF;
                IF bitx1 = '0' AND bitx0 = '0' AND bity1 = '1' AND bity0 = '0'THEN
                habesq <= '1';
                habdir <= '0';
                senesq <= '1';
                sendir <= '1';
                pwm <= '0';
                END IF;
                IF bitx1 = '0' AND bitx0 = '0' AND bity1 = '1' AND bity0 = '1'THEN
                habesq <= '0';
```

```
habdir <= '0';
senesq <= '1';
sendir <= '1';
pwm <= '0';
END IF;
IF bitx1 = '0' AND bitx0 = '1' AND bity1 = '0' AND bity0 = '0'THEN
habesq <= '0';
habdir <= '1';
senesq <= '1';
sendir <= '1';
pwm <= '1';
END IF;
IF bitx1 = '0' AND bitx0 = '1' AND bity1 = '0' AND bity0 = '1'THEN
habesq <= '1';
habdir <= '1';
senesq <= '1';
sendir <= '1';
pwm <= '1';
END IF;
IF bitx1 = '0' AND bitx0 = '1' AND bity1 = '1' AND bity0 = '0'THEN
habesq <= '1';
habdir <= '0';
senesq <= '1';
sendir <= '1';
pwm <= '1';
END IF;
IF bitx1 = '0' AND bitx0 = '1' AND bity1 = '1' AND bity0 = '1'THEN
habesq <= '0';
habdir <= '0';
senesq <= '1';
sendir <= '1';
pwm <= '1';
END IF;
IF bitx1 = '1' AND bitx0 = '0' AND bity1 = '0' AND bity0 = '0'THEN
habesq <= '0';
```

```
habdir <= '0';
senesq <= '0';
sendir <= '0';
pwm <= '1';
END IF;
IF bitx1 = '1' AND bitx0 = '0' AND bity1 = '0' AND bity0 = '1'THEN
habesq <= '0';
habdir <= '0';
senesq <= '0';
sendir <= '0';
pwm <= '1';
END IF;
IF bitx1 = '1' AND bitx0 = '0' AND bity1 = '1' AND bity0 = '0'THEN
habesq <= '0';
habdir <= '0';
senesq <= '0';
sendir <= '0';
pwm <= '1';
END IF;
IF bitx1 = '1' AND bitx0 = '0' AND bity1 = '1' AND bity0 = '1'THEN
habesq <= '0';
habdir <= '0';
senesq <= '0';
sendir <= '0';
pwm <= '1';
END IF;
IF bitx1 = '1' AND bitx0 = '1' AND bity1 = '0' AND bity0 = '0'THEN
habesq <= '1';
habdir <= '0';
senesq <= '0';
sendir <= '0';
pwm <= '0';
END IF;
IF bitx1 = '1' AND bitx0 = '1' AND bity1 = '0' AND bity0 = '1'THEN
habesq <= '1';
```

```
                    habdir <= '1';
                    senesq <= '0';
                    sendir <= '0';
                    pwm <= '0';
                    END IF;
                    IF bitx1 = '1' AND bitx0 = '1' AND bity1 = '1' AND bity0 = '0'THEN
                    habesq <= '0';
                    habdir <= '1';
                    senesq <= '0';
                    sendir <= '0';
                    pwm <= '0';
                    END IF;
                    IF bitx1 = '1' AND bitx0 = '1' AND bity1 = '1' AND bity0 = '1'THEN
                    habesq <= '0';
                    habdir <= '0';
                    senesq <= '0';
                    sendir <= '0';
                    pwm <= '0';
                    END IF;


   END IF;

      END PROCESS;
--              valida <= '1' WHEN (estado = dois)  ELSE '0';
END mapa;
-------------------------------------------------------------------------------
-- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
--
-------------------------------------------------------------------------------
-- Módulo: gerador
-- Descrição: gera os pulsos de trajetória com período proporcional a entrada de 8 bits
-- O valor 255 decimal não gera pulsos (período infinito).
-- O valor 0 gera um pulso com a máxima frequencia de saída desejada.
-- O sinal maxfreq deve ser um clock com o dobro da frequencia desejada.
```

```
-- Arquivo: gerador.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 05 de janeiro de 2002
-- Histórico:
--------------------------------------------------------------------------------
ENTITY gerador IS
	PORT(
		reg			: IN	INTEGER RANGE 0 TO 255;		-- valor de controle do ciclo de trabalho
		maxfreq			: IN	BIT;				-- Sinal de clock para sincronização
		en			: IN	BIT;				-- sinal de habilitação
		inibe		: IN	BIT;				-- sinal de inibição
		perfil		: OUT	BIT);				-- pulso com o perfil desejado
END gerador;

ARCHITECTURE gerador OF gerador IS
	-------------------------------------------------------
	-- SINAIS INTERNOS
	-------------------------------------------------------
	TYPE ESTADOS IS (zero, um);
	SIGNAL estado : ESTADOS;
	SIGNAL contador : INTEGER RANGE 0 TO 255;
	SIGNAL aux : INTEGER RANGE 0 TO 255;



BEGIN
	PROCESS (maxfreq, reg, inibe)
	BEGIN
   IF ((maxfreq'EVENT) AND (maxfreq ='1')) THEN
```

```
                    IF (en = '1') THEN
                    aux <= reg;
                    END IF;

CASE estado IS

                    WHEN zero =>
                    contador <= contador - 1;
                    IF (contador = 0) THEN
                            contador <= aux;
                            estado <= um ;
                    END IF;

                    WHEN um =>
                    contador <= contador - 1;
                    IF (contador = 0) THEN
                            contador <= aux;
                            estado <= zero ;
                    END IF;
END CASE;

   END IF;

      END PROCESS;
              perfil <= '1' WHEN estado = zero  AND inibe = '0' AND reg < 255 ELSE '0';
             END gerador;
```

D.3 Robô Móvel Experimental

```
--------------------------------------------------------------------------------
--      UNICAMP _ CEFET-PR
--      Laboratório de Automação Integrada e Robótica
--------------------------------------------------------------------------------
-- Módulo: encoder16..vhd
-- Descrição: implementa contador de pulsos dentro de uma janela de tempo definida pelo sinal de entrada "janela"
-- O valor máximo de contagem é de 65536 pulsos.
-- No caso de ultrapassagem deste valor é gerado um sinal de overflow.

-- Arquivo: encoder16.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 21-MAI-2002
-- Histórico:
-- criado em 15-ago-2002
--------------------------------------------------------------------------------
ENTITY encoder16 IS
      PORT(
            clock          : IN    BIT;       -- clock usado na máquina de estados (pulsos do encoder)
            clr            : IN    BIT;          -- zera contador interno e saída reg
            janela         : IN    BIT;       -- define a base de tempo em que será feita a contagem (encoder com maior resolução
                                                                                  -- necessita de base de tempo menor (calibração)
            flag1          : IN    BIT;       -- bit que inibe alteração de reg durante uma leitura (precedência sobre o clr)
            ovflow    : OUT   BIT;
            reg                     : OUT INTEGER RANGE 0 TO 65535);          -- valor de saida do encoder
```

```
        END encoder16;

ARCHITECTURE encoder16 OF encoder16 IS
    -------------------------------------------------------
    -- SINAIS INTERNOS
    -------------------------------------------------------
    TYPE ESTADOS IS (espera, conta, final);
    SIGNAL estado : ESTADOS;
    SIGNAL janant : BIT ;
    SIGNAL contador : INTEGER RANGE 0 TO 65536;

BEGIN
    PROCESS (clock,janela)
    BEGIN
  IF ((clock'EVENT) AND (clock='0')) THEN -- aquarda transição descendente

    CASE estado IS

                WHEN espera =>
                IF (janela = '0' and janant = '1') THEN        -- espera transição descendente
do sinal val
                        janant <= '0';
                END IF;
                IF (janela = '1' and janant = '0') THEN        -- transição ascendente do sinal
janela
                        janant <= '1';
                        contador <= 0;
                        ovflow <= '0';
                        estado <= conta;
                END IF;
                IF (clr = '1') THEN
                        contador <= 0;
                        estado <= final;
                END IF;
```

```
                WHEN conta =>                                   -- armazena as
proximos 7 bits como endereço

                IF (janela = '0' AND janant = '1') THEN     -- transição descendente do
sinal janela -> fim de contagem
                        estado <= final;
                END IF;
                IF (contador <65536) THEN
                        contador <= contador + 1;
                ELSIF
                        ovflow <= '1';
                END IF;
                IF (clr = '1') THEN
                        contador <= 0;
                        estado <= final;
                END IF;

                WHEN final =>    -- armazena na saída o valor do contador
                IF ( flag1 = '0') THEN
                reg <= contador;
                END IF;
                contador <= 0;
                estado <= espera;

END CASE;
END IF;
END PROCESS;
END encoder16;

--------------------------------------------------------------------------
-- UNICAMP _ CEFET-PR  -  Doutorado
--
-- Projeto  bussula--
--------------------------------------------------------------------------
-- Módulo: contapasso
```

```
-- Descrição: detecta janela de tempo de 1ms (+/- 0.05 ms)
-- Arquivo: contapasso.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 21-MAI-2001
-- Histórico:
--------------------------------------------------------------------------------
ENTITY janela IS
      PORT(
            clock   : IN     BIT;       -- Sinal de clock de 0,01ms - 100kHz                reset    :
IN    BIT;              -- Sinal de reset
            sinal   : IN    BIT;              -- Pulsos de largura variável - busca-se valores entre
1ms +/- 0,05ms
            tipo    : OUT   BIT;       -- define o nível lógico
            saida   : OUT BIT);            -- Gera um pulso na rampa de descida do sinal se o
mesmo atende exigências
END janela;

ARCHITECTURE janela OF janela IS
      ------------------------------------------------------
      -- SINAIS INTERNOS
      ------------------------------------------------------
      TYPE ESTADOS IS (high,low,valid);
      SIGNAL estado : ESTADOS;
      SIGNAL contador : INTEGER RANGE 0 TO 63;
BEGIN
      PROCESS (clock,sinal)
      BEGIN
   IF (clock'EVENT) AND (clock='1')  THEN

CASE estado IS
                        WHEN high =>
                  contador <= contador + 1;
                        IF (contador = 63) THEN estado <= low; -- longo demais
```

```
                    END IF;
                    IF (sinal = '0')  THEN -- fim de pulso
                        IF ( contador <60) AND (contador >40) THEN
                        contador <= 0;
                        tipo <= '1';
                        estado <= valid;
                        END IF;
                        IF ( contador <35) AND (contador >15) THEN
                        contador <= 0;
                        tipo <= '0';
                        estado <= valid;
                        END IF;
                        IF ( contador >=60) OR ((contador <= 40) AND ( contador >=35))
OR (contador <= 15) THEN
                        contador <= 0;
                        estado <= low;
                        END IF;
                    END IF;

                    WHEN low =>
                    IF (sinal = '1') THEN
                estado <= high;
                    END IF;

                    WHEN valid =>
                contador <= contador + 1;
                IF (contador= 10) THEN
                    contador <= 0;
                    estado <= low;
                    END IF;
END CASE;
  END IF;
    END PROCESS;
            saida <= '1' WHEN (estado = valid) ELSE '0';
END janela;
```

```
--------------------------------------------------------------------------------
--      UNICAMP _ CEFET-PR
--      Laboratório de Automação Integrada e Robótica
--------------------------------------------------------------------------------
-- Módulo: dec_com.vhd
-- Descrição: maquina de estados para decodificação de campos de comando de escrita
-- ou leitura em registradores internos do módulo de comunicação local--
--
-- São usados os seguintes campos para escrita em memória:
-- 1 bit de identificação de escrita ou leitura (bit = 1 implica em escrita)
-- 7 bits de campo de endereço (128 possíveis regsitradores)
-- 8 bits de dados
-- São usados os seguintes campos para leitura em memória:
-- 1 bit de identificação de escrita ou leitura (bit = 0 implica em leitura)
-- 7 bits de campo de endereço (128 possíveis regsitradores)

-- Arquivo: dec_com.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 21-MAI-2002
-- Histórico:
-- criado em 21-mai-2002
-- modificado em 10-out-2002 - campo de dados passado de 16 bits para 8 bits
--------------------------------------------------------------------------------
ENTITY dec_com IS
      PORT(
            clock          : IN    BIT;      -- clock usado na máquina de estados
            val            : IN    BIT;      -- Pulso indicando ocorrencia de bit válido
            tipo           : IN   BIT;             -- estado do bit válido, "1" ou "0"
            dados          : out   BIT_VECTOR (7 DOWNTO 0); -- byte de dados a ser
escrito em um registrador
            endereco       : out   BIT_VECTOR (6 DOWNTO 0); -- endereço do registrador
a ser escrito ou lido
```

```
            h_escrita       : OUT BIT;          -- habilita escrita se h_escrita = '1'
            h_leitura       : OUT BIT;          -- habilita leitura se h_leitura = '1'
--          txstart  : OUT BIT;          -- inicio da transmissão de um pacote serial
            not_tx          : OUT BIT;          -- controle de transmissão do módulo Bim-
433
            not_rx          : OUT BIT);         -- controle de transmissão do módulo Bim-
433
            END dec_com;

ARCHITECTURE dec_com OF dec_com IS
    -------------------------------------------------------
    -- SINAIS INTERNOS
    -------------------------------------------------------
    TYPE ESTADOS IS (modo, enderec, dado, final);
    SIGNAL estado : ESTADOS;
    SIGNAL aux : BIT;
    SIGNAL valant : BIT ; --valant iniciaizado com zero
    SIGNAL contador : INTEGER RANGE 0 TO 31;
    SIGNAL I : INTEGER RANGE 0 TO 6;
    SIGNAL J : INTEGER RANGE 0 TO 7;
BEGIN
    PROCESS (clock,tipo)
    BEGIN
  IF ((clock'EVENT) AND (clock='0')) THEN -- aquarda transição descendente


    CASE estado IS

                WHEN modo =>
                IF (val = '0' AND valant = '1') THEN        -- espera transição descendente
do sinal val
                        valant <= '0';
                        IF (tipo = '1') THEN  -- prepara para escrita em memória
                                aux <= '1';
                                not_tx <= '1';
            -- seta recepção
```

```
                    not_rx <= '0';
                    I <= 0;
                    estado <= enderec;
                END IF;

                IF (tipo = '0') THEN   -- prepara para leitura em memória
                    aux <= '0';
                    I <= 0;
                    not_tx <= '0';
        -- seta transmissão
                    not_rx <= '1';
                    estado <= enderec;
                END IF;
            ELSIF (val = '1' AND valant = '0') THEN    --    aquarda    uma    transição
ascendente
                valant <= '1';
            END IF;

            WHEN enderec =>                                        --    armazena    as
proximos 7 bits como endereço

            IF (val = '0' AND valant = '1') THEN          -- espera transição descendente
do sinal val
                valant <= '0';
                endereco(I) <= tipo;
                I <= I +1;
        --IF (contador = 8 AND aux = '1') THEN
                IF (I = 6 AND aux = '1') THEN
                    J <= 0;
                    estado <= dado;
                END IF;
                --IF (contador = 9 AND aux = '0') THEN
                IF (I = 6 AND aux = '0') THEN
                    J <= 0;
                    estado <= final;
                END IF;
```

```
                ELSIF (val = '1' AND valant = '0') THEN    --    aquarda    uma    transição
ascendente
                        valant <= '1';
                END IF;



                WHEN dado =>                                                              --
habilita shift register de dados por

        -- dezesseis transições nos bits de entrada
                IF (val = '0' AND valant = '1') THEN          -- espera transição descendente
do sinal val
                        valant <= '0';
                        dados(J) <= tipo ;
                        J <= J +1;
                        IF (J = 7) THEN
                        contador <= 0;
                        estado <= final;
                        END IF;
                ELSIF (val = '1' AND valant = '0') THEN    --    aquarda    uma    transição
ascendente
                        valant <= '1';
                END IF;

                WHEN final =>    -- gera pulsos de havbilitação
                contador <= contador +1;
                IF (aux = '1' AND contador = 10) THEN     -- habilita escrita
                        h_escrita <= '1';
                END IF;
                IF (aux = '0' AND contador = 10) THEN     -- habilita leitura
                        h_leitura <= '1';
                END IF;
                IF (contador = 30) THEN      -- desabilita memória
                        h_escrita <= '0';
                        h_leitura <= '0';
```

```
                    contador <= 0;
                    estado <= modo;
                END IF;

    END CASE;

      END IF;

        END PROCESS;
    END dec_com;
```

D.4 Robô de Competição

```
    ---------------------------------------------------------------------------
    -- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
    --
    ---------------------------------------------------------------------------
    -- Módulo: gerador
    -- Descrição: gera os pulsos de trajetória com período proporcional a entrada de 8 bits
    -- O valor 255 decimal não gera pulsos (período infinito).
    -- O valor 0 gera um pulso com a máxima frequencia de saída desejada.
    -- O sinal maxfreq deve ser um clock com o dobro da frequencia desejada.
    -- Arquivo: gerador.VHD
    -- EPLD:
    -- Autor: Carlos Raimundo Erig Lima
    -- Data de criação: 05 de janeiro de 2002
    -- Histórico:
    ---------------------------------------------------------------------------
    ENTITY gerador IS
        PORT(
            reg                 : IN    INTEGER RANGE 0 TO 255;         --
    valor de controle do ciclo de trabalho
            maxfreq             : IN    BIT;                            --  Sinal
    de clock para sincronização
```

```
        en                  : IN  BIT;                  -- sinal de habilitação
        inibe               : IN  BIT;                  -- sinal de inibição
        perfil              : OUT BIT);                 -- pulso com o perfil desejado
END gerador;

ARCHITECTURE gerador OF gerador IS
    -------------------------------------------------------
    -- SINAIS INTERNOS
    -------------------------------------------------------
    TYPE ESTADOS IS (zero, um);
    SIGNAL estado : ESTADOS;
    SIGNAL contador : INTEGER RANGE 0 TO 255;
    SIGNAL aux : INTEGER RANGE 0 TO 255;
BEGIN
    PROCESS (maxfreq, reg, inibe)
    BEGIN
  IF ((maxfreq'EVENT) AND (maxfreq ='1')) THEN

                IF (en = '1') THEN
                aux <= reg;
                END IF;
CASE estado IS
                WHEN zero =>
                contador <= contador - 1;
                IF (contador = 0) THEN
                        contador <= aux;
                        estado <= um ;
                END IF;
                WHEN um =>
                contador <= contador - 1;
                IF (contador = 0) THEN
                        contador <= aux;
```

```
                        estado <= zero ;
                    END IF;
END CASE;
END IF;
END PROCESS;
            perfil <= '1' WHEN estado = zero  AND inibe = '0' AND reg < 255 ELSE '0';
END gerador;
--------------------------------------------------------------------------------
-- UNICAMP - LABORATORIO DE AUTOMAÇÃO E ROBÓTICA
--
-- Projeto Gerra de Robos - setembro de 2001
--------------------------------------------------------------------------------
-- Módulo: duty
-- Descrição: gera perfis de trabalho. São possíveis 16 perfis entre
-- com difeentes ciclos de trabalhos. Todos os perfis são de 2kHz
-- Arquivo: duty.VHD
-- EPLD:
--
-- Autor: Carlos Raimundo Erig Lima
-- Data de criação: 05 de setembro de 2001
-- Histórico:
--------------------------------------------------------------------------------
ENTITY duty IS
    PORT(
        clock           : IN  BIT;    -- Sinal de clock para sincronização
        falha       : IN  BIT;        -- sinal de inibição
        entrada0        : IN  BIT;        -- bit de seleção
        entrada1        : IN  BIT;        -- bit de seleção
        entrada2        : IN  BIT;        -- bit de seleção
        perfil      : OUT BIT);       -- pulso com o perfil desejado
END duty;

ARCHITECTURE duty OF duty IS
    ------------------------------------------------------
    -- SINAIS INTERNOS
```

```
    ---------------------------------------------------
    TYPE ESTADOS IS (zero, um);
    SIGNAL estado : ESTADOS;
    SIGNAL contador : INTEGER RANGE 0 TO 32;

BEGIN
    PROCESS (clock,entrada0, entrada1, entrada2, falha)
    BEGIN
  IF ((clock'EVENT) AND (clock='1')) THEN


CASE estado IS
                    -- NÍVEL 1
                    WHEN zero =>
                    contador <= contador +1;
                    IF entrada0 = '0' AND entrada1 = '0' AND entrada2 = '0' THEN
                    estado <= um;
                    END IF;
                    IF entrada0 = '1' AND entrada1 = '0' AND entrada2 = '0' AND contador = 2
THEN
                    estado <= um;
                    END IF;
                    IF entrada0 = '0' AND entrada1 = '1' AND entrada2 = '0'AND contador = 4
THEN
                    estado <= um;
                    END IF;
                    IF entrada0 = '1' AND entrada1 = '1' AND entrada2 = '0' AND contador = 6
THEN
                    estado <= um;
                    END IF;
                    IF entrada0 = '0' AND entrada1 = '0' AND entrada2 = '1' AND contador = 8
THEN
                    estado <= um;
                    END IF;
                    IF entrada0 = '1' AND entrada1 = '0' AND entrada2 = '1' AND contador =
10 THEN
```

```
                    estado <= um;
                    END IF;
                    IF entrada0 = '0' AND entrada1 = '0' AND entrada2 = '1' AND contador =
16 THEN
                    estado <= um;
                    END IF;
                    IF entrada0 = '1' AND entrada1 = '0' AND entrada2 = '1' AND contador =
19 THEN
                    estado <= um;
                    END IF;

                    -- NÍVEL 0
                    WHEN um =>
                    contador <= contador +1;
                    IF contador = 20 THEN
                    contador <= 0;
                    estado <= zero;
                    END IF;

END CASE;

END IF;
END PROCESS;
                perfil <= '1' WHEN (estado = zero AND falha = '0'AND (entrada0 /= '0' OR
entrada1 /= '0' OR entrada2 /= '0')) OR (entrada0 = '1' AND entrada1 = '1' AND entrada2 = '1')
ELSE '0';
END duty;
```

Apêndice E – Artigos Publicados

1) Lima, C. R. E.; Rosário, J. M.; Silva, N. C. A Proposal of Flexible Architecture for Mobile Robotics - 7th Mechatronics Forum International Conference, Atlanta, US, 2000.

2) Silva, N. C; Rosário, J. M.; Lima, C. R. E. Actuator Selection for Antropomorphic Robots Using Power And Heating Hating - 7th Mechatronics Forum International Conference, Atlanta, US, 2000.

3) Silva, N. Cardoso da; Rosário, J. Maurício; Lima, Carlos R. E. Computer Aided Choise for DC Motors of Antromorphics Robots – 1st IFAC Conference on Mechatronics Systems – Germany, 2000.

4) Silva; N. C. da; Rosário, J. M.; Pimenta, K. B.; Cassemiro, E. R; Lima; Carlos R. E. Modelagem dinâmica adaptada ao processo de seleção dos atuadores das juntas para dispositivos com cadeia cinemática robótica antropomorfica. CBA 2000 – Brasil, 2000.

5) Lima, Carlos Raimundo Erig; Rosário, João Maurício. Utilização de Circuitos Lógicos Programáveis para Controle de Atuadores em Robôs Móveis. Revista Robótica, Portugal, 2° trimestre de 2001.

6) Lima, C. R. E.; Dumur D.; Rosário, J. M. Determinação do Modelo de Referência Interna para um Controlador GPC Embarcado. APLICON 2001, editado por J. M. Balthazar, V. A. Oliveira e J. M. Rosário, EEUSP São Carlos, 2001.

7) Silva, N. Cardoso da; Lima; Carlos R. E.; Rosário; J. Maurício. Programmable Logical

Devices, A Computational Solution to Simplify Prostheses Control. 22nd Iberian Latin-American Congress on Computational Methods in Engineering, Campinas, Brazil, 2001.

8) Silva, N. C. da; Silva, J. V. L. da; Rosário, J. M.; Lima C. R. E. Estudo, Desenvolvimento e Processo de Fabricação de Próteses Antropomórficas Ativas Destacando a Prototipagem Rápida e a Usinagem. 16th Brazilian Congress of Mechanical Engineering, COBEM 2001.

9) Lima, C. R. E.; Rosário, J. M.; Dumur D. Open Architecture Design for Mobile Robotics. 2nd IFAC Workshop on Intelligent Assembly and Disassenbly, IAD'2001, Canela, Brazil, 2001.

10) Lima, C. R. E.; Rosário, J. M. Implementação de Cadeira de Rodas com Dispositivos Lógicos Programáveis. 1º. Congresso Temático de Dinâmica e Controle da Sociedade Brasileira de Matemática Aplicada e Computacional, DINCON 2002, editado por J. M. Balthazar, V. A. Oliveira, G. N. Silva e J. M. Rosário Rio Claro, Brazil, 2002.

11) Rosário, J. M.; Oliveira, C. de; Sá, C. E. A.; Lima, C. R. E. Proposal of Methodology for the Modeling and Control of Manipulators. Journal of the Brazilian Society of Mechanical Sciences, Vol. XXIV, no. 3, 2002.

12) Rosário, J. M.;Lima, C. R. E.; Dumur D. Reconfigurable Architecture Proposal to Application on Mobile Embedded Systems Prototypes. Aceito para apresentação no X DINAME em Ubatuba, Brazil, em março de 2003.