UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES NEBULOSOS SINTONIZADOS POR ALGORITMOS GENÉTICOS

Davi Nunes Oliveira

Fortaleza
2010

O46i Oliveira, Davi Nunes
Implementação em FPGA de controladores nebulosos sintonizados por algoritmos genéticos / Davi Nunes Oliveira, 2010.
88 f.; il.; enc.

Orientador: Prof. Dr. Arthur Plínio de Souza Braga
Co-orientador: Prof. Dr. Otacílio da Mota Almeida
Área de concentração: Eletrônica de potência e Acionamentos
Dissertação (Mestrado) - Universidade Federal do Ceará, Centro de Tecnologia, Fortaleza, 2010.

1. Engenharia Elétrica. 2. Eletrônica de Potência. 3. Sistemas Difusos. 4. Conjuntos nebulosos. 5. Algoritmos Genéticos. 6. Controle de Processos. I. Braga, Arthur Plinio de Souza. (orient.). II. Almeida, Otacílio da Mota (co-orient.) III. Universidade Federal do Ceará – Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

CDD 621.3

DAVI NUNES OLIVEIRA

IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES NEBULOSOS SINTONIZADOS POR ALGORITMOS GENÉTICOS

Dissertação submetida à Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Arthur Plinio de Souza Braga.
Co-orientador: Prof. Dr. Otacílio da Mota Almeida.

Fortaleza
2010

DAVI NUNES OLIVEIRA

IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES NEBULOSOS SINTONIZADOS POR ALGORITMOS GENÉTICOS

Esta Dissertação foi julgada adequada para a obtenção de título de Mestre em Engenharia Elétrica, Área de Concentração em Eletrônica de Potência e Acionamentos, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Ceará.

Davi Nunes Oliveira

Orientador:

Prof. Arthur Plinio de Souza Braga, Dr.

Banca Examinadora:

Prof. Fábio Meneghetti Ugulino de Araújo, Dr.

Prof. Otacílio da Mota Almeida, Dr.

Prof. Bismark Claure Torrico, Dr.

Fortaleza, 09 de agosto de 2010

Aos meus pais Dary Alves Oliveira e Denise Nunes Oliveira.
Aos meus irmãos Daniel e Danilo.
Aos meus avós José Nunes de Melo (in memoriam) e Anna Celina Nunes de Melo,
Dion Amorim de Oliveira (in memoriam) e Alzira Alves de Oliveira.
A todos da minha família que não citei.
A todos que me incentivaram.
Eu dedico esse trabalho.

AGRADECIMENTOS

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) que contribuiu com apoio financeiro necessário à realização desse trabalho e desenvolvimento científico.

Ao professor Arthur Plinio de Souza Braga pela paciência e dedicação demonstradas durante a orientação deste trabalho, transmitindo sempre de forma competente e precisa os seus conhecimentos. Agradeço imensamente a confiança em mim depositada.

Quero também agradecer ao professor Otacílio da Mota Almeida por sua presença e ajuda constante através de seus conselhos, dispondo sempre da sua experiência e conhecimentos desde o início do curso de mestrado, durante o período das disciplinas, passando pela concepção deste projeto e o grande auxílio em sua execução.

Aos professores Fernando Antunes, José Carlos, Laurinda, Luiz Henrique, Ricardo Thé, Paulo Praça e a todos do Departamento de Engenharia Elétrica da UFC, Socorro, Mário e Rafael, responsáveis diretamente ou indiretamente pela minha formação na graduação e no programa de Mestrado.

Aos meus amigos e colegas de mestrado: Antonio Barbosa, David Erel, Fernando Sobreira, José Brito, Paulo Praça, Rafael, Ranoyca, Rodrigo Paulino e Wilkley Bezerra por todo apoio, incentivo, conhecimentos técnicos e companheirismo em todas as etapas vencidas.

A todos os meus familiares por sempre incentivarem minha formação profissional e pela grande ajuda nos momentos de maiores desafios em minha vida.

Oliveira, D. N. “IMPLEMENTAÇÃO EM FPGA DE CONTROLADORES NEBULOSOS SINTONIZADOS POR ALGORITMOS GENÉTICOS”, Universidade Federal do Ceará – UFC, 2010, 73p.

Esta dissertação propõe uma sistemática de projeto para controladores nebulosos em *hardware* FPGA, além de propor uma metodologia de sintonia desses controladores com a utilização de algoritmos genéticos. As etapas de simulação e resultados experimentais da implementação foram obtidos sobre o controle de uma planta não-linear: o pêndulo amortecido. O sistema é composto por uma placa de desenvolvimento de projetos em FPGA e placas de interface para acionamentos e aquisição de dados. Como características principais deste projeto podem ser citadas: o algoritmo de controle digital implementado com a utilização de uma linguagem de descrição de *hardware*; a arquitetura de processamento com paralelismo visando à melhoria do desempenho do sistema; e a aplicação de uma metodologia baseada em algoritmos genéticos como ferramenta de busca para sintonia do controlador nebuloso com função de desempenho baseada em desvios relativos à resposta desejada. A partir de um modelo do processo foram realizadas simulações e a sintonia de controladores nebulosos através da aplicação de algoritmos genéticos. Com os resultados obtidos em simulação, é realizada a implementação do controlador nebuloso em VHDL associada a uma interface para aquisição de dados dos ensaios. Os resultados experimentais validam a análise teórica e confirmam o desempenho do sistema, além de apresentarem resultados satisfatórios para diversos pontos de operação para um modelo linear obtido a partir de um sistema não-linear.

Palavras-Chave: Sistema Nebuloso Híbrido, Controle nebuloso, FPGA, VHDL, Algoritmos Genéticos, Processos não-lineares.

Oliveira, D. N. “FPGA IMPLEMENTATION OF FUZZY CONTROLLERS TUNED BY GENETIC ALGORITHMS”, Universidade Federal do Ceará – UFC, 2010, 73p.

This master thesis proposes a systematic design for fuzzy controllers in FPGA hardware, and proposes a methodology to tune these controllers using genetic algorithms. The steps of simulation and experimental results of the implementation were obtained over the control of a nonlinear plant: the damped pendulum. The system consists of a FPGA project development board and interface cards for drives and data acquisition. As main features of this project can be cited: the digital control algorithm implemented by using a hardware description language, the parallel processing architecture to provide improved system performance, and implementing a methodology based on genetic algorithms as search tool to tune the fuzzy controller with function based performance deviations on the desired response. From a process model simulations were carried out and tuning fuzzy controllers through the application of genetic algorithms. With the results obtained in simulation is performed the implementation of the fuzzy controller in VHDL associated with an interface for data acquisition trials. Experimental results validate the theoretical analysis and confirm system performance, and offer satisfactory results for several operating points for a linear model obtained from a nonlinear system.

Keywords: Fuzzy Hybrid System, Fuzzy Control, FPGA, VHDL, Genetic Algorithms, Nonlinear Processes.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

SIMBOLOGIA

Simbologia	Significado	Unidade
A/D	Conversor digital analógico	-
D	Razão cíclica	-
E_{ss}	Erro absoluto acumulado	-
f_m	Freqüência da moduladora	Hz
n	Relação de transformação	-
R	Resistência	Ω
t	Tempo	s
T_s	Período de chaveamento	s
V_{cc}	Tensão de saída do retificador	V
V_{ref}	Tensão de referência	V
V_s	Fonte de entrada	V

ACRÔNIMOS E ABREVIATURAS

Simbologia	Significado
AG	Algoritmo Genético
ADC	*Analog-Digital Converter* (Conversor Analógico-Digital)
ASIC	*Application-specific integrated circuit* (CI para aplicação específica)
CA	Corrente Alternada
CAD	*Computer Aided Design* (Desenho Assistido por Computador)
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CC	Corrente contínua
CI	Circuito Integrado
CN	Controlador Nebuloso
CPLD	*Complex Programmable Logic Devices* (PLD Complexo)
DSP	*Digital Signal Processor* (Processador Digital de Sinais)
EDIF	*Electronic Design Interchange Format*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
EOC	*End-of-Conversion* (Fim do processo de conversão)
FIS	*Fuzzy Inference System* (Sistema de Inferência Nebuloso)
FLC	*Fuzzy Logic Controller* (Controlador nebuloso)
FLIPS	*Fuzzy Logic Inferences per Second*
FRPS	*Fuzzy Rules per Second*
GAL	*Generic PAL* (PAL Genérico)
GND	*Ground* (Potencial de zero Volts)
I/O	*Input/Output* (Entrada/Saída)
IAE	*Integral Absolute Error* (Integral do Erro Absoluto)
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
JTAG	*Joint Test Action Group*
NN	*Neural Network* (Rede Neural)
OTP	*One-Time Programming* (Programável apenas uma vez)
PAL	*Programmable Array Logic*
PID	Proporcional Integral Derivativo
PLA	*Programmable Logic Array*
PLD	*Programmable Logic Device*
PWM	*Pulse-Width Modulation* (Modulação por largura de pulso)
RMSE	*Root Mean-Square Error* (Erro Médio Quadrático)
RNA	Rede Neural Artificial
SISO	*Single Input Single Output*
SRAM	*Static Random Access Memory* (Memória Estática de Acesso Aleatório)
SPLD	*Simple Programmable Logic Devices* (PLD Simples)
VHDL	Linguagem descritiva de alto nível para circuitos integrados de alta velocidade
VHSIC	*Very High-Speed Integrated Circuit*
VLSI	*Very Large Scale Integration*
UFC	Universidade Federal do Ceará

SÍMBOLOS DE UNIDADES DE GRANDEZAS FÍSICAS

Símbolo	Significado
A	Ampère
Δ	Delta
Hz	Hertz
mm	Milímetro
Ω	Ohm
rad	Radiano
s	Segundo
V	Volt

CAPÍTULO 1

INTRODUÇÃO

Muitos especialistas em controle denominam controladores avançados quando se referem aos controladores adaptativos, não-lineares, preditivos e inteligentes [Åström e Wittenmark, 1995]. Um dos objetivos do controle avançado de processos é a redução de oscilações da produção. Estudos revelaram que 80% dos laços de controle em indústrias de transformação amplificavam, em vez de reduzir, as oscilações do processo. Em média, essas variabilidades representam perda de produção de 5%, o que, na maioria dos casos, justificaria o custeio de estudos de implementação de algoritmos de controle avançado [Fairbanks, 2007].

Controladores convencionais podem não ter um desempenho satisfatório no controle de processos que apresentam características não-lineares. As características aleatórias e não previsíveis de sistemas não-lineares dificulta a aplicação de técnicas de controle devido ao comportamento desequilibrado e aperiódico no domínio do tempo. Além disso, uma mudança no ponto de operação pode alterar a dinâmica do sistema, não sendo compensada por um controlador linear. Na teoria de controle pode-se encontrar diversas estratégias para tornar o controle de processos não-lineares mais eficiente. A utilização de sistemas inteligentes em controle tem despertado grande interesse nos últimos anos pela possibilidade de aplicação em sistemas com características complexas. Dentre as técnicas mais utilizadas estão a Lógica Nebulosa ("*fuzzy*") [Simões e Shaw, 2007] e as Redes Neurais Artificiais (RNA) [Haykin, 1998]. Apesar de estar se formando um mercado crescente para controladores avançados, poucos fabricantes oferecem esta arquitetura integrada para aplicação em sistemas produtivos [Chalhoub, 2006] [Ormondi and Rajapakse, 2006].

A aplicabilidade de lógica nebulosa e redes neurais em plataformas FPGA e DSP têm sido tema de diversos trabalhos técnicos e científicos nos últimos anos [Chalhoub, 2006] [Jung, 2007] [Patra, 2006].

Este trabalho terá como escopo o estudo, desenvolvimento e implementação de um controlador nebuloso em hardware *Field Programmable Gate Array* (FPGA), onde o ajuste

dos conjuntos nebulosos é tratado como um problema de busca no espaço de parâmetros que define as funções de pertinência do controlador nebuloso. Para a resolução do problema de busca são aplicados algoritmos genéticos para evolução do desempenho de um conjunto de controladores nebulosos criados inicialmente. A evolução representa a busca de uma solução global, ou seja, a representação do controlador com o melhor desempenho para o sistema. As implementações apresentadas são aplicadas em plataforma FPGA. Como resultados do trabalho, apresenta-se o desenvolvimento de algoritmos de controle e técnicas de implementação. Como contribuições trazidas pelo trabalho pode-se citar a formalização de técnicas para a elaboração de controladores nebulosos em FPGA usando VHDL, a descrição de uma metodologia para evolução de controladores nebulosos por algoritmos genéticos, e o desenvolvimento, implementação e testes de controladores nebulosos em um sistema não-linear.

1.1. ORGANIZAÇÃO DOS CAPÍTULOS

Os capítulos estão organizado de forma a apresentar os fundamentos sobre conjuntos nebulosos, e lógica nebulosa no Capítulo 2, onde também são comentadas algumas implementações e expectativas para o uso de FPGAs com controladores nebulosos. No Capítulo 3 é feita uma explanação sobre dispositivos lógicos programáveis, classe à qual pertencem os FPGAs, as tecnologias empregadas na sua fabricação e, é feita uma explanação sobre as etapas de projeto que envolvem dispositivos FPGA. O Capítulo 4 contém o embasamento teórico sobre algoritmos genéticos e os operadores mais utilizados, além da estrutura do controlador nebuloso implementado em *Very High Speed Integrated Circuit Hardware Description Language* (VHDL). No Capítulo 5 são apresentados os resultados experimentais. A avaliação do sistema de inferência nebuloso implementado no FPGA tendo como referência um sistema de inferência implementado em Matlab®. Também são abordadas as estratégias de codificação das funções de pertinência para operação do algoritmo genético para evolução dos controladores. Resultados da identificação da planta, simulação dos controladores nebulosos e evolução por algoritmos genéticos são comparados para avaliação da melhoria incorporada ao sistema. No Capítulo 6 aborda as conclusões do trabalho, sugestões e perspectivas para trabalhos futuros.

1.2. PRODUÇÃO GERADA NA PESQUISA

Durante o curso do desenvolvimento da pesquisa com foco no tema desta dissertação foram submetidos e aprovados os seguintes artigos:

- D. N. Oliveira; A. P. S. Braga; O. M. Almeida. "*Design and Implementation of a Fuzzy Logic Controller on an FPGA using VHDL*". 29th North American Fuzzy Information Processing Society Annual Conference, Toronto, Canada, 2010.
- D. N. Oliveira; A. P. S. Braga; O. M. Almeida. "*Plataforma de prototipação rápida de controladores PID e nebulosos em FPGAs*". XVIII Congresso Brasileiro de Automática, CBA2010, Bonito, MS, Brasil, 2010.
- D. N. Oliveira; A. P. S. Braga; O. M. Almeida. "*Fuzzy implementado em ladder com funções de pertinência descontínuas*". XVIII Congresso Brasileiro de Automática, CBA2010, Bonito, MS, Brasil, 2010.
- D. N. Oliveira; A. P. S. Braga; A. B. S. Junior; V. P. Pinto; O. M. Almeida. "*Sintonia Evolutiva de Controladores Nebulosos*". XIV Congreso Latinoamericano de Automática, Santiago, Chile, 2010.
- D. N. Oliveira; A. P. S. Braga; A. B. S. Junior; V. P. Pinto; O. M. Almeida. "*Evolutionary Tuning of Fuzzy Controllers*". IX Portuguese Conference on Automatic Control, CONTROLO2010, Coimbra, Portugal, 2010.

Outros artigos aprovados desenvolvidos ao longo do programa foram:

- F. R. P. Magalhães; R. S. T. Pontes; R. O. Sousa; D. N. Oliveira; F. E. O. Barrozo; V. P. B. Aguiar. "*Correias transportadoras: um estudo de eficiência energética para o acionamento a velocidade variável*". XVII Congresso Brasileiro de Automática, CBA2008, Juiz de Fora, MG, Brasil, 2008.
- F. F. L. Freitas; W. Correia; D. N. Oliveira; O. M. Almeida. "*Aplicação de Controlador Preditivo Baseado em Modelo com Restrições a um Compressor Industrial*". VIII Conferência Internacional de Aplicações Industriais, INDUSCON2008, Poços de Caldas, MG, Brasil, 2008.
- D. N. Oliveira; B. F. S. Sousa; A. P. S. Braga; F. A. T. F. Silva; A. S. Teixeira. "*Aplicação de Mapas de Kohonen em imagem de satélite do semi-árido e comparação com o método da máxima verossimilhança*". XIV Simpósio Brasileiro de Sensoriamento Remoto, SBSR, Natal, RN, Brasil, 2009.
- A. B. Moreira; R. S. T. Pontes; D. N. Oliveira; V. S. C. Teixeira; V. P. B. Aguiar. "*Eficiência Energética em Sistemas de Ventilação Axial*". 3º Congresso Brasileiro de Eficiência Energética, IIICBEE, Belém, PA, Brasil, 2009.

CAPÍTULO 2

CONTROLE NEBULOSO

Os princípios da lógica nebulosa (*fuzzy* ou difusa) foram introduzidos por Lofti A. Zadeh em 1965 [Zadeh, 1965] e os princípios do controle nebuloso foram introduzidos por Mamdani e Sugeno [Mamdani, 1974] [Sugeno, 1985] [Yager e Zadeh, 1992].

A lógica nebulosa pode ser vista como uma extensão da lógica clássica que oferece uma alternativa eficiente, e matematicamente formalizada, para a representação e manipulação de uma base de conhecimento que incorpora incertezas e imprecisão. Esse novo modelo viola as suposições da lógica Aristotélica, base do raciocínio lógico ocidental, que trata afirmativas como verdadeiras ou falsas, não podendo assumir condições parcialmente verdadeiras ou parcialmente falsas. Entre a certeza de ser e a certeza de não ser, existem infinitas incertezas inerentes à informação representada em linguagem natural que são tratadas adequadamente pela lógica nebulosa [Yager e Zadeh, 1992]. Esta capacidade de lidar com incertezas é particularmente interessante no controle de processos, quando há imprecisões na leitura de sensores e dificuldade em descrever precisamente a dinâmica de processos complexos.

A base para a lógica nebulosa está na teoria dos conjuntos nebulosos [Zadeh, 1965], que estende o conceito clássico de pertinência de um elemento a um conjunto para o de uma função de pertinência (*membership function*) que pode assumir valores no intervalo [0, 1]. Com isso, pode-se dar uma gradação que indica o quanto um elemento pertence ao conjunto. A partir das funções de pertinência, relações entre conjuntos nebulosos podem ser definidas para funcionar como operadores lógicos [Yager e Zadeh, 1992] – permitindo escrever regras, e criar mecanismos para inferência e composição de regras. Tais regras podem ser utilizadas como base de conhecimento que descreve o comportamento desejado para um controlador [Yager e Zadeh, 1992].

Este capítulo está organizado de forma a apresentar: os princípios dos conjuntos nebulosos e da lógica nebulosa que permitem construir uma base de conhecimento capaz de

lidar com incertezas (Seção 2.1), a forma como um sistema de inferência nebulosa pode ser interfaceado com processos reais (Seção 2.2), implementações em hardware de controladores nebulosos (Seção 2.3), e as perspectivas que se abrem com o uso de FPGAs para a implementação de controladores nebulosos (Seção 2.4).

2.1. CONJUNTOS NEBULOSOS E LÓGICA NEBULOSA

O agrupamento de elementos com uma ou mais características em comum é chamado de conjunto. Na teoria clássica de conjuntos um elemento pode pertencer ou não a um conjunto, tendo função característica do tipo $\mu_A(x) = 1$ se x pertence ao conjunto A, e $\mu_A(x) = 0$ se x não pertence ao conjunto A. Conjuntos nebulosos podem ser interpretados como uma extensão dos conjuntos clássicos, pois permitem pertinência parcial a determinado conjunto variando entre a pertinência total e a completa exclusão, ou seja, $\mu_A(x) = [0,1]$ [D'Amore, 1998] [Passino e Yurkovich, 1998] [Pedrycz e Gomide, 2007] [Simões e Shaw, 2007]. Quanto maior o valor da pertinência, ou grau de pertinência, de um elemento a um conjunto nebuloso, maior a compatibilidade do elemento com a classe descrita pelo conjunto nebuloso [D'Amore, 1998] [Yager e Zadeh, 1992].

Assim, um conjunto nebuloso *A* pode ser descrito sobre um conjunto suporte *X* como um conjunto de pares ordenados do tipo:

$$A = \{\mu_A(x) / x \mid x \in X \} \tag{2.1}$$

sendo: $\mu_A(x)$ a função de pertinência do elemento *x* em *A*, que define um mapeamento no intervalo fechado [0, 1] [Tanscheit, 1992]. A função de pertinência indica o quanto um elemento $x \in X$ pertence a um dado conjunto nebuloso *A*. Assim, definido o conjunto suporte *X*, um conjunto nebuloso *A* também pode ser descrito apenas por sua função de pertinência [Tanscheit, 1992].

A ideia principal dos conjuntos nebulosos pode ser melhor compreendida se for levado em consideração que existe uma gama de conjuntos onde há a dificuldade de se estabelecer um valor limiar que caracterize os elementos que pertencem ou não ao conjunto (exemplo: definir o conjunto dos homens altos, ou dos idosos). Com a teoria de conjuntos clássica não é possível formalizar a transição gradual de característica exemplificada. A Figura 2.1 descreve o conjunto de todos os números menores que 18, a margem sobre o ponto x = 18 é natural. Porém, essa naturalidade se perde quando consideramos a mesma figura como sendo a representação da temperatura mínima confortável para um ambiente de laboratório de

pesquisa. Intuitivamente, sabe-se que a sensação térmica não se altera abruptamente em um ponto, mas sim gradualmente – ocorrendo suavemente a transição confortável [Bandemer e Gottwald, 1996].

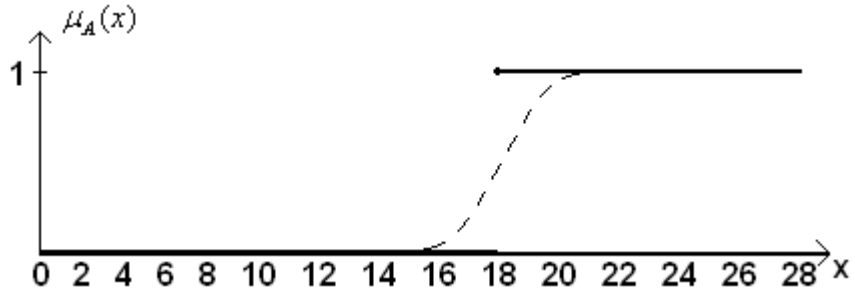


Figura 2.1 – A pertinência de um elemento: (a) na teoria clássica de conjuntos (em linha contínua), e (b) na teoria dos conjuntos nebulosos (em linha tracejada).

Portanto, uma maneira de visualizar os conjuntos nebulosos descrevendo suas características principais é observar os atributos das funções de pertinência. Conjuntos nebulosos raramente aparecem como entidades isoladas em processamento e modelagem nebulosa, sendo comum a formação de grupos de entidades com significados semânticos geralmente referidos a termos cognitivos [Pedrycz e Gomide, 2007].

Outro ponto importante é o tipo de função de pertinência utilizada, visto que na literatura estão dispostas diversas opções. Os tipos mais comuns são (Figura 2.2): triangular, trapezoidal e sigmoide.

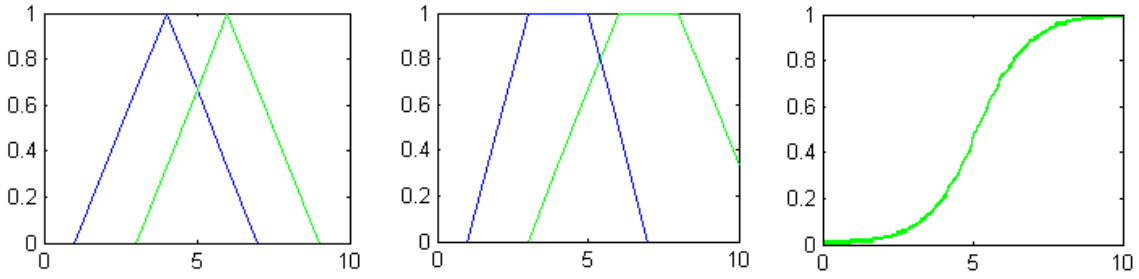


Figura 2.2 – Exemplos de Funções de pertinência: (a) triangular, (b) trapezoidal e (c) sigmoide.

2.1.1. Operações com Conjuntos Nebulosos

Os operadores básicos para conjuntos nebulosos descritos sobre um mesmo conjunto Universo foram propostos por Zadeh [Zadeh, 1965], e são uma extensão dos operadores para conjuntos da lógica clássica: união, interseção e complemento, também definidos na lógica clássica. Estas operações são descritas sobre as funções de pertinência dos conjuntos conforme segue abaixo [Bandemer, 1996]:

$$\mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)] = \mu_A(x) \vee \mu_B(x) \tag{2.2}$$

$$\mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)] = \mu_A(x) \wedge \mu_B(x) \tag{2.3}$$

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{2.4}$$

para $x \in \mathrm{X}$.

As operações *MAX* e *MIN* são definidas como:

$$\max(x, y) = \begin{cases} y \leftrightarrow x \leq y \\ x \leftrightarrow y \leq x \end{cases} \tag{2.5}$$

$$\min(x, y) = \begin{cases} x \leftrightarrow x \leq y \\ y \leftrightarrow y \leq x \end{cases} \tag{2.6}$$

As operações de união, interseção e complemento são ilustradas na Figura 2.3 sobre as funções de pertinência dos conjuntos nebulosos *A* e *B*.

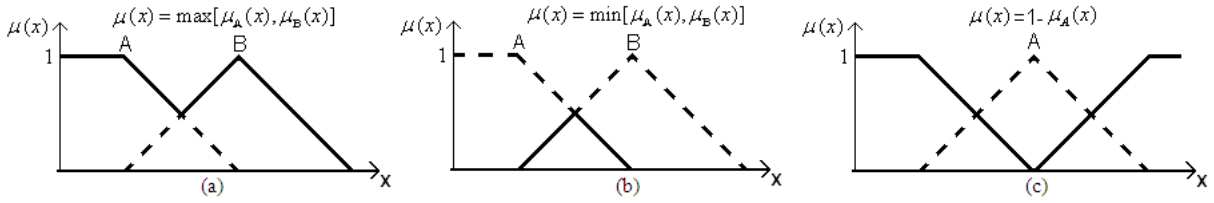


Figura 2.3 – Exemplo de operadores: (a) união, (b) interseção e (c) complemento.

2.1.2. Relações entre Conjuntos Nebulosos de Universos Distintos

As Equações (2.2) – (2.4) descrevem operações entre conjuntos nebulosos com um mesmo universo[1]. Porém, podem também ser realizadas operações entre conjuntos nebulosos de universos distintos – são as relações.

A relação entre conjuntos nebulosos de universos distintos é realizada sobre o produto cartesiano dos universos destes conjuntos. Se forem considerados dois conjuntos nebulosos *A* e *B* com universos *X* e *Y*, respectivamente, uma relação *R* entre conjuntos será descrita no produto cartesiano dos dois universos $X \times Y$. Uma função de pertinência $\mu_R(x, y)$ definirá a relação *R* a partir das funções de pertinência individuais $\mu_A(x)$ e $\mu_B(y)$ [Tanscheit, 1992]. Operações lógicas como 'OU' ($\vee$), 'E' ($\wedge$) e implicação ($\rightarrow$) podem ser

[1] Um conjunto universo *X* refere-se a todos os valores $x \in X$. O conjunto suporte de um conjunto nebuloso *A* refere-se ao conjunto de elementos no universo *X* para os quais $\mu_A(x)>0$.

implementadas como relações entre conjuntos nebulosos de universos distintos [Tanscheit, 1992] [Passino e Yurkovich, 1998] [Pedrycz e Gomide, 2007].

2.1.3. VARIÁVEIS LINGUÍSTICAS

A modelagem matemática clássica é baseada em descrições quantitativas dos processos, e faz uso de valores exatos ou acompanhados de suas faixas de erro. Mas existem diversas situações no cotidiano, onde processos são descritos muito mais qualitativamente, fazendo uso de adjetivos que incitam informações "vagas". Como exemplo do cotidiano, tem-se a descrição do clima como quente, frio, ou morno – é fornecida uma descrição qualitativa de fácil compreensão intuitiva, porém numericamente "vaga". O problema que surge é, como essas informações podem ser implementadas e/ou interpretadas por computadores. É necessária uma correlação entre as descrições qualitativas e valores a serem manipulados computacionalmente [Bandemer e Gottwald, 1996]. Os conjuntos nebulosos podem ser usados para realizar essa correlação.

Continuando com o exemplo da descrição do clima, os conceitos de quente, frio e morno podem ser representados por conjuntos nebulosos: todos tem um mesmo conjunto suporte X (uma faixa de valores de temperatura), porém as funções de pertinência representariam o quanto cada temperatura estaria associada ao conceito.

Seguindo esta ideia de representar informações "vagas" utilizando conjuntos nebulosos, diz-se que o clima é uma VARIÁVEL LINGUÍSTICA e as possíveis descrições do clima (quente, frio ou morno) são TERMOS.

Essa técnica de nomear os valores das variáveis nebulosas usando palavras do cotidiano foi explanada por [Zadeh,1975] que criou o conceito de variáveis linguísticas [Bandemer e Gottwald, 1996]. A aplicação mais interessante das variáveis linguísticas é de comporem uma estrutura intuitiva que permite descrever um processo. Uma variável linguística u no conjunto de suporte U pode assumir "valores" em um conjunto de termos, nomes ou rótulos, $T(u)$ – com cada valor sendo um conjunto nebuloso. Por exemplo, se u for velocidade, então seu conjunto de termos $T(u)$ poderia ser:

$$T(velocidade)=\{negativo, zero, positivo\} \tag{2.7}$$

Assim, sobre o conjunto de suporte U=[-3,3], os termos *negativo*, *zero* e *positivo* da variável linguística *velocidade* podem ser descritos pelas funções de pertinência apresentadas na Figura 2.4.

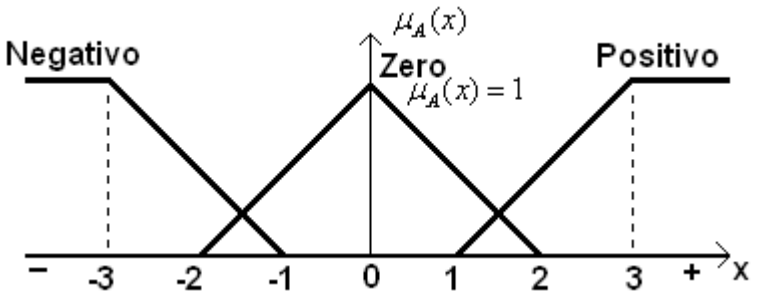


Figura 2.4 – Funções de pertinência dos termos da variável linguísticas velocidade.

Relações entre termos podem ser definidas considerando o mesmo universo de discurso ou universos diferentes. A negação NÃO e os conectivos E e OU podem ser definidos em termos das operações de complementação, interseção e união, respectivamente. Em geral, o conectivo E é usado com variáveis de universos de discurso diferentes. Por exemplo, o clima é quente e úmido. Se os termos pertencerem ao mesmo universo, a interpretação semântica invalida a premissa. Por exemplo, temperatura está alta e baixa, não faz sentido. Variáveis de um mesmo universo de discurso podem ser conectadas pelo conectivo E quando elas são negadas. Por exemplo, vapor é quente E NÃO frio [Tanscheit, 1992]. O conectivo OU pode conectar valores linguísticos de uma mesma variável dentro do mesmo universo de discurso, ou valores linguísticos em universos distintos.

Para a geração de uma quantidade maior de termos dentro de um universo de discurso pode-se utilizar pronomes ou adjetivos como, por exemplo, MUITO e PEQUENO.

2.1.4. IMPLICAÇÃO E LÓGICA NEBULOSA

As saídas das regras são obtidas a partir de uma relação entre um conjunto nebuloso ANTECEDENTE e um CONSEQUENTE. A partir das variáveis linguísticas e dos conectivos lógicos (relação entre conjuntos) é possível escrever sentenças que codifiquem informações intuitivas sobe um processo a partir dos conjuntos nebulosos. Essa é a base da Lógica Nebulosa [Zadeh, 1965] [Passino e Yurkovich, 1998] [Pedrycz e Gomide, 2007]. Normalmente, estas sentenças são REGRAS no formato SE-ENTÃO (implicação).

Estas relações R de implicação podem ser implementadas de diferentes formas [Passino e Yurkovich, 1998] [Pedrycz e Gomide, 2007] a partir das funções de pertinência dos conjuntos ANTECEDENTES, $\mu_A(x)$ e CONSEQUENTE, $\mu_C(y)$. Uma implementação da relação de implicação muito utilizada na literatura é a proposta por [Mamdani, 1974]:

$$\mu_R(x,y) = \mu_A(x) \wedge \mu_C(y) \tag{2.8}$$

A implicação de Mamdani, ou implicação MAX-MIN, gera um conjunto nebuloso R com função de pertinência descrita pela Equação 2.8 e universo de discurso dado pelo produto cartesiano $x \times y$ dos conjuntos suporte do ANTECEDENTE, x, e do CONSEQUENTE, y. Este conjunto nebuloso é a saída da regra.

As saídas de diversas regras podem ser combinadas através da operação MAX [Tanscheit, 1992] – é a Regra de Inferência Composicional.

2.1.5. REGRA DE INFERÊNCIA COMPOSICIONAL

Pela possibilidade de superposição entre os termos linguísticos e a correspondência parcial entre as regras elaboradas, comumente mais de uma regra de controle nebuloso pode ser válida em um mesmo instante. A metodologia utilizada para decidir qual a saída final gerada da composição das saídas de todas as regras é definida como um processo de resolução de conflitos [Yager e Zadeh, 1992]. Por exemplo, tendo as seguintes regras:

Regra 1: SE X é A1 E Y é B1 ENTÃO Z é C1 (2.9)

Regra 2: SE X é A2 E Y é B2 ENTÃO Z é C2 (2.10)

sendo X, Y e Z variáveis linguísticas, e A1, A2, B1, B2, C1 e C2 termos das variáveis linguísticas. Se tivermos x0 e y0 como entradas, e seus graus de pertinência aos termos X e Y representados por $\mu_{A_1}(x_0)$ e $\mu_{B_1}(y_0)$ para a Regra 1. E, similarmente $\mu_{A_2}(x_0)$ e $\mu_{B_2}(y_0)$ para a Regra 2. Então, o antecedente da Regra 1 é dado por $\alpha_1 = \mu_{A_1}(x_0) \wedge \mu_{B_1}(y_0)$, onde refere-se ao operador de conjunção ou MIN. Para a Regra 2, temos $\alpha_2 = \mu_{A_2}(x_0) \wedge \mu_{B_2}(y_0)$.

As saídas das Regras 1 e 2 são calculadas a partir da relação de implicação (Equação 2.11) entre antecedente e termo consequente de cada regra:

$$\mu_{C'_1}(\omega) = \alpha_1 \wedge \mu_{C'_1}(\omega) \text{ e } \mu_{C'_2}(\omega) = \alpha_2 \wedge \mu_{C'_2}(\omega) \quad (2.11)$$

Estes dois conjuntos nebulosos de saída, descritos pelas funções de pertinência da Equação 2.11, podem ser combinados para gerar um único conjunto nebuloso com função de pertinência $\mu_C(\omega) = \mu_{C'_1}(\omega) \vee \mu_{C'_2}(\omega)$, que corresponde ao operador MAX [Tanscheit, 1992] [Yager e Zadeh, 1992].

O mecanismo de inferência ocorre basicamente em duas etapas: identificação e dedução. Na etapa de identificação é analisada a relevância de cada regra para a atual situação das

entradas. Já na etapa de dedução é elaborada uma conclusão a partir da composição dos resultados de cada regra [Passino e Yurkovich, 1998].

A Figura 2.5 ilustra a implicação MAX-MIN [Passino e Yurkovich, 1998], onde as duas primeiras colunas (ENTRADA1 e ENTRADA2) estão relacionadas às funções de pertinência dos termos antecedentes de cada regra [Sivanandam *et al.*, 2007], e a terceira coluna (SAIDA) está relacionada com o termo consequente de cada regra. Os valores de pertinência são obtidos para cada valor de entrada e compostos seguindo a base de regras. O terceiro gráfico da terceira coluna representa a composição do sistema de inferência, onde o valor final depende dos valores de entrada e do método de desnebulização. O ponto indicado por CG na Figura 2.5 indica o valor escalar resultante da aplicação do método de desnebulização conhecido como centroide (será detalhado na Seção 2.2.3) sobre o conjunto nebuloso de saída.

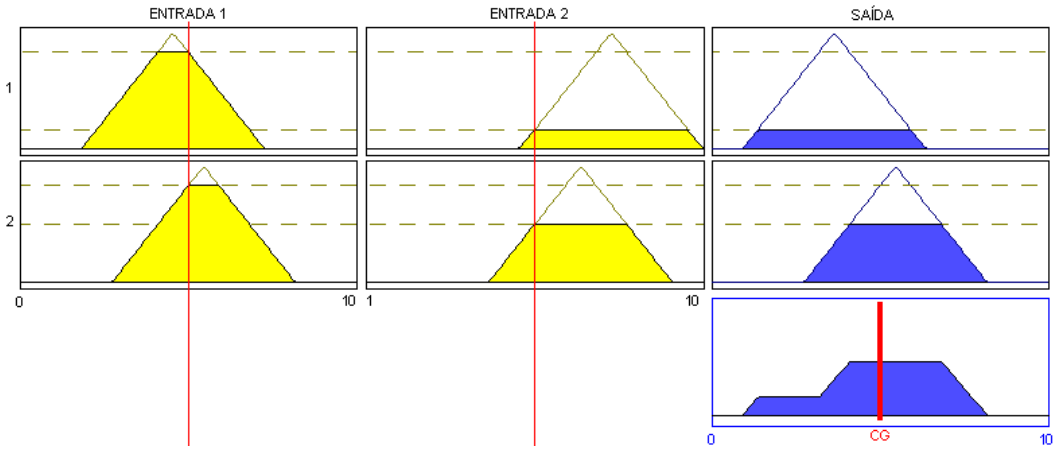


Figura 2.5 – Inferência nebulosa.

Um sistema de inferência como o descrito (Figura 2.5) pode ser utilizado em um controlador nebuloso [Passino e Yurkovich, 1998] [Pedrycz e Gomide, 2007] para inferir a saída de controle que deve ser gerada a partir de uma base de regras, e das entradas fornecidas a partir do processo que se deseja controlar.

2.2. CONTROLADORES NEBULOSOS

As pesquisas pioneiras sobre controle nebuloso foram motivadas e embasadas pelos trabalhos de Zadeh sobre a análise de sistemas baseada na teoria dos conjuntos nebulosos e aproximações linguísticas [Zadeh, 1965] [Zadeh, 1968]. Após a formalização do primeiro controlador nebuloso [Mamdani, 1974], houve um desenvolvimento do tema com diversas aplicações [Mamdani e Gaines, 1981] [Sugeno, 1985] [Ying *et al.*, 1990].

Os controladores nebulosos baseiam-se em lógica nebulosa, que é mais próxima da linguagem natural humana do que a lógica clássica, possibilitando captar a natureza, a aproximação inexata do mundo real. A essência do controlador nebuloso é um conjunto de regras linguísticas de controle relacionadas com conceitos de implicação nebulosa e inferência composicional [Lee, 1990].

Um típico controlador nebuloso está representado na Figura 2.6, e consiste de uma etapa de nebulização, um conjunto de regras (a base de conhecimento), um mecanismo de inferência, e uma etapa de desnebulização.

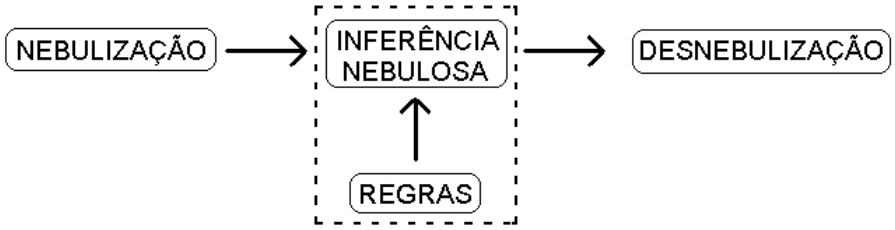


Figura 2.6 – Diagrama de blocos de um sistema nebuloso típico.

2.2.1. BLOCO DE NEBULIZAÇÃO

O valor de entrada, que pode ser a leitura de um sensor, deve ser convertido em termos das variáveis linguísticas aplicadas nas funções de pertinência de entrada [Yager e Zadeh, 1992]. A etapa de nebulização produz conjuntos nebulosos a partir dos valores de entrada do controlador. A Figura 2.7 exemplifica a associação de um grau de pertinência correspondente ao valor de leitura de um determinado elemento sensor [Yager e Zadeh, 1992]: por exemplo, caso se obtenha uma leitura do sensor igual a 4, será gerado um conjunto nebuloso com $\mu(x) = 0$ se $x \neq 4$ e $\mu(4) = 0{,}75$.

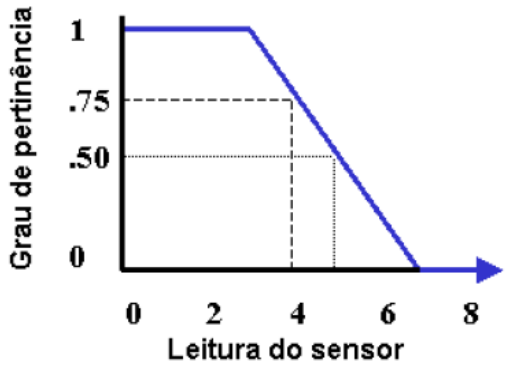


Figura 2.7 – Função de pertinência.

Na etapa de nebulização, o valor escalar correspondente à leitura dos sensores é convertido em um conjunto nebuloso cuja função de pertinência depende da função de pertinência do termo da variável linguística que se esteja considerando. A implementação da etapa de nebulização pode ser realizada, basicamente, de duas formas [D'Amore, 1998]: tabelas (*look-up table*) ou aproximação por trecho de retas, que pode ter a implementação de forma sequencial ou combinacional [D'Amore, 1998].

Na utilização das tabelas (Figura 2.8), as funções de pertinência são armazenadas em uma memória, permitindo sintetizar funções com qualquer perfil. Porém, considerando que uma função de pertinência possui valores diferentes de zero em apenas um pequeno intervalo do universo de discurso, grande parte da memória é desperdiçada. Como alternativa pode ser realizada a alocação de várias funções de pertinência em uma mesma memória, desde que não haja superposição no perfil das funções [D'Amore, 1998].

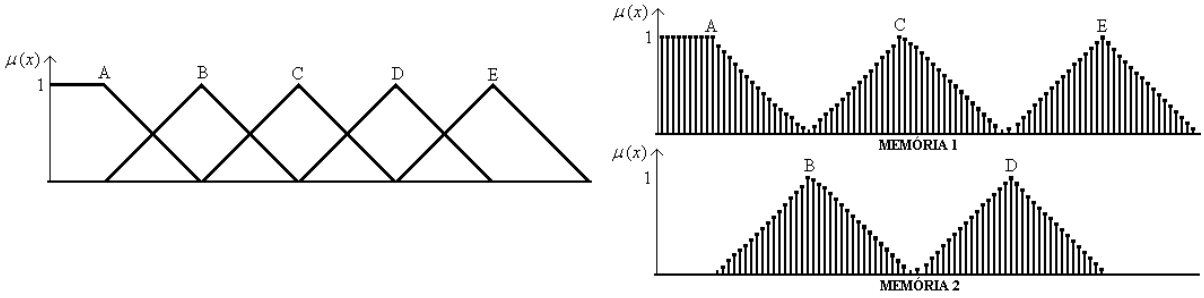


Figura 2.8 – Funções de pertinência armazenadas em tabelas.

Realizando a aproximação das funções de pertinência por trechos de reta, as funções são codificadas em parâmetros para reduzir a utilização da memória. Mas, existem restrições nas formas possíveis de função que podem ser geradas e, na maioria dos casos, o perfil das funções é descrito por trechos de retas [D'Amore, 1998].

A proposta de Knuth [Halgamuge *et al.*, 1994] para decodificação das funções de pertinência em trechos de reta é baseada na definição dos trechos de reta por três informações: o ponto inicial da reta no universo de discurso, o valor da função de pertinência nesse ponto e a inclinação da reta. Nakamura [Nakamura *et al.*, 1993] propôs um gerador combinacional onde a funções de pertinência é codificada numa palavra de 16 bits, possibilitando a formação de funções trapezoidais. Os principais parâmetros fornecidos para o gerador são: o centro da função, as inclinações laterais e o tamanho do topo do trapézio [D'Amore, 1998]. Os circuitos nebulizadores estão ligados à arquitetura do processador. As propostas do tipo tabela simplificam o projeto da máquina e permitem uma maior flexibilidade das formas de funções

de pertinência. No entanto, ocupam uma área de memória muito grande para sua implementação, mesmo considerando a possibilidade de várias funções serem inseridas numa mesma memória. Considerando que as funções de pertinência são geralmente triangulares, trapezoidais ou gaussianas aproximadas por trapézios, as implementações que codificam as funções de pertinência por trechos de retas permitem atender a maior parte das aplicações [D'Amore, 1998].

2.2.2. BLOCO DE INFERÊNCIA

A composição da base de conhecimento é realizada com a utilização dos termos linguísticos combinados das entradas relacionados com os termos linguísticos das saídas. O nível de granularidade aplicado às definições dos termos linguísticos é um parâmetro importante para a superfície de controle final [Yager e Zadeh, 1992].

Sugeno [Sugeno, 1985] sugere quatro métodos para realização da base de conhecimento:

- Conhecimento tácito
- Modelagem das ações de controle do operador
- Modelagem do processo
- Utilização de técnicas de auto-organização

Após a determinação dos valores das variáveis de controle (conjunto suporte), uma base de conhecimento deve ser descrita usando as variáveis de controle e criando relações entre as variáveis linguísticas de entrada e saída. Segundo [Yager, 1996], Sugeno sugere quatro métodos para criação da base de conhecimento. O primeiro é baseado na experiência e conhecimento sobre o processo, o segundo baseia-se na modelagem das ações de controle do operador. O terceiro refere-se à modelagem do processo e o quarto trata de técnicas de auto-organização. O primeiro método é o mais utilizado [Yager e Zadeh, 1992].

Um conceito que pode ser empregado para reduzir o processamento durante a inferência é a detecção de regras ativas, já que para um determinado conjunto de entrada apenas um subconjunto de regras que compõem a base de conhecimento contribui para o resultado final. As regras que compõem esse subconjunto são chamadas regras ativas. As regras restantes podem ser descartadas do processo de inferência devido ao grau nulo de pertinência das funções a elas associadas [D'Amore, 1998].

2.2.3. BLOCO DE DESNEBULIZAÇÃO

A etapa de desnebulização produz uma saída não-nebulosa (escalar) que melhor representa o conjunto nebuloso resultante do sistema de inferência. Diversos métodos de desnebulização podem ser encontradas na literatura, sendo os mais aplicados:

- **Método de desnebulização Tsukamoto** [Yager, 1992]: o valor escalar de saída do controlador nebuloso pode ser calculado por:

$$ZT = \frac{\sum_{i=1}^{n} \omega_i x_i}{\sum_{i=1}^{n} \omega_i} \tag{2.12}$$

  sendo n o número de regras com grau de ativação ω_i maior do que zero. E x_i é o i-ésimo valor do conjunto suporte de saída.

- **Método do centroide (*Center of Area*)** [Yager, 1992]: assume-se que a ação de controle resultante da combinação das funções de pertinência, corresponde ao centro de gravidade calculado sobre a distribuição da ação de controle nas funções de pertinência.

$$ZCOA = \frac{\sum_{j=1}^{q} z_j \mu c(z_j)}{\sum_{j=1}^{q} \mu c(z_j)} \tag{2.13}$$

  sendo q é o número de níveis da saída, z_j é o valor correspondente à saída no nível de entrada j e $\mu c(z_j)$ representa o valor de pertinência em C.

- **Método *Mean of Maximum*** [Yager, 1992]: gera um valor absoluto de controle através das médias dos valores suporte máximos das funções de pertinência. Para um universo discreto, pode ser calculado por:

$$ZMOM = \sum_{j=1}^{l} \frac{z_j}{l} \tag{2.14}$$

  sendo l o número de ocorrências do valor máximo de pertinência atingido por z.

O bloco de desnebulização, nas implementações em processadores dedicados, é o bloco que limita a velocidade de processamento e o tempo de resposta do controlador devido sua complexidade. Os processos de desnebulização empregados na maior parte das

implementações são: o método do centroide ou centro de gravidade, e o método da altura ou máximo das médias. Mesmo nos métodos de desnebulização que requerem menor carga de processamento são necessárias operações de multiplicação e divisão [D'Amore, 1998]. As unidades desnebulizadoras necessitam de uma série de interações para o cálculo do valor final. As etapas somatórias são realizadas por uma unidade somadora e um registrador para acúmulo do valor a cada ciclo de processamento. As unidades divisoras, normalmente, aguardam o término das operações de soma para iniciar o cálculo da divisão [D'Amore, 1998].

2.3. IMPLEMENTAÇÃO EM *HARDWARE* DE CONTROLADORES NEBULOSOS

A primeira máquina de controle nebuloso foi desenvolvida em 1984 por Togai e Watanabe [Togai e Watanabe, 1985] [Kandel e Langholdz, 1998]. Esta máquina era uma implementação digital com uma entrada e uma saída (*Single Input Single Output* – SISO), e apresentava processamento de desnebulização externo. Em 1986 foi apresentada por Yamakawa e Miki a primeira realização de um controlador nebuloso analógico com o processo de desnebulização interno [D'Amore, 1998] [Kandel e Langholdz, 1998].

Posteriormente, surgiram um grande número de propostas adotando implementações analógicas e digitais [Kandel e Langholdz, 1998]. A comparação de desempenho entre essas máquinas não é simples devido suas diferentes características, como: número de entradas, número de saídas, número máximo de regras permitido, tipos de funções de pertinência sintetizáveis, etc.

Nas implementações analógicas, a informação é representada por variáveis físicas como tensão, corrente, carga, etc. A vantagem de uma implementação analógica reside no fato de que muitas das funções a serem implementadas possuem uma adequação muito boa com circuitos clássicos de processamento analógico, e a interface entre as entradas e saídas é feita de modo direto. Além disso, a transferência de dados entre blocos pode ser feita por apenas uma linha de comunicação, reduzindo a área ocupada pelo circuito integrado com as interligações. Porém, essas implementações tem a informação representada por variáveis físicas diretamente, sofrendo uma dependência muito grande da aplicação e dificilmente permitem alteração do tipo de dado a ser tratado sem uma reavaliação do projeto [D'Amore, 1998].

Nas implementações digitais, a informação processada é representada na forma de dados binários que não sofre degeneração ao longo das etapas de comunicação entre blocos e processamento. A dependência da arquitetura com o processo a ser aplicado é mínima. As

implementações digitais podem ser sintetizadas a partir de linguagens de descrição de alto nível, tipo *Very High Speed Integrated Circuit Hardware Description Language* (VHDL) [D'Amore, 1998], e os circuitos sintetizados podem ser implementados em dispositivos lógicos programáveis (*Programmable Logic Device* - PLD) que permitem inúmeras reconfigurações do mesmo componente. Entretanto, a transmissão de dados em uma implementação digital é onerosa devido ao número de linhas necessárias para interligação, limitando assim, a flexibilidade no processamento paralelo. Quatro tipos de abordagem podem ser adotadas: microprocessadores de uso geral, microprocessadores dedicados, coprocessadores e máquinas de inferência nebulosa. Nas implementações utilizando microprocessadores de uso geral e dedicados, toda a estratégia de controle é descrita em um complexo programa que pode ser adaptado para novas aplicações, elevando assim o grau de flexibilidade. Os co-processadores e as máquinas de inferência nebulosa são circuitos dedicados ao processamento da lógica nebulosa [D'Amore, 1998].

Atualmente, as duas principais soluções de hardware na implementação de um controlador são DSPs e FPGAs. Portanto, de acordo com a natureza do algoritmo a ser implementado, o desenvolvedor tem que escolher entre essas duas possibilidades [Monmasson e Cirstea, 2007].

Com relação às restrições de tempo do algoritmo, que se baseiam principalmente nas interdependências entre os dados. Quanto maior for esta dependência, mais o algoritmo torna-se sequencial. A solução de software (DSPs) se adapta perfeitamente a este caso. Por outro lado, se o fluxo de dados revela muitas possibilidades de paralelismo (baixa dependência de dados e concorrência entre as operações), a solução de *hardware* (FPGAs) se torna mais interessante. A possibilidade de síntese de circuitos independentes em um projeto de FPGA torna a ferramenta poderosa pela capacidade de execução de diversos circuitos sequenciais simultaneamente. No entanto, restrições de tempo não são suficientes para caracterizar completamente um algoritmo. Sua complexidade também é um elemento-chave. A complexidade de um algoritmo é avaliada de duas maneiras: o número de operações e sua regularidade de ocorrência. Na verdade, um algoritmo que apresenta um número significativo de operações não é, necessariamente, complexo, se a maioria destas operações é idêntica [Monmasson e Cirstea, 2007].

No campo do controle digital de sistemas elétricos, os algoritmos são quase todos incluídos na área de intersecção dessas duas tecnologias. No entanto, em muitos casos, a aplicação em um DSP é preferível por razões históricas. As soluções de *software* são mais

antigas, e não assustam os desenvolvedores, pois são baseados em programação. No entanto, essa apreensão dos projetistas é cada vez menos fundamentada, tendo em vista a evolução das metodologias de projeto e ferramentas auxiliares [Monmasson e Cirstea, 2007].

Os benefícios do uso de FPGAs para controlar sistemas elétricos industriais, é baseada na habilidade dos FPGAs de executar quase que instantaneamente suas tarefas. Se o processo envolver conversão de sinais analógicos para digitais, a execução do algoritmo de controle e a leitura de dados pode haver um paralelismo entre os processos. Como consequência, a falta de rapidez do controlador deixa de ser um fator limitante do sistema [Monmasson e Cirstea, 2007].

Devido à capacidade do FPGA de transcrever para a arquitetura de hardware todos os potenciais paralelismos do algoritmo de controle, FPGAs têm uma fração do período de comutação para execução em tempo real de um algoritmo complexo inteiro. A consequência direta dessa extrema rapidez é o consumo de um grande número de recursos internos do chip, aumentando o custo. No entanto, usando técnicas de otimização, tais como *pipelining*, o projetista pode facilmente construir uma arquitetura equilibrada, que respeite o limite de recursos e rapidez de execução do algoritmo de controle [Monmasson e Cirstea, 2007].

Tais reações instantâneas fazem dos controladores baseados em FPGA muito próximos em seus comportamentos com os seus homólogos analógicos. Elas preservam as suas vantagens (não há atraso de cálculo, maior largura de banda), sem as suas desvantagens (imprecisão dos parâmetros, baixo nível de integração). Portanto, esta propriedade quase analógica poderia ser suficiente para promover esta tecnologia na implementação de mais e mais sistemas industriais de controle digital [Monmasson e Cirstea, 2007].

2.4. COMENTÁRIOS FINAIS

Controladores baseados em Lógica Nebulosa [Zadeh, 1965] vêm despertando um interesse cada vez maior pelo potencial de atuar satisfatoriamente sobre sistemas não-lineares a partir de uma Base de Conhecimento com Regras intuitivas. Dado este interesse, resta buscar formas eficientes de implementar e sintonizar estes controladores.

O uso de dispositivos eletrônicos de automação cada vez mais modernos têm facilitado implementações de algoritmos mais complexos de controle e inteligência artificial em *hardware*. Devido a isso, uma ampla gama de projetos de controladores inteligentes e complexos têm sido desenvolvida, inclusive para aplicações industriais. Um número significante desses projetos considera como elemento principal os *Field Programmable Gate*

Array (FPGAs) devido às características de prototipagem rápida e da flexibilidade oferecida por estes dispositivos [Floyd, 2006].

FPGAs constituem dispositivos adequados para a execução de Controladores Nebulosos (CNs), e um grande número de trabalhos têm sido publicados sobre a implementação de CNs nesta plataforma de *hardware* [Monmasson e Cirstea, 2007].

Estão presentes na literatura diversas formas de implementação de sistemas de inferência nebulosa e controladores nebulosos. Alguns trabalhos tratam da utilização de um ambiente de desenvolvimento de sistemas nebulosos, onde estão incluídas ferramentas para a especificação da estrutura do controlador e para a descrição dos sistemas, além de ferramentas de síntese em *software* e em *hardware* [Barriga *et al.*, 2006]. Também são descritos estilos de modelagem de lógica nebulosa utilizando VHDL para projeto VLSI [Rani *et al.*, 2005].

A implementação em *hardware* de controladores nebulosos em FPGA é muito importante por causa do número crescente de aplicações que exigem paralelismo e de alta velocidade de processamento nebuloso [Rani *et al.*, 2005]. Ferramentas para a tradução da representação em alto nível de controladores nebulosos em descrições VHDL já estão disponíveis e são base para estudos de metodologias e técnicas de implementação [Lago *et al.*, 1998]. EDA VHDL Synopsys Simulator, FPGA Synopsys Compilator, F1.3 Xilinx's Project Manager, e Hardware VCC Corporation Object Technology são ferramentas EDA utilizadas a fim de implementar controladores nebulosos a partir do códigos VHDL criados automaticamente [Kim, 2000].

Este trabalho apresenta uma implementação própria de um Controlador Nebuloso [Haykin, 1998] em um FPGA, sintonizado utilizando Algoritmos Genéticos [Mitchell, 1999].

CAPÍTULO 3

DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

Este capítulo traz uma explanação a respeito de dispositivos lógicos programáveis [Tocci e Widmer, 2003] e suas características, como também aborda o campo dos FPGAs (*Field Programmable Gate Array*) [Floyd, 2006] que é a tecnologia do componente empregado para a realização deste trabalho.

Os dispositivos de lógica programável (*Programmable Logic Devices* - PLDs) são uma categoria de dispositivos digitais de complexidades diferentes disponíveis no mercado que permitem ao usuário especificar a operação lógica desejada por meio de programação através de ferramentas de desenvolvimento, geralmente, fornecidos pelos próprios fabricantes [Barr, 1999] [Tocci e Widmer, 2003].

No projeto de circuitos lógicos, é comum a prática de se identificar as entradas e saídas, e descrever a tabela-verdade de forma a conter todas as combinações possíveis de entrada e os estados requeridos para as saídas em função de cada condição de entrada. Como alternativa, uma expressão Booleana pode descrever a operação do circuito. A partir desse ponto, o projetista deve encontrar a relação algébrica simplificada e selecionar os Circuitos Integrados (CIs) a serem interligados de forma a implementar o circuito [Tocci e Widmer, 2003]. Para evitar o consumo demasiado de tempo, reduzir a propensão a erros e eliminar etapas entediantes do projeto, o uso de dispositivos de lógica programável permite que essas etapas sejam realizadas com o auxílio de um computador com *software* de desenvolvimento para PLD. O conceito que fundamenta o uso de dispositivos lógicos programáveis é o de se utilizar arranjos com um grande número de portas lógicas em um único CI, e controlar eletronicamente as conexões entre as essas portas [Tocci e Widmer, 2003]. A Figura 3.1 ilustra um exemplo simplificado de um dispositivo de lógica programável.

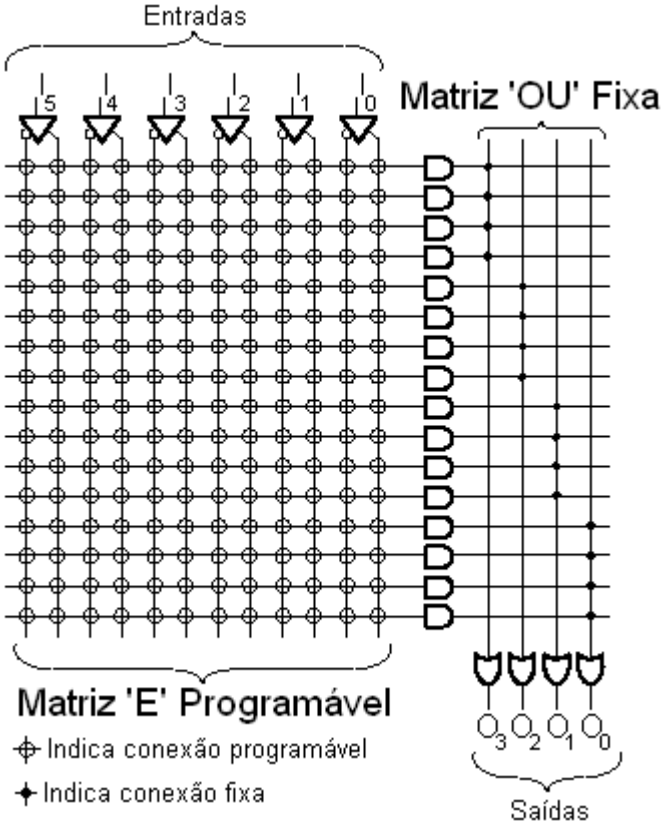


Figura 3.1 – Estrutura matricial de dispositivos PLD.

A matriz ‘E’ programável da Figura 3.1 permite indicar quais termos produto (saídas das portas ‘E’) estarão presentes nas expressões lógicas (na forma de soma-de-produtos) das saídas O_i. Os PLDs vêm ganhando mercado por permitirem a elaboração da mesma funcionalidade em um único CI, em vez de usar diversos *chips* lógicos individuais. O que significa menor espaço ocupado na placa, menor consumo de energia requerido, maior confiabilidade, menor complexidade de desenvolvimento e, geralmente, menor custo de fabricação [Tocci e Widmer, 2003].

Este Capítulo traz uma introdução sobre PLDs na Seção 3.1, assim como classificações de acordo com sua complexidade e tecnologia empregada em sua fabricação. A Seção 3.2 trata da linguagem de descrição de hardware e programação de PLDs.

3.1. PRINCÍPIOS DE PLDS

Programmable Logic Devices (PLDs) foram introduzidos na década de 1970. A ideia era construir circuitos lógicos combinacionais programáveis. No entanto, contrariamente aos microprocessadores, que podem executar um programa, mas possuem um *hardware* fixo, a programação de PLDs se destina a nível de *hardware*. Em outras palavras, o PLD é um chip

de uso geral, cujo hardware pode ser reconfigurado para aplicações específicas [Pedroni, 2004].

Os primeiros PLDs eram chamados PAL (*Programmable Array Logic*) ou PLA (*Programmable Logic Array*), dependendo do esquema de programação. Eles usavam apenas portas lógicas (exceto *flip-flops*), permitindo apenas a implementação de circuitos combinacionais. Para contornar esse problema, PLDs com registradores foram lançados pouco depois, e incluíam um *flip-flop* em cada saída do circuito. A partir de então, funções sequenciais simples poderiam ser implementadas [Pedroni, 2004].

No início dos anos 1980, circuitos lógicos adicionais foram acrescentados para cada saída PLD. A nova célula de produção, chamada macrocélula, continha, além do *flip-flop*, portas lógicas e multiplexadores. Além disso, a própria célula era programável, permitindo diferentes modos de operação. Além disso, foi incorporada a possibilidade de uma realimentação a partir do sinal de saída do circuito de volta para a matriz programável, que deu ao PLD maior flexibilidade. Esta nova estrutura de PLD foi chamada de PAL genérico (*Generic PAL* – GAL) [Pedroni, 2004]. Todos esses *chips* (PAL, PLA, PLD registrador, e GAL) hoje são referidos coletivamente como PLDs simples (SPLD). O GAL é o único ainda fabricado em encapsulamento independente [Pedroni, 2004].

Mais tarde, vários dispositivos GAL puderam ser fabricados em um mesmo *chip*, utilizando tecnologias de roteamento e purificação de silício mais sofisticadas, e várias características adicionais (como suporte a interface JTAG e vários padrões da lógica) foram incorporadas. Este novo componente ficou conhecido como PLD complexo (CPLD). CPLDs são atualmente muito populares devido à sua alta densidade, alto desempenho e baixo custo [Pedroni, 2004]. Finalmente, em meados de 1980, os *Field Programmable Gate Arrays* (FPGAs) foram introduzidos. FPGAs diferem dos CPLDs pela arquitetura, tecnologia, recursos internos e custos. Destinam-se principalmente na execução projetos de grande porte, e circuitos de alto desempenho [Pedroni, 2004]. Assim, para fins didáticos, pode-se classificar os PLDs segundo seu grau de integração, e suas arquiteturas, em: SPLD (*Simple Programmable Logic Device*), CPLD (*Complex Programmable Logic Devices*) e FPGA (*Field Programmable Gate Array*) [Barr, 1999]. Em meados da década de 90, surgiu uma tecnologia de controle analógica, os FPAAs (*Field Programmable Analog Array*). Porém esses dispositivos não acompanharam a evolução e expansão dos seus parentes digitais FPGAs [Barr, 1999]. A Figura 3.2 mostra árvore das arquiteturas dos sistemas digitais, e como os PLDs se posicionam entre os Sistemas Digitais.

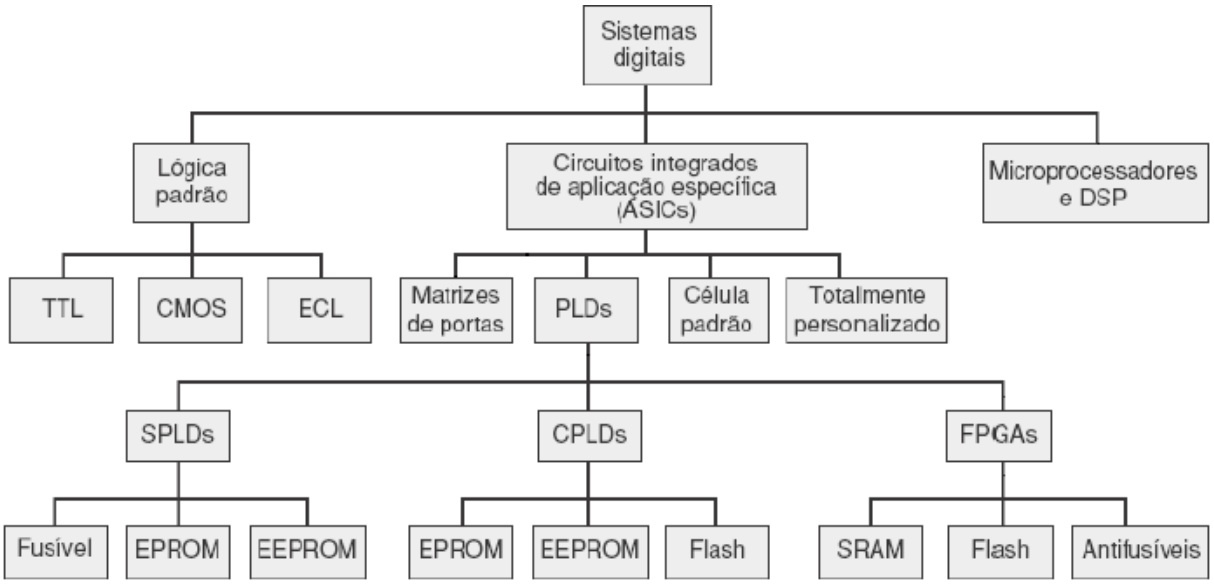


Figura 3.2 – Resumo das arquiteturas de sistemas digitais.

3.1.1. SPLD

Como mencionado anteriormente, PAL, PLA, GAL são conhecidos como PLDs simples (SPLD). Os dois tipos principais de dispositivos lógicos programáveis simples (SPLDs) são o PAL (lógica de arranjo programável) e o GAL (lógica de arranjo genérico). A estrutura básica de PALs e GALs é um arranjo ‘E’ programável e um arranjo ‘OU’ fixo, que é a realização da arquitetura básica de soma de produtos [Floyd, 2006].

Componentes PAL (Programmable Array Logic) foram introduzidos pela Monolithic Memories em meados dos anos 1970. Sua arquitetura básica consiste em um arranjo programável de portas ‘E’ que se conecta a um arranjo fixo de portas ‘OU’ [Barr, 1999]. Geralmente, os dispositivos PAL são implementados com a tecnologia antifusível e, portanto, são programáveis apenas uma vez (*One-Time Programming* – OTP) [Thomas, 2006].

A estrutura de um dispositivo PAL permite que qualquer lógica de soma de produtos com um número definido de variáveis seja implementada, ou seja, qualquer função lógica combinacional [Thomas, 2006].

O dispositivo GAL é essencialmente um dispositivo PAL que pode ser reprogramado. Os atuais dispositivos PAL e GAL tem muitas portas ‘E’ e ‘OU’. Uma porta ‘OU’ combinada com a sua lógica de saída associada é tipicamente denominada macrocélula e sua complexidade depende do dispositivo. Uma macrocélula pode ser configurada para lógica combinacional, lógica registrada ou uma combinação de ambas. A lógica registrada significa

que existe um flip-flop na macrocélula para prover uma função lógica sequencial [Thomas, 2006].

3.1.2. CPLDs

PLDs complexos, ou CPLDs, são dispositivos que combinam dispositivos de lógica de arranjo programável (*Programmable Array Logic* – PAL) no mesmo chip. Os próprios blocos lógicos têm conexões 'E' programáveis, 'OU' fixa, com menos termos produto do que a maioria dos dispositivos PAL. Quando um maior número de termos produto é necessário, uma matriz NAND expansora pode ser conectada como um termo entrada ou diversos blocos lógicos podem ser combinados para implementar a expressão. Os CPLDs também usam macrocélulas programáveis. O flip-flop usado para implementar o registrador na macrocélula pode ser configurado para operação D, JK, T ou SR. [Tocci e Widmer, 2003]

3.1.3. FPGA

A arquitetura de um FPGA é composta de três estruturas básicas: blocos lógicos, interconexões e blocos de I/O (*Input/Output* ou Entrada/Saída). Os FPGAs oferecem um grande número de blocos lógicos que contêm lógica combinacional programável e circuitos registradores independentes. Os blocos de entrada/saída (I/O) formam uma borda ao redor do dispositivo. Cada um destes blocos pode ser configurado como entrada fixa, saída fixa ou acesso bidirecional aos pinos. As saídas têm capacidade *tristate*, e registradores podem ser usados para armazenar dados de entrada ou de saída de I/O de uso geral disponíveis na parte externa do componente [Barr, 1999] [Tocci e Widmer, 2003].

Em um FPGA todos os blocos lógicos, e os blocos de entrada/saída, podem ser interconectados por programação para implementar virtualmente qualquer circuito lógico. As interconexões programáveis são implementadas através de caminhos que percorrem linhas e colunas nos canais entre os blocos lógicos [Tocci e Widmer, 2003]. Os FPGAs mais recentes são produzidos através de um processo de cobre de 65 nm. Sua densidade pode chegar a mais de 10 milhões de portas equivalentes por chip, com sistema de frequências de clock de mais de 500 MHz. Os dois principais fabricantes de FPGA são Altera® e Xilinx® [Monmasson e Cirstea, 2007].

FPGAs são disponibilizados em várias arquiteturas diferentes, que utilizam diversas tecnologias para armazenamento dos dados que definem as conexões programáveis, tais como SRAM (*Static Random Access Memory* ou Memória Estática de Acesso Aleatório), EEPROM

(*Electrically-Erasable Programmable Read-Only Memory*), flash EEPROM e antifusível [Tocci e Widmer, 2003]. A maioria das tecnologias empregadas em sua fabricação permite que o componente seja reprogramável. Isso permite que o circuito lógico seja alterado sem a necessidade de haver alterações no *hardware* adjacente ao FPGA [Barr, 1999].

Dada esta flexibilidade dos FPGAs, e sua velocidade, muitos projetos de controladores digitais vem sendo desenvolvidos nesta plataforma de hardware [Monmasson e Cirstea, 2007]. Estas características motivaram este trabalho a adotar os FPGAs na implementação de Controladores Nebulosos. A próxima seção apresenta como estes dispositivos lógicos são programados.

3.2. LINGUAGEM DE DESCRIÇÃO DE HARDWARE: PROGRAMANDO FPGAS

No início da década de 80, o Departamento de Defesa dos Estados Unidos desenvolveu uma linguagem de descrição de hardware para especificar e simular sistemas muito complexos utilizados na programação de seus circuitos integrados de maior velocidade (*Very High-Speed Integrated Circuit* - VHSIC). Essa linguagem envolveu um método padronizado pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) de descrição de circuitos lógicos para finalidades de projeto, simulação e documentação. Essa linguagem de descrição de hardware para VHSIC é conhecida como VHDL [Tocci e Widmer, 2003].

Linguagens de descrição de hardware, incluindo a VHDL, são direcionadas para o desenvolvimento de projetos com PLDs, especialmente para sistemas mais complexos usando CPLDs e FPGAs, segmento que desperta interesse da indústria [Tocci e Widmer, 2003]. A possibilidade de reprogramar um dispositivo após ele estar operando em um sistema abre um novo campo para o projeto de sistemas digitais. Atualizações e melhorias podem ser realizadas através de uma conexão do ambiente de desenvolvimento do fabricante para o sistema do usuário [Tocci e Widmer, 2003].

O código de descrição é compilado, e programado no dispositivo para ser executado. Descrever circuitos como um diagrama de blocos esquemáticos digitais também é possível, porém bem menos popular e mais complexo que utilizar ferramentas baseadas em linguagens descritivas (exemplo VHDL e Verilog) [Barr , 1999].

Uma característica divergente entre o desenvolvimento de *hardware* e de *software* é a lógica de raciocínio do desenvolvedor para abordar os problemas do projeto. Desenvolvedores de *software* tendem a seguir uma linha de raciocínio sequencial, mesmo quando no desenvolvimento de aplicações multitarefa. As linhas de código sempre são escritas para

serem executadas em uma ordem específica, pelo menos dentro de uma tarefa em particular. Mesmo com a utilização de um sistema operacional que cria a aparência de paralelismo, há apenas um núcleo de execução das operações lógicas. Durante o projeto de *hardware*, os desenvolvedores buscam a otimização e melhoria de desempenho empregando técnicas de paralelismo. Com isso, uma gama de sinais pode ser processada paralelamente, por estarem associadas a um fluxo de execução próprio (série de macro-células e interconexões) até o destino, representado por sinais de saída. Dessa forma, a descrição do *hardware* cria estruturas que podem ser executadas simultaneamente [Barr, 1999].

Tipicamente, a etapa inicial do projeto, onde são definidos os requisitos do projeto e elaboradas as lógicas, é compartilhada e seguida da etapa de síntese e simulação funcional. No processo de síntese, uma representação intermediária do projeto do *hardware*, chamada de *netlist*, é produzida. O *netlist* é um elemento independente do projeto e seu conteúdo não depende de um FPGA ou CPLD em particular, ele é armazenado geralmente em um formato padrão, conhecido como Formato Intermediário para Intercâmbio de Projeto ou *Electronic Design Interchange Format* (EDIF). No momento da simulação funcional, um simulador é utilizado para analisar o projeto a fim de avaliar o correto funcionamento dos circuitos lógicos descritos e a obtensão das respostas desejadas. Nessa etapa, o projetista pode certifica-se de que a lógica de funcionamento está correta antes de avançar no desenvolvimento [Barr, 1999].

A etapa seguinte do processo está relacionada com a alocação dos recursos do *chip* utilizado para o projeto em desenvolvimento. Também conhecido como *place & route* (posicionar e rotear), este passo requer a associação das estruturas lógicas descritas na *netlist* em macro-células, interconexões e pinos reais de entrada e saída. Este processo permite otimizações manuais ou automáticas das disposições e tem como resultado um conjunto de dados chamado *bitstream*, que deverão ser carregados no chip para a execução do projeto [Barr, 1999]. De acordo com as restrições do projeto, uma etapa de análise temporal se faz necessária para avaliação do desempenho do componente. Nela, os atrasos de transporte e efeitos dos caminhos e interconexões entre os blocos lógicos e blocos de I/O. Satisfeitas as restrições o projeto pode ser programado no *chip* [Barr, 1999].

O fluxograma de projeto usando softwares de desenvolvimento assistido por computador (*Computer Aided Design* – CAD) para implementação de circuitos lógicos em dispositivos lógicos programáveis é mostrado na Figura 3.3.

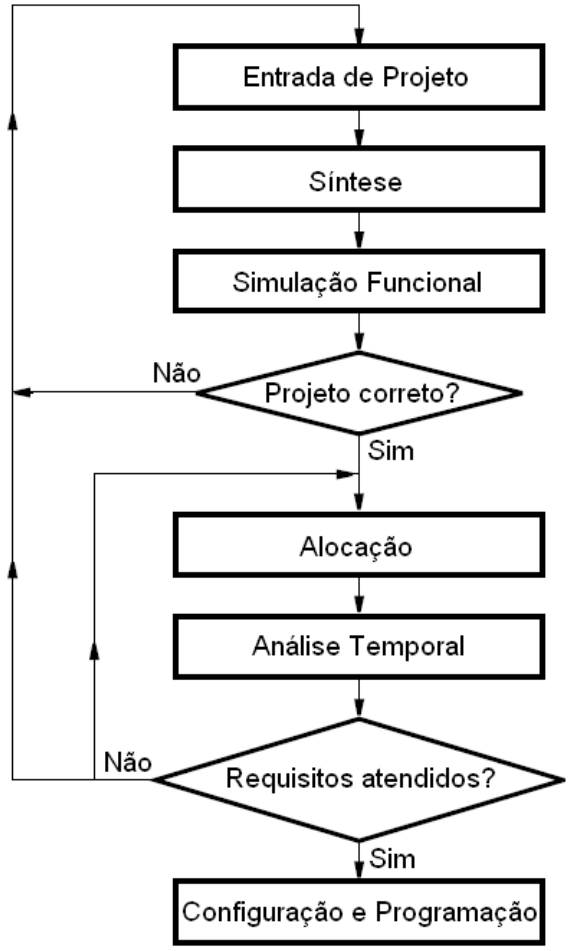


Figura 3.3 – Fluxograma de desenvolvimento em FPGA.

Os passos básicos de desenvolvimento são [ALTERA, 2009]:

- **Entrada de Projeto**: o circuito desejado é especificado usando uma linguagem de descrição de *hardware*, como Verilog ou VHDL, ou por meio de diagramas esquemáticos.
- **Síntese**: a ferramenta CAD sintetiza o circuito gerando um *netlist* que relaciona os elementos de lógica e as conexões entre eles necessários para realizar o circuito.
- **Simulação Funcional**: nesta etapa o circuito sintetizado é testado para verificar sua funcionalidade. A simulação não leva em conta as características temporais do circuito.
- **Alocação**: etapa onde é realizada alocação dos elementos lógicos definidos no *netlist* para os elementos lógicos do *chip* FPGA real. Também são traçadas as conexões necessárias entre os elementos lógicos.

- **Análise Temporal**: atrasos de propagação ao longo das conexões e elementos lógicos são analisados para fornecer uma indicação do desempenho do circuito.
- **Configuração e Programação**: o circuito projetado é implementado em um chip FPGA real pela programação a configuração dos elementos lógicos e conexões necessárias.

O software Quartus II fornece um ambiente de desenvolvimento completo para FPGAs de diversas famílias. O ambiente oferece uma interface gráfica com o usuário e um ambiente de desenvolvimento integrado que engloba todas as etapas de entrada do projeto para a programação de dispositivos FPGAs. Diferentes tipos de arquivos (diagrama de bloco, descrição em VHDL, descrição em Verilog, entre outros) podem ser combinados em uma estrutura hierárquica de projeto. A interface permite que o usuário inclua vários arquivos em um mesmo projeto, podendo haver transferência de informação entre eles. O software Quartus II possui ferramentas de edição e síntese de lógica, simulação funcional e temporal, análise temporal e de propagação de atrasos de transporte, localização automática de erros e, dispositivos para programação e verificação.

3.3. COMENTÁRIOS FINAIS

Os FPGAs fornecem prototipagem rápida e, em contraste com os processadores de uso geral, o FPGA representa, efetivamente, o circuito de lógica necessária para implementar o algoritmo desejado ao invés de uma sequencia de instruções sobre os recursos de hardware predefinidos. Assim, é possível alcançar um melhor desempenho que os processadores de uso geral.

Estes dispositivos lógicos são utilizados para implementar o Controlador Nebuloso descrito no próximo Capítulo.

CAPÍTULO 4

SINTONIA EVOLUTIVA DE CONTROLADORES NEBULOSOS

No contexto dos processos não-lineares complexos, onde os requisitos de projeto não podem ser satisfeitos utilizando métodos de controle convencional baseados em modelos lineares, muita atenção deve ser dada ao desenvolvimento de técnicas de controle não-linear [Yen, 1995], [Callai *et al*, 2007]. Os sistemas de controle devem garantir um adequado desempenho no intervalo das condições de operação do processo, estando sujeitos a diversas restrições e influências. Controladores Nebulosos (CN) vem se tornando uma estratégia cada vez mais atraente para o controle de processos não-lineares e com informações imprecisas [Passino, 1998], [Coelho *et al*, 2003], [Callai *et al*, 2007], despertando interesse cada vez maior pela capacidade de expressar com regras intuitivas simples estratégias de controle para sistemas complexos.

As leis de controle nebuloso (no caso de controladores linguísticos ou do tipo Mamdani [Passino, 1998]) são definidas por regras que associam variáveis linguísticas de entrada a variáveis linguísticas de saída. Os possíveis valores destas variáveis linguísticas são descritos por conjuntos nebulosos [Passino, 1998]. Porém, uma das dificuldades na utilização dos CNs é a definição destes conjuntos nebulosos utilizados para descrever as leis de controle [Callai *et al*, 2007] [Reznik, 1997]. Este capítulo traz uma proposta (implementada em simulação e experimentalmente no Capítulo 5) baseada em Algoritmos Genéticos (AG) [Silva, 2003] para a sintonia das funções de pertinência de Controladores Nebulosos com sistema de inferência do tipo Mamdani [Passino e Yurkovich, 1998].

Este trabalho trata o ajuste dos conjuntos nebulosos como um problema de busca no espaço dos parâmetros que descrevem as funções de pertinência do CN. Em se tratando de algoritmos de busca, os Algoritmos Genéticos (AGs) [Michalewicz, 1994] são algoritmos

probabilísticos inspirados no princípio Darwiniano de evolução das espécies e na teoria genética. AGs podem ser utilizados como um mecanismo de busca paralela e adaptativa baseado no princípio de sobrevivência de indivíduos mais aptos e na reprodução dos mesmos. Operadores genéticos, cruzamento e mutação, que satisfaçam certas restrições são propostos e testados neste trabalho.

Dado um sinal de referência (*set-point*), a avaliação dos CNs é feita a partir de medidas do tempo de subida, sobressinal (*overshoot)*, e erro absoluto acumulado ou integral do erro absoluto (IAE) da resposta obtida pelo sistema. Esta avaliação é utilizada como função de aptidão (*fitness*) para o AG. De acordo com os testes realizados, em um modelo de segunda ordem obtido para um ponto de operação do sistema não-linear considerado com os controladores sintonizados, há significativa melhoria do desempenho do CN ao longo do processo de evolução do algoritmo genético. Operadores de cruzamento e mutação próprios para a aplicação na sintonia das funções de pertinência das entradas e da saída do CN são descritos neste trabalho. Os resultados obtidos apresentam melhoria no desempenho dos controladores. A metodologia pode ser adaptada para outras plantas a partir da disponibilidade de um modelo do sistema, dado que o projeto aborda a evolução por AG em simulação.

Este capítulo está organizado de forma a apresentar: uma introdução teórica básica sobre Algoritmos Genéticos (Seção 4.1), seguida de uma descrição dos operadores genéticos implementados neste trabalho (Seção 4.2), uma descrição da implementação em FPGA de um Controlador Nebuloso (Seção 4.3) e comentários finais (Seção 4.4).

4.1. PANORAMA SOBRE A TEORIA DE ALGORITMOS GENÉTICOS

Algoritmos evolucionários são estruturas que utilizam modelos computacionais dos processos naturais de evolução como ferramenta para resolver problemas usando o conceito de simulação da evolução das espécies através de seleção, mutação e reprodução [Linden, 2006].

Algoritmos Genéticos (AGs) [Mitchell, 1999] foram propostos por John Holland na década de 1960 e desenvolvidos por Holland, seus alunos e colegas da Universidade de Michigan entre 1960 e 1970. Em contraste com as estratégias evolutivas ou programação evolutiva [Mitchell, 1999], o objetivo original de Holland não foi projetar algoritmos para resolver problemas específicos, mas sim o de estudar formalmente o fenômeno da adaptação como ocorre na natureza e desenvolver formas de incorporar aos sistemas de computador os mecanismos de adaptação natural [Beasley *et al.*, 1993].

AGs [Michalewicz, 1994] são um ramo dos algoritmos evolucionários e constituem uma ferramenta computacional inspirada na evolução, que busca de forma heurística uma solução potencial para um problema de otimização específico adotando uma codificação semelhante à de um cromossomo ou indivíduo para as possíveis soluções. São indicados para a solução de problemas de otimização global complexos que envolvem um grande número de variáveis e, consequentemente, espaços de soluções de dimensões elevadas. Entretanto, dependendo das características do problema de otimização, os operadores genéticos tem uma implementação diferenciada [Goldberg, 1989]. Operadores genéticos consistem em aproximações computacionais de fenômenos existentes na natureza, como a reprodução e a mutação genética [Linden, 2006].

Os Algoritmos Genéticos usam uma analogia direta do fenômeno de seleção natural, onde trabalha-se com uma população de indivíduos. Cada indivíduo representa uma possível solução para um dado problema. A cada indivíduo é atribuída uma pontuação de aptidão (*fitness*) de acordo com a qualidade da solução para o problema apresentado [Beasley *et al.*, 1993]. Por exemplo, a pontuação de aptidão para o projeto de uma ponte pode ser estabelecido pela razão força / peso de um determinado projeto. Na natureza, isso é equivalente à avaliação de indivíduos em competição por recursos. Os indivíduos mais aptos têm maiores oportunidades para reprodução. Com maiores chances de sobreviver, conseguem realizar um maior número de cruzamentos com outros indivíduos na população. Isso produz novos indivíduos que partilham algumas características herdadas de cada um dos pais. Os indivíduos menos aptos têm menos probabilidade de serem selecionados para reprodução, reduzindo a permanência de suas características na população e, assim, tendem a morrer [Beasley *et al.*, 1993]. Populações de indivíduos são criadas e submetidas aos operadores genéticos: seleção, recombinação (*crossover*) e mutação. Estes operadores utilizam uma caracterização da qualidade de cada indivíduo para avaliação e geração de um processo de evolução natural para a obtensão de um indivíduo ou população de melhor qualidade como solução para o problema [Linden, 2006].

Uma função de aptidão avalia cada solução, codificada como um cromossomo (indivíduo), dentro de um conjunto de possíveis soluções (população). Caso esta aptidão dos indivíduos da população atual atinja um critério pré-estabelecido, o algoritmo para. Caso esse critério não seja alcançado, a população é modificada através da aplicação de operadores genéticos sobre os indivíduos existentes. Os dois principais operadores genéticos são [Michalewicz, 1994]:

- **Cruzamento:** combina soluções já existentes, gerando novas soluções que guardam características das soluções passadas.
- **Mutação:** altera de forma aleatória as soluções presentes.

Os operadores cruzamento e mutação são aplicados aos indivíduos de forma que os indivíduos mais aptos tenham maior probabilidade de serem selecionados do que os menos aptos. A pequena probabilidade dos indivíduos menos aptos serem selecionados permite que nem todos sejam descartados da população, evitando assim, uma rápida convergência genética para um mesmo conjunto de características e permitindo um busca mais ampla pelo universo de soluções. A convergência genética se traduz em uma população com baixa diversidade genética e dificuldade de evolução [Linden, 2006].

Um terceiro operador genético comumente utilizado é o operador Elitismo. Sua principal característica é a de preservar as melhores soluções no grupo ao longo das gerações. A Figura 4.1 mostra o fluxograma típico de um AG.

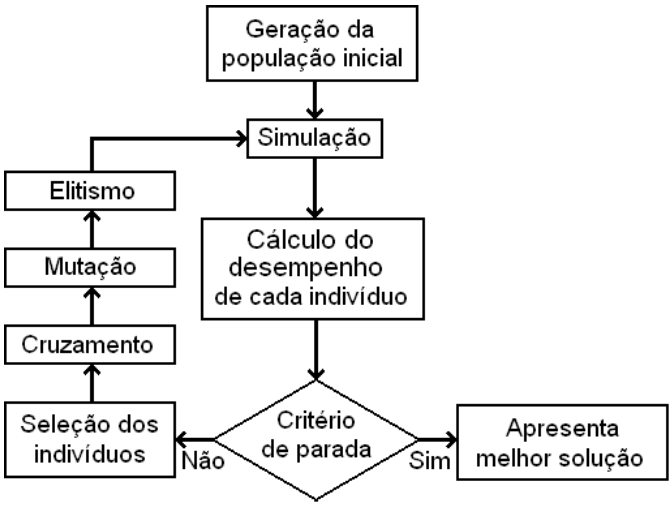


Figura 4.1 – Fluxograma de um AG típico.

AGs são técnicas heurísticas probabilísticas e não determinísticas. O que significa que a partir de uma mesma população inicial e mesmo conjunto de parâmetros, podem ser encontradas soluções diferentes a cada vez que o algoritmo é executado. Diferente dos esquemas enumerativos, os algoritmos genéticos não buscam em todos os pontos possíveis [Linden, 2006]. Também diferenciam-se dos esquemas aleatórios por constituir um sistema de busca que utiliza informações pertinentes ao problema e não opera com caminhadas aleatórias pelo espaço de soluções, mas sim direcionando sua busca através do mecanismo de seleção, equivalente ao processo de seleção natural [Linden, 2006].

As rotinas que compõem AGs são em geral simples e necessitam apenas de informações locais ao ponto avaliado, não necessitando de derivadas ou outras informações. Isto torna os AGs aplicáveis aos problemas reais que, em geral, incluem descontinuidades duras. Descontinuidades duras são situações onde os dados são discretos ou não possuem derivadas, comuns em problemas de alocações de recursos [Linden, 2006].

4.2. DESCRIÇÃO DA PROPOSTA IMPLEMENTADA

O objetivo desta seção é descrever o algoritmo proposto para o ajuste dos parâmetros das funções de pertinência do controlador nebuloso com a utilização de Algoritmos Genéticos, que tem como função principal a melhoria do desempenho do controlador.

Como proposta de melhoria do desempenho do controlador será utilizada uma técnica de ajuste dos parâmetros das funções de pertinência do controlador nebuloso a partir de AGs. Funções de pertinência dos tipos triangulares e trapezoidais são codificadas com base em quatro parâmetros. Esses parâmetros representam os pontos limites onde a função de pertinência possui seus valores extremos (0 ou 1). A Figura 4.2 ilustra a formação das funções de pertinência com exemplos para os casos triangular e trapezoidal.

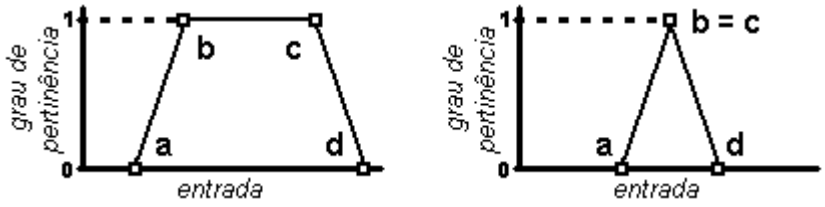


Figura 4.2 – Construção das funções de pertinência.

Cada variável associada às entradas e saídas do controlador tem sua lei de formação como descrita na Figura 4.3, onde o caractere ‘A’ indica um campo alfanumérico e o caractere ‘N’ indica um campo numérico.

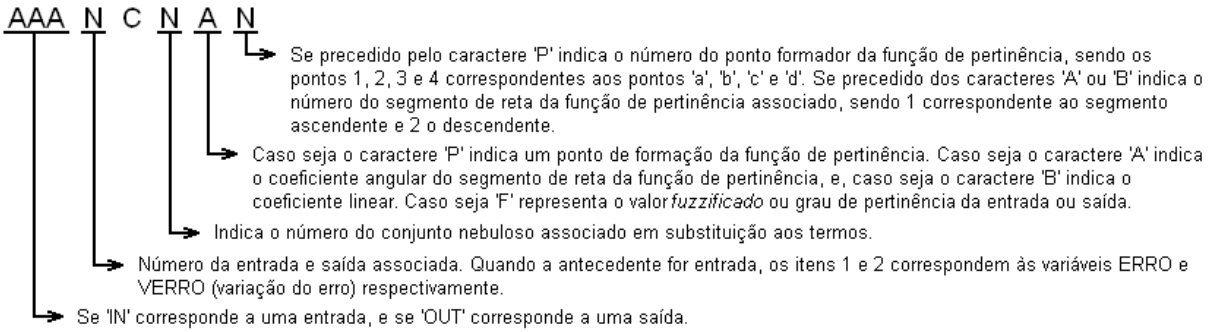


Figura 4.3 – Variáveis associadas às entradas e saídas.

A primeira etapa do algoritmo é responsável pela criação da população inicial de indivíduos, onde, cada indivíduo corresponde a um controlador. A população inicial é criada a

partir da geração aleatória de pontos (‘a’, ‘b’, ‘c’ e ‘d’) para as funções de pertinência, sendo observadas algumas restrições construtivas das funções de pertinência.

A Figura 4.4 ilustra a composição de um cromossomo originado a partir do algoritmo de formação da população inicial. Cada cromossomo possui a codificação para a formação das funções de pertinência dos termos das variáveis linguísticas usadas nas entradas e nas saídas do CN. Assim, o cromossomo está dividido em três partes: termos para a entrada ‘Erro’, termos para a entrada ‘ΔErro’ e termos para a saída. Cada uma destas partes está dividida em subpartes (termos). Na estrutura de cada termo está codificada a localização dos quatro pontos que definem as funções de pertinência dos termos. Maiores detalhes sobre a nomenclatura das variáveis utilizada são comentados na Seção 4.3.

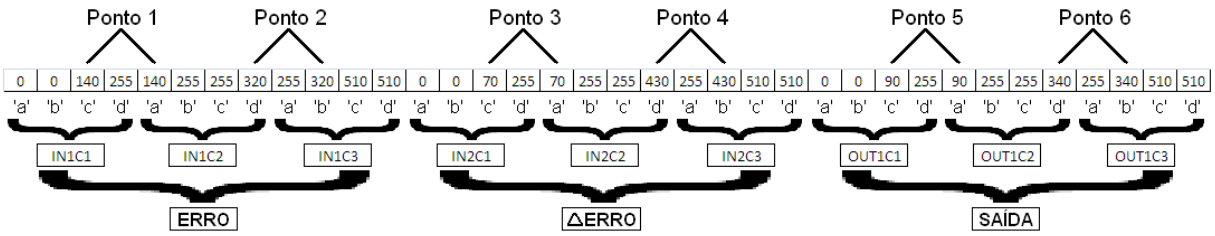


Figura 4.4 – Exemplo de cromossomo.

De acordo com a Figura 4.4 existem pontos de cada indivíduo da população inicial que apresentam valores equivalentes. O fato se dá devido a uma estratégia de programação da composição do cromossomo, que tende a seguir uma simetria em relação aos pontos definidos para as funções de pertinência de forma a manter o somatório dos valores de pertinência para cada valor do universo de entrada constante e unitário. Portanto, os pontos ‘c’ do primeiro termo e ‘a’ do segundo termo, assim como, os pontos ‘d’ do segundo termo e ‘b’ do terceiro termo de cada variável linguística têm o mesmo valor atribuído na criação do cromossomo, como serão apresentados na Seção 5.3. Este artifício utilizado também busca simplificar o trecho de código para geração dos indivíduos da população inicial, como também reduzir o tempo de processamento desta etapa. Apenas seis pontos para cada cromossomo são gerados aleatoriamente como representados os pontos de 1 à 6 na Figura 4.4.

Existem restrições sobre a criação dos cromossomos. Por exemplo, o horizonte de discurso é limitado e bem definido. Outra restrição existente entre os pontos que descrevem uma função de pertinência, como ilustrado na Figura 4.2, é considerar que 'a' ≤ 'b' ≤ 'c' ≤ 'd'. Para tanto, a escolha de um ponto aleatório pelo algoritmo deve respeitar os limites dos pontos anterior e sucessor em cada função de pertinência.

Outra etapa corresponde ao sistema de simulação de processo, onde cada cromossomo é carregado em uma estrutura do tipo *Fuzzy Inference System* (FIS) ou sistema de inferência nebulosa, e simulado. A partir do modelo estimado da planta (detalhado na Seção 5.1), e da definição de um sinal de referência para o sistema, pode ser iniciada a simulação onde se compara o estado da planta com o sinal de referência para se obter as entradas do controlador (erro e variação do erro). O valor da variável 'Erro' é dado pela Equação 4.1:

$$Erro = R - Y \tag{4.1}$$

onde R representa o sinal de referência (*set-point*) aplicado e Y representa a resposta da planta, para a aplicação é considerada a posição angular do eixo do pêndulo amortecido.

A saída do controlador obtida é aplicada à planta e um novo estado é determinado. A etapa de simulação pode ser executada com a utilização do sistema de inferência nebuloso desenvolvido com código próprio ou utilizando a estrutura existente na ferramenta *Fuzzy Logic Toolbox* do Matlab. A Figura 4.5 ilustra a interface da ferramenta *Fuzzy Logic Toolbox* do Matlab.

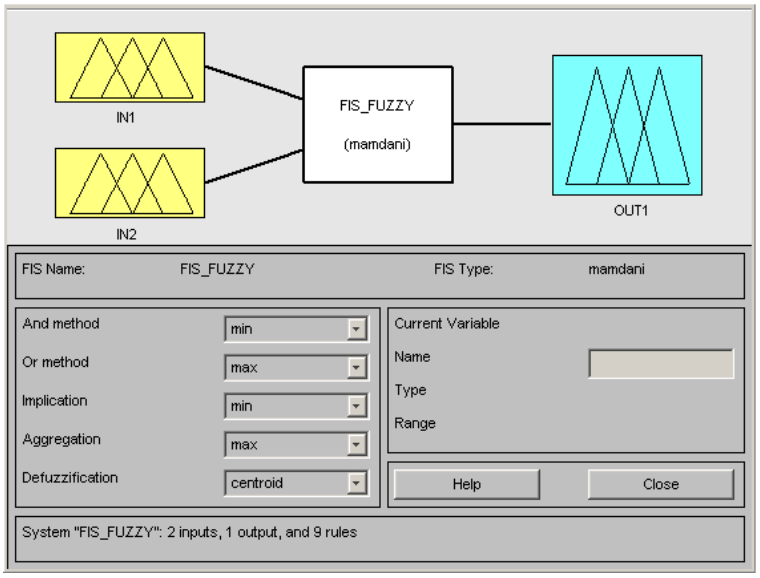


Figura 4.5 – Interface com a estrutura FIS.

Em seguida, é iniciada a simulação onde é obtido o comportamento da planta para cada controlador testado. A cada amostra são calculados os valores de erro e variação do erro, carregados como entrada na estrutura FIS (Figura 4.5) e obtido o valor da variação na saída do controlador aplicada à planta. Pontos de saturação foram inseridos na ação de controle e resposta da planta como fatores limitantes em conformidade com a estrutura de hardware descrita na Seção 4.3.

O bloco ‘Simulação’ representado na Figura 4.1 é detalhado na Figura 4.6, onde a partir de um sinal de referência, é feita a comparação de magnitude com a resposta atual da planta e obtido o valor do sinal de erro (conforme Equação 4.1). O sinal de erro representa uma das entradas do controlador. Uma técnica de derivação do sinal simples é aplicada onde o sinal do erro é comparado com um sinal de erro anterior sendo a diferença tomada como a variação do sinal de erro. Os sinais erro e variação do erro são as entradas do sistema de inferência do controlador nebuloso (detalhado na Figura 4.5). A saída do sistema de inferência é multiplicada por um fator de ganho cujo resultado é o valor de variação na ação de controle, que é acrescentado ao acumulador e aplicado à planta.

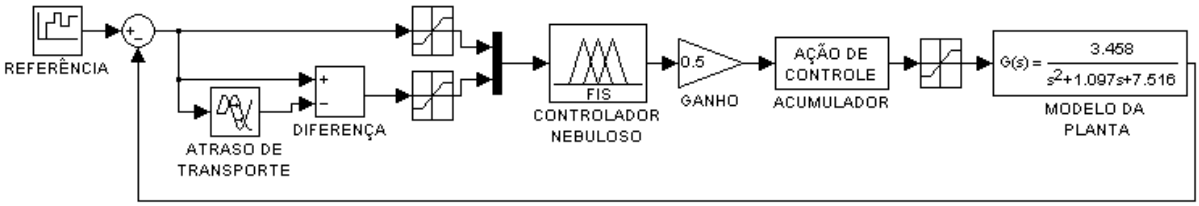


Figura 4.6 – Diagrama de blocos da etapa de simulação.

A partir da resposta do modelo ao indivíduo considerado são extraídas três medidas [Åström e Wittenmark, 1995]: tempo de subida, sobressinal, e do erro absoluto. Os fatores de ganho de 10 vezes para o sobressinal e de 100 vezes para o tempo de subida foram utilizados para adequar os valores das parcelas para uma mesma escala para que os fatores tenham influência aproximada sobre o valor final. Assim, o *fitness* é dado por:

$$fitness = sobressind \times 10 + tempo\_subida \times 100 + erro\_absoluto \qquad (4.2)$$

Na etapa seguinte, encontra-se concentrada a lógica de evolução por algoritmos genéticos. Inicialmente, há a construção de uma roleta para seleção entre os indivíduos mais aptos [Goldberg, 1989] [Michalewicz, 1994]: cada divisão da roleta está associada a um indivíduo, e tem uma área inversamente proporcional à sua aptidão calculada pela Equação 4.2. Como forma de manter uma resolução suficiente para diferenciar os melhores indivíduos dos piores é realizado escalonamento [Mitchell, 1999] da roleta de forma que a faixa da roleta ocupada por cada indivíduo seja definida pela divisão entre o melhor valor de aptidão, e o valor de aptidão do indivíduo analisado. Assim sendo, como o indivíduo mais apto é o de menor valor de sua função de aptidão dentro da população, o resultado da faixa da roleta será unitário para o melhor indivíduo e o restante estará no intervalo [0,1]. Dessa forma pode-se garantir uma resolução satisfatória para diferenciar os melhores dos piores indivíduos mesmo

em patamares muito distintos de desempenho. A roleta terá uma dimensão variável proporcional ao somatório das aptidões de todos os indivíduos da população.

Após a seleção dos pares de pais, os operadores genéticos são aplicados. Tomando-se cada par de pais, é realizado um sorteio para decidir se haverá ou não cruzamento de acordo com a probabilidade de ocorrência pré-determinada por um parâmetro inicial do algoritmo chamado taxa de cruzamento (*crossover*) inicialmente ajustado em 95% [Michalewicz, 1994]. O operador *crossover* funciona como na Figura 4.7: um ponto é gerado aleatoriamente, e indica o ponto de cruzamento entre os genes.

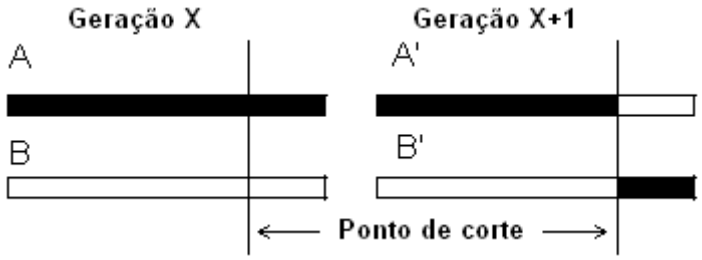


Figura 4.7 – Ocorrência de crossover.

O tamanho da população não é reduzido ao longo das gerações, sendo mantido constante. A formação da população da nova geração ocorre da seguinte maneira. Os pares são sorteados e, aleatoriamente, podem ocorrer cruzamentos ou não. Caso haja o cruzamento, os filhos são formados de acordo com as características cruzadas dos pais em torno do ponto de cruzamento e passam a substituir os pais na próxima geração. Caso não haja cruzamento os pais são levados à próxima geração, podendo ainda sofrer mutação. Assim, espera-se reduzir a pressão de seleção e evitar convergências prematuras.

Em seguida, o operador de mutação (Figura 4.8) é aplicado a cada gene com uma probabilidade dada pela taxa de mutação, ajustada em 5% [Mitchell, 1999].

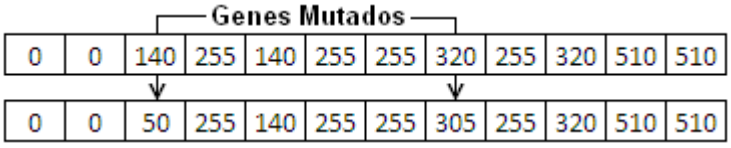


Figura 4.8 – Ocorrência de mutação.

A aplicação de AG também realiza o operador elitismo. Após a realização dos cruzamentos e mutações, ocorre a transposição do melhor indivíduo da geração atual para a seguinte. Essa transposição pode ocorrer de duas maneiras. Caso a população tenha um número par de indivíduos um dos filhos será sobreposto pelo melhor indivíduo da geração passada. Caso a população tenha um número ímpar de indivíduos o melhor indivíduo da

geração passada é adicionado à população de filhos. O número de indivíduos selecionados pelo elitismo é mantido unitário para não aumentar a pressão de seleção, ou seja, o risco de convergência prematura.

Formada a nova geração, esta é levada à etapa de simulação, onde são realizados ensaios a cerca da resposta, a aplicação de um degrau predefinido, dos novos indivíduos da população. O critério de parada é baseado no número de gerações definidos na simulação.

Para efeitos de visualização e avaliação do processo de evolução são exibidos gráficos correspondentes à resposta do controlador, o erro e a ação de controle obtido por meio de testes em cada indivíduo, o desempenho do melhor indivíduo da geração, o desempenho médio dos indivíduos e as formas das funções de pertinência dos indivíduos relevantes na população (resultados mostrados no Capítulo 5). Os gráficos são atualizados a cada geração, sendo possível acompanhar a evolução do desempenho dos controladores.

4.3. IMPLEMENTAÇÃO EM HARDWARE DE CONTROLADORES NEBULOSOS

O desenvolvimento, implementação e testes do algoritmo de controle nebuloso foram realizados em *Very High Speed Integrated Circuits Hardware Description Language* (VHDL). O algoritmo foi implementado em um kit de desenvolvimento Altera Cyclone-II que utiliza componentes da família EP2C20F484C7. A Figura 4.9 ilustra o kit de desenvolvimento utilizado.

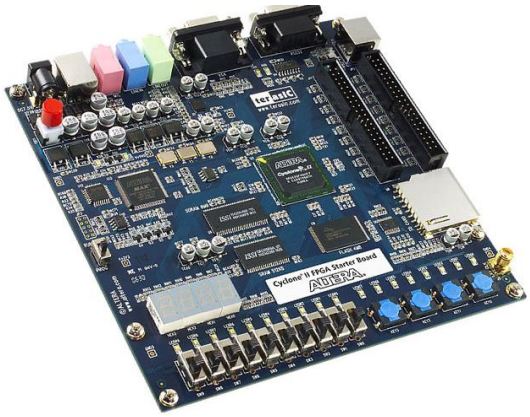


Figura 4.9 – Placa Cyclone II.

Cada valor de entrada e saída do controlador nebuloso é representado como um dado de 8 *bits*. Nove funções de pertinência foram consideradas para a entrada e saída de variáveis, sendo três para a entrada ERRO (IN1), três para a entrada VERRO (variação do erro) (IN2) e três para a saída (OUT1) que corresponde à variação da ação de controle.

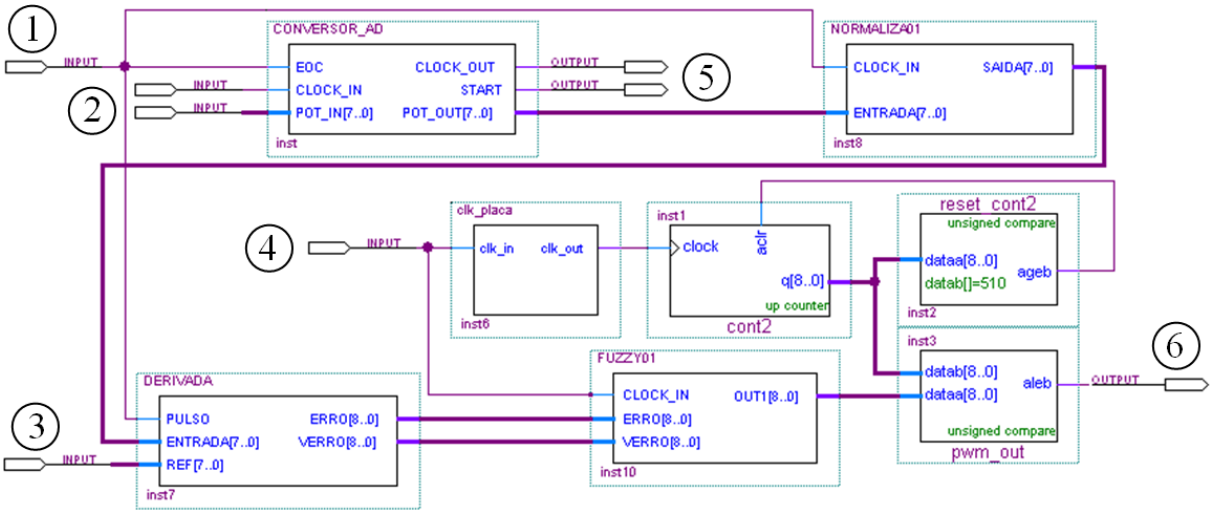


Figura 4.10 – Arquitetura do controlador nebuloso implementado.

Conforme a Figura 4.10, o sistema possui quatro entradas destacadas e identificadas por números de 1-4; e duas saídas identificadas pelos números 5-6:

1) Recebe o sinal do conversor analógico digital (ADC) de fim da conversão (EOC) que indica que os dados convertidos estão prontos para serem lidos,

2) Recebe paralelamente os 8 *bits* referentes à conversão do valor da variável a ser controlada para valores digitais,

3) Entrada de referência que recebe o valor desejado para posicionamento do sistema (*set-point*),

4) Entrada de sinal do oscilador da placa (*clock*) em 50MHz.

5) Saídas de controle e comando para o conversor ADC que representam a taxa de conversão e o comando de início de conversão,

6) Saída PWM (*Pulse Width Modulation*) associada à placa de potência para acionamento do atuador do processo a ser controlado.

A referência desejada pode ser indicada através do acionamento de 8 chaves localizadas na placa Cyclone II Development Kit (Figura 4.9), ou através de um bloco GERADOR implementado em VHDL e incorporado ao projeto (associado ao item 3 da Figura 4.10). A saída do sistema é um sinal PWM que possui *duty cycle* (ciclo de trabalho) correspondente ao valor de saída do controlador nebuloso, como está apresentado no item 6 da Figura 4.10. Este sinal aciona o atuador do processo.

A seguir é feita a descrição individual de cada bloco:

- **CONVERSOR_AD:** O bloco gera e fornece sinais de *clock* e *start* de conversão, e recebe o sinal de fim de conversão (EOC) do conversor analógico-digital (ADC) de 8 bits ADC0808 associados com a medida a ser controlada. O ADC compõe o circuito de conversão descrito no Apêndice A.
- **NORMALIZA01:** A função do bloco é reduzir o nível de ruído do sinal enviado pelo conversor ADC, calculando o valor médio entre as últimas cinco conversões realizadas dentro do período de amostragem.
- **DERIVADA:** Realiza a comparação entre o sinal de *setpoint* (referência) e a média das leituras da posição do sistema realizada pelo bloco NORMALIZA01 a fim de obter o valor de erro do sistema. O erro do sistema (descrito na Equação 4.1) é dado pela diferença entre o sinal de referência e o valor fornecido pelo conversor AD. Este bloco também possui uma estrutura de fila que armazena os valores dos erros anteriores para obter a variação do valor de erro em um horizonte de observação limitado e configurável. Os valores de saída do bloco constituem as entradas para o controlador nebuloso (erro e variação do erro). Os valores de erro e variação do erro são exibidos em displays de sete segmentos existentes da placa de desenvolvimento para permitir o acompanhamento visual da dinâmica do sistema.
- **CLK_PLACA:** Divisor de frequência que recebe o sinal de *clock* de um gerador externo existente na placa de desenvolvimento Cyclone II e efetua a divisão para condicionamento do sinal de *clock* para o gerador de PWM.
- **CONT2:** O bloco contador que compõe o gerador de PWM. Este bloco realiza a contagem de pulsos gerados pelo bloco CLK_PLACA. Cada incremento no bloco contador corresponde a um passo no sinal do PWM que pode assumir valores lógicos verdadeiros ou falsos para acionamento do motor.
- **RESET_CONT2:** Bloco comparador entre o valor instantâneo do contador e um valor constante predefinido correspondente ao comprimento máximo entre os ciclos de PWM. Quando os valores apresentam igualdade é gerado um pulso que reinicia a contagem do bloco CONT2.
- **PWM_OUT:** Este bloco, em conjunto com os blocos CLK_PLACA, CONT2 e RESET_CONT2, compõe o gerador de PWM. O bloco é encarregado de realizar a comparação de magnitude entre o valor instantâneo do contador e o valor de

saída do controlador nebuloso inserido no bloco FUZZY01. Se o valor do contador for menor ou igual ao valor de saída do controlador nebuloso a saída do bloco é posta em nível lógico verdadeiro, caso contrário é atribuído nível lógico falso. O sinal de saída desse bloco é direcionado para a placa de acionamento do atuador do sistema.

- **FUZZY01:** Este bloco possui as entradas de erro (ERRO), variação do erro (VERRO) e sinal de *clock* (CLOCK_IN) para sincronismo do controlador nebuloso, além da saída (OUT1) onde é apresentada a resposta do controlador ao estado da planta. A saída do bloco está associada diretamente com o bloco PWM_OUT para formação do ciclo de trabalho do PWM.

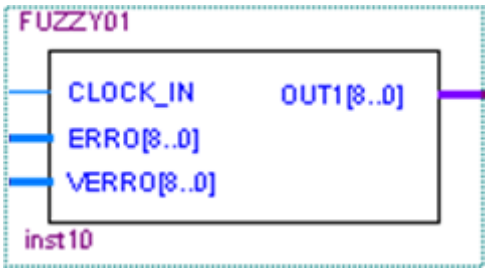


Figura 4.11 – Bloco encapsulado do controlador nebuloso.

Os sinais de erro (ERRO) e variação do erro (VERRO) são fornecidos através de blocos descritos anteriormente. O bloco FUZZY01 é projetado utilizando os conceitos nebulosos descritos na Seção 2.2. Conforme descrito na Figura 2.6, um controlador nebuloso típico possui os componentes descritos nas subseções seguintes.

As variáveis utilizadas no código VHDL possuem leis de formação (conforme descrito na Figura 4.3) para permitir que sejam flexibilizadas as quantidades de recursos necessários para a implementação. Assim, pode-se manter o padrão existente caso seja necessário a criação de mais regras, mais termos ou funções de pertinência específicas. A lei de formação empregada na criação da variável é definida de acordo com a associação. Cada variável pode estar associada às entradas e saídas, às regras, ou ao processo de desnebulização.

As variáveis associadas às regras possuem os nomes formados por RXF, onde X representa o número da regra. No caso das variáveis utilizadas no processo de desnebulização, os nomes são específicos de acordo com o método empregado. Sendo NUMZCOA a variável utilizada como acumulador do numerador do cálculo do centroide, DENZCOA a variável utilizada como acumulador de denominador e ZCOA a variável de armazenamento do valor

final de saída do sistema de inferência pelo método de desnebulização pelo cálculo do centroide.

4.3.1. PROCESSO DE NEBULIZAÇÃO

As entradas para o erro (ERRO) e variação do erro (VERRO) foram condicionadas com o mesmo número de funções de pertinência e mesmo universo de discurso (valores decimais de 0 à 510). Estes valores foram escolhidos para permitir a representação dos dados de 8 bits na faixa de valores positivos e negativos (-255 à 256). Os pontos das funções de pertinência são definidos em variáveis de 9-bits. Para cada função de pertinência há quatro variáveis ('a', 'b', 'c' e 'd') que determinam a forma da função (Figura 4.2). O grau de pertinência a um valor de entrada em relação a um conjunto nebuloso *A* é representado por μ_A e computado da seguinte forma:

$$\mu_A = \begin{cases} 0 & , \quad entrada \leq 'a' \\ a_1.entrada + b_1 & , 'a' < entrada < 'b' \\ 1 & , 'b' \leq entrada \leq 'c' \\ a_2.entrada + b_2 & , 'c' < entrada < 'd' \\ 0 & , \quad entrada \geq 'd' \end{cases} \tag{4.3}$$

onde a_1, a_2, b_1 e b_2 podem corresponder, por exemplo, às variáveis IN1C1A1, IN1C1A2, IN1C1B1 e IN1C1B2, respectivamente, que representam os coeficientes angulares e lineares dos segmentos de reta formadores da função de pertinência.

Para a inclinação positiva da definição do conjunto nebuloso que ocorre entre os pontos 'a' e 'b' da função de pertinência, é possível se obter a equação da reta que liga os dois pontos. A Figura 4.12 mostra como podem ser calculados os coeficientes angulares (IN1C1A1) e lineares (IN1C1B1) utilizando a linguagem VHDL.

```
IF (IN1C1P1 = IN1C1P2) THEN            --Cálculo do segmento ascendente
    IN1C1A1 := 1000;                   --entre os pontos 'a' e 'b'.
ELSE
    IN1C1A1 := 1000/(IN1C1P2-IN1C1P1); --Coeficiente angular
END IF;
IN1C1B1 := -(IN1C1A1*IN1C1P1);         --Coeficiente linear
```

Figura 4.12 – Cálculo segmento de reta ascendente das funções de pertinência.

Para a inclinação negativa da definição do conjunto nebuloso que ocorre entre os pontos 'c' e 'd' da função de pertinência, é possível se obter a equação da reta que liga os dois pontos.

A Figura 4.13 mostra como podem ser calculados os coeficientes angulares (IN1C1A2) e lineares (IN1C1B2) utilizando a linguagem VHDL.

```
IF (IN1C1P3 = IN1C1P4) THEN            --Cálculo do segmento descendente
    IN1C1A2 := 1000;                   --entre os pontos 'c' e 'd'.
ELSE
    IN1C1A2 := 1000/(IN1C1P3-IN1C1P4); --Coeficiente angular
END IF;
IN1C1B2 := -(IN1C1A2*IN1C1P4);         --Coeficiente linear
```

Figura 4.13 – Cálculo segmento de reta descendente das funções de pertinência.

Os valores obtidos para os coeficientes possuem um fator de multiplicação por 1000 para manter uma boa precisão usando variáveis inteiras. O cálculo dos coeficientes para todas as funções de pertinência são calculados de forma semelhante, inclusive as relacionadas à variável de saída. Pode-se observar que em ambos os casos o cálculo é composto da inversão da diferença absoluta entre os pontos envolvidos para o caso do coeficiente angular e, do produto do coeficiente angular pelo ponto que toca o eixo das abscissas em cada caso.

Após a obtenção dos coeficientes de todas as funções de pertinência pode-se efetuar o processo de nebulização propriamente dito. O valor lido em cada entrada é comparado com os valores dos pontos que compõem as funções de pertinência. Conforme o caso (exemplificado na Equação 4.3) os valores são atribuídos. A Figura 4.14 mostra o código para o processo de nebulização.

```
IF (IN1<=IN1C1P1) THEN                          --Atribuição de valor 0.
    IN1C1F := 0000;
ELSIF ((IN1>=IN1C1P1) AND (IN1<=IN1C1P2)) THEN  --Cálculo sobre o segmento
    IN1C1F := IN1C1A1*IN1+IN1C1B1;              --ascendente.
ELSIF ((IN1>=IN1C1P2) AND (IN1<=IN1C1P3)) THEN  --Atribuição de valor 1.
    IN1C1F := 1000;
ELSIF ((IN1>=IN1C1P3) AND (IN1<=IN1C1P4)) THEN  --Cálculo sobre o segmento
    IN1C1F := IN1C1A2*IN1+IN1C1B2;              --desscendente.
ELSIF (IN1>=IN1C1P4) THEN                       --Atribuição de valor 0.
    IN1C1F := 0000;
END IF;
```

Figura 4.14 – Cálculo do valor fuzzificado para um conjunto nebuloso.

4.3.2. AVALIAÇÃO DAS REGRAS

O módulo avaliador das regras é composto pela base de regras e o sistema de inferência nebuloso. Nele ocorre a fase de inferência descrita na Seção 2.2.2. Com base na fuzificação das variáveis de entrada erro (ERRO, IN1) e variação do erro (VERRO, IN2), um conjunto nebuloso de saída é calculado como mostrado na Figura 2.5. Considerando as regras IF-THEN como:

SE ERRO = A **E** VERRO = B **ENTÃO** OUTPUT = C (4.4)

A operação ‘E’ entre ERRO e VERRO para cada regra é realizada pelo seguinte algoritmo:

```
IF ((IN1C1F>0) AND (IN2C1F>0) AND (IN1C1F<=IN2C1F)) THEN
    R1F:=IN1C1F;
ELSIF ((IN1C1F>0) AND (IN2C1F>0) AND (IN2C1F<=IN1C1F)) THEN
    R1F:=IN2C1F;
ELSE
    R1F:=0;
END IF;
```

Figura 4.15 – Cálculo da relevância da regra em VHDL.

Cada regra tem sua relevância calculada como o código mostrado na Figura 4.15. O operador ‘E’ é realizado entre os termos antecedentes das regras. Estes valores são armazenados em variáveis para o processo de inferência. Os resultados destas etapas são os consequentes para cada regra, que serão agrupados de acordo com os termos consequentes das regras, e comparados com as funções de pertinência de saída associada para compor a saída em termos de conjuntos nebulosos.

A composição das saídas das regras ocorre com a aplicação do operador MAX para o conjunto de regras com o mesmo termo consequente de saída. A Tabela 4.1 mostra a base de regras utilizada, sendo: NE=Negativo, ZO=Zero e PO=Positivo.

Tabela 4.1 – Regras dos controladores nebulosos.

		Erro		
		NE	**ZO**	**PO**
ΔErro	**NE**	NE	NE	ZO
	ZO	NE	ZO	PO
	PO	ZO	PO	PO

A base de regras utilizada pode ser encontrada com frequência na literatura [Coelho *et al.*, 2003]. As regras utilizadas têm a forma **SE** <CONDIÇÃO1> **E** <CONDIÇÃO2> **ENTÃO** <CONCLUSÃO>, por exemplo:

SE <Erro é ZO> **E** <ΔErro é NE> **ENTÃO** <ZCOA é PO> (4.5)

Como a operação MAX não é nativa da linguagem VHDL, podem ser utilizadas comparações entre os valores de saída das regras para se obter o valor máximo. De acordo com a base de regras na Tabela 4.1, para o conjunto de saída C1 (NE) estão associadas as regras 1, 2 e 4 como mostra a Figura 4.16.

```
IF (R1F>R2F) THEN    --Correspondência ao conjunto O1
    OUT1C1:=R1F;       de Saída 1 (R1,R2,R4)
ELSE                 --Executa o MAX entre as regras
    OUT1C1:=R2F;
END IF;
IF (R4F>R1F) AND (R4F>R2F)THEN
    OUT1C1:=R4F;
END IF;
```

Figura 4.16 – Operador MAX entre regras associadas a um mesmo conjunto de saída.

4.3.3. PROCESSO DE DESNEBULIZAÇÃO

O processo de desnebulização (defuzificação) converte os valores nebulosos em uma saída real. Existem diferentes métodos para desnebulização (Seção 2.2.3). O cálculo do centroide (Equação 2.13) foi o método utilizado para a implementação. Um processo de três etapas é seguido no módulo de desnebulização. A primeira etapa consiste na determinação do ponto sobre a função de pertinência de saída analisado, que é associado a uma variável do tipo contador (CONT). Todos os pontos do horizonte de discurso de saída que estão associados com algum termo devem ser analisados individualmente. Na segunda etapa, mostrada na Figura 4.18, é aplicado o operador MIN entre o resultante das regras para determinado conjunto de saída (Figura 4.16) e o valor da função de pertinência de saída do conjunto associado às regras para o ponto analisado. O terceiro passo é converter o conjunto nebuloso resultante da composição das regras em uma saída (*crisp*). O algoritmo a seguir exemplifica o processo.

```
IF (CLOCK'EVENT AND CLOCK='1') THEN
    CONT:=CONT+1;
    OUT1C1F:=0;
    IF ((CONT>OUT1C1P1) AND (CONT<OUT1C1P2)) THEN
        OUT1C1F:=(OUT1C1A1*CONT)+OUT1C1B1;
    ELSIF ((CONT>=OUT1C1P2) AND (CONT<=OUT1C1P3)) THEN
        OUT1C1F:=1000;
    ELSIF ((CONT>OUT1C1P3) AND (CONT<OUT1C1P4)) THEN
        OUT1C1F:=(OUT1C1A2*CONT)+OUT1C1B2;
    END IF;
```

Figura 4.17 – Análise da superfície de saída.

Na Figura 4.17 é exibido o laço de varredura de toda a superfície que define as funções de pertinência de saída. A cada ciclo é analisado um ponto da superfície para comparação com os resultados do conjunto de regras. Esta comparação é realizada individualmente para cada termo e suas regras associadas. Em seguida, na Figura 4.18, a operação MIN é feita entre o resultante das regras que têm como termo consequente o mesmo conjunto de saída (Figura 4.16) e o valor do ponto da função de transferência para o termo correspondente analisado.

```
IF (OUT1C1F<OUT1C1) THEN
    OUT1C1:=OUT1C1F;
END IF;
```

Figura 4.18 – Operador MIN entre consequente das regras e conjunto de saída.

A Figura 4.19 mostra o cálculo para a saída, através da acumulação dos valores do numerador e denominador utilizando o método do centroide, e armazenando o resultado final na variável 'ZCOA' após a análise de toda a superfície de saída.

```
NUMZCOA:=NUMZCOA+(CONT*OUT1C1);
DENZCOA:=DENZCOA+OUT1C1;
IF (CONT>LIMITE) THEN
    CONT:=0;
    ZCOA:=NUMZCOA/DENZCOA;
    NUMZCOA:=0;
    DENZCOA:=0;
END IF;
END IF;
```

Figura 4.19 – Cálculo da centroide em VHDL.

4.4. COMENTÁRIOS FINAIS

Com base na teoria de algoritmos genéticos e com a descrição dos operadores utilizados, é formalizada a descrição da proposta de implementação e codificação que permite integrar a configuração do controlador nebuloso com o cromossomo. De forma que o cromossomo, como estrutura do algoritmo genético que possibilita a execução do algoritmo de evolução, incorpore os parâmetros dos controladores nebulosos, tornando possível a execução de ensaios com as duas estruturas.

Neste capítulo também foram explanados trechos do código do controlador nebuloso implementado em VHDL. Também foram apresentados os blocos auxiliares ao sistema de inferência nebulosa, que condicionam os sinais de entrada e possibilitam a operação do controlador.

Da observação do comportamento do sistema sob a ação do controlador serão levantados os critérios de desempenho (tempo de resposta, sobressinal e erro absoluto) que ajudarão na observação da evolução dos controladores com a aplicação das técnicas de algoritmos genéticos.

CAPÍTULO 5

EXPERIMENTOS E RESULTADOS

Os testes de desempenho dos controladores nebulosos sintonizados pela metodologia descrita são apresentados neste capítulo para demonstrar a precisão do sistema simulado e implementado no Capítulo 4. Nas simulações, o *Fuzzy Logic Toolbox* do software MATLAB™ é adotado como referência para avaliação do sistema de inferência nebulosa implementado no FPGA. Com relação ao desempenho de cada controlador testado em simulação, foram desenvolvidos algoritmos para obtenção dos valores de critérios de desempenho (Seção 4.2) para os controladores avaliados para evolução por algoritmos genéticos. Ao final dos testes o melhor controlador é aplicado na planta real para se avaliar o comportamento do sistema.

Identificação de sistemas dinâmicos é o campo de interesse em construção de modelos matemáticos de sistemas não-lineares a partir de dados experimentais medidos ou observados [Ljung, 1987]. Normalmente, uma estrutura de um determinado modelo linear ou não-linear que contém parâmetros desconhecidos é escolhida pelo usuário. Em geral, os parâmetros devem ser calculados de modo que os erros entre a resposta estimada (ou prevista) e as saídas reais do sistema sejam minimizados, a fim de captar a dinâmica do sistema o mais próximo possível. O modelo resultante pode ser usado como uma ferramenta para análise, simulação, e projeto de sistemas de controle.

Neste trabalho, é utilizada uma técnica de identificação de sistemas tipicamente aplicada para sistemas de segunda ordem pouco amortecidos [Aguirre, 2007] para se obter a função de *fitness* (Equação 4.2). A técnica consiste na estimação da frequência natural do sistema e do seu coeficiente de amortecimento.

Considera-se uma planta não-linear simples como caso de teste: o pêndulo amortecido [Oliveira *et al.*, 2010]. Apesar de a planta apresentar comportamento característico de sistema

não-linear, a identificação é realizada em torno de um ponto de operação onde considera-se como um sistema linear de segunda ordem.

Na sequência do capítulo é feita uma descrição da planta utilizada na Seção 5.1, um teste comparativo do sistema de inferência nebulosa é apresentado na Seção 5.2, simulações dos controladores são apresentadas na Seção 5.3, a Seção 5.4 traz dados de testes com a planta real e a Seção 5.5 traz os comentários finais do capítulo. Uma descrição da placa de conversão de sinal analógico-digital, condicionamento de sinal para os níveis de tensão da placa de desenvolvimento FPGA e, acionamento por PWM do motor propulsor é dada com maiores detalhes no Apêndice A. Um projeto desenvolvido para testes com implementação de controladores PID também foi desenvolvido e é descrita no Apêndice B.

5.1. PLANTA DE TESTE: PÊNDULO AMORTECIDO

A fim de avaliar e testar a metodologia proposta para o ajuste do controlador nebuloso, um processo não-linear é considerado: o pêndulo amortecido [Khalil, 1996]. A Figura 5.1 mostra o pêndulo.

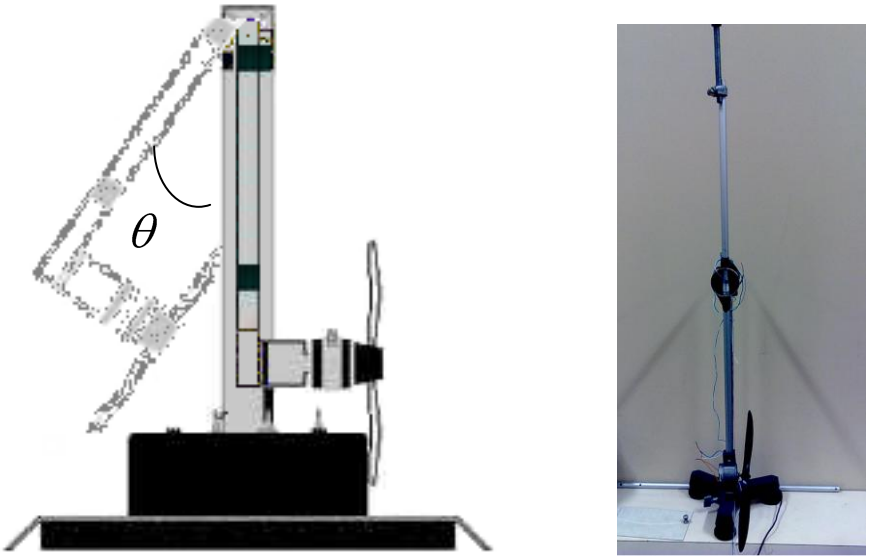


Figura 5.1 – Pêndulo amortecido.

Considerando θ o ângulo entre o eixo e haste vertical, o modelo de espaço de estado do pêndulo, tendo como variáveis de estado a posição (Equação 5.1) e a velocidade (Equação 5.2), e é dada por [Khalil, 1996]:

$$\dot{x}_1 = x_2 \tag{5.1}$$

$$\dot{x}_2 = -\frac{g}{l} sen x_1 - \frac{k}{m} x_2 + cV \tag{5.2}$$

onde: *l* é o comprimento da haste, m é a massa do motor, g é a aceleração da gravidade, k é um coeficiente de atrito, V é a tensão aplicada ao motor de corrente contínua, e c é uma constante de proporcionalidade entre a tensão aplicada ao motor e a força induzida pela hélice acoplada ao eixo do motor.

A equação do modelo de segunda ordem é obtida a partir da resposta real da planta à aplicação de um degrau para determinado ponto de operação por meio de métodos de identificação [Ljung, 1987]. A resposta ao degrau do sistema é mostrada na Figura 5.2 onde, a aquisição dos dados foi realizada a uma taxa de 10 amostras por segundo por um período de 15 segundos. A Figura 5.2 foi obtida com a aplicação de um degrau com valor 250. Como resposta foi obtida uma curva onde se podem observar três de ciclos visíveis de oscilação até a estabilização em torno do valor 115, e uma frequência do sinal amortecido ω (número de oscilações em um segundo) de 0,5Hz. Os valores analisados para posição são valores absolutos referentes ao sinal convertido do potenciômetro pelo conversor analógico-digital de 8-*bits* (valores de 0 a 255). A relação entre o valor de posição apresentado e o ângulo atingido pelo pêndulo pode ser ajustada através de potenciômetros inseridos no circuito de conversão, como descrito no Apêndice A. O ajuste realizado para os ensaios apresenta uma relação 2:1, ou seja, o ângulo atingido pelo pêndulo corresponde à metade do valor de posição apresentado.

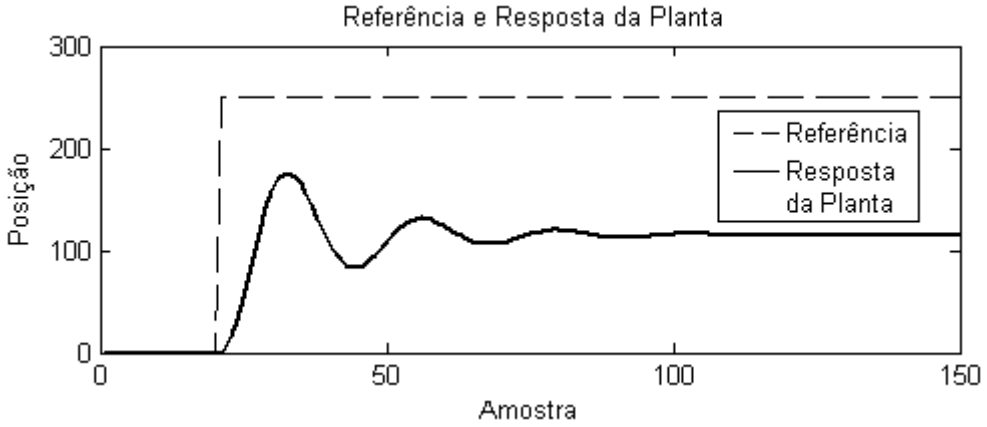


Figura 5.2 – Resposta em malha aberta.

Para o cálculo do ganho é realizada a divisão do valor de regime ou de acomodação pelo valor do degrau aplicado. Assim, o valor do ganho é dado por [Aguirre, 2007]:

$$k = \frac{115}{250} = 0{,}46 \tag{5.3}$$

A frequência natural do sistema é dada por:

$$\omega_n = 2\pi\omega = 2{,}74 \tag{5.4}$$

O quociente de amortecimento é dado por:

$$\xi = \frac{0{,}6}{\omega} = 0{,}2 \tag{5.5}$$

O modelo linear de segunda ordem do pêndulo para o ponto de operação mencionado pode ser obtido pela substituição dos parâmetros calculados nas Equações 5.3 – 5.5 na expressão [Aguirre, 2007]:

$$G(s) = \frac{k\omega_n{}^2}{1s^2 + 2\xi\omega_n + \omega_n{}^2} \tag{5.6}$$

Um modelo linear de segunda ordem do pêndulo amortecido é considerado para calcular o *fitness*:

$$G(s) = \frac{3{,}458}{s^2 + 1{,}097s + 7{,}516} \tag{5.7}$$

De posse da Equação 5.7 o sistema foi discretizado a fim de possibilitar as simulações. O período de amostragem foi definido em um décimo de segundo. A função de transferência discretizada encontrada é utilizada para as simulações.

5.2. AVALIAÇÃO DO SISTEMA DE INFERÊNCIA IMPLEMENTADO EM FPGA

O sistema de inferência nebulosa descrito na Seção 4.3 foi testado em comparação com a estrutura FIS existente no *Fuzzy Logic Toolbox* do Matlab. A estrutura do sistema de inferência nebulosa foi compilada com sucesso e simulada com a atribuição de valores nas entradas da estrutura para obtenção dos valores de saída. Os ensaios de simulação foram realizados para verificar a precisão e repetibilidade dos resultados apresentados pelo projeto em VHDL. Alguns resultados de simulações são apresentados na Figura 5.3 com entradas ERRO e VERRO e resultado do centroide na variável ZCOA.

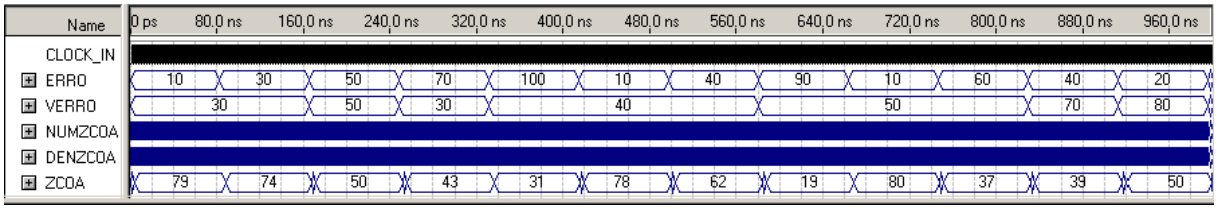


Figura 5.3 – Resultados em simulação.

A Tabela 5.1 mostra os resultados obtidos para as saídas apresentadas pela utilização do FIS da Fuzzy Logic Toolbox do Matlab® e para o FIS implementado no FPGA utilizando VHDL. A configuração do sistema de inferência para obtensão dos resultados foi realizada

com a construção das funções de pertinência de forma simétrica e sem relação com a estrutura aplicada ao modelo, servido apenas para a verificação funcional do sistema de inferência em VHDL. Os resultados foram comparados através do cálculo da porcentagem (%) de erro entre eles.

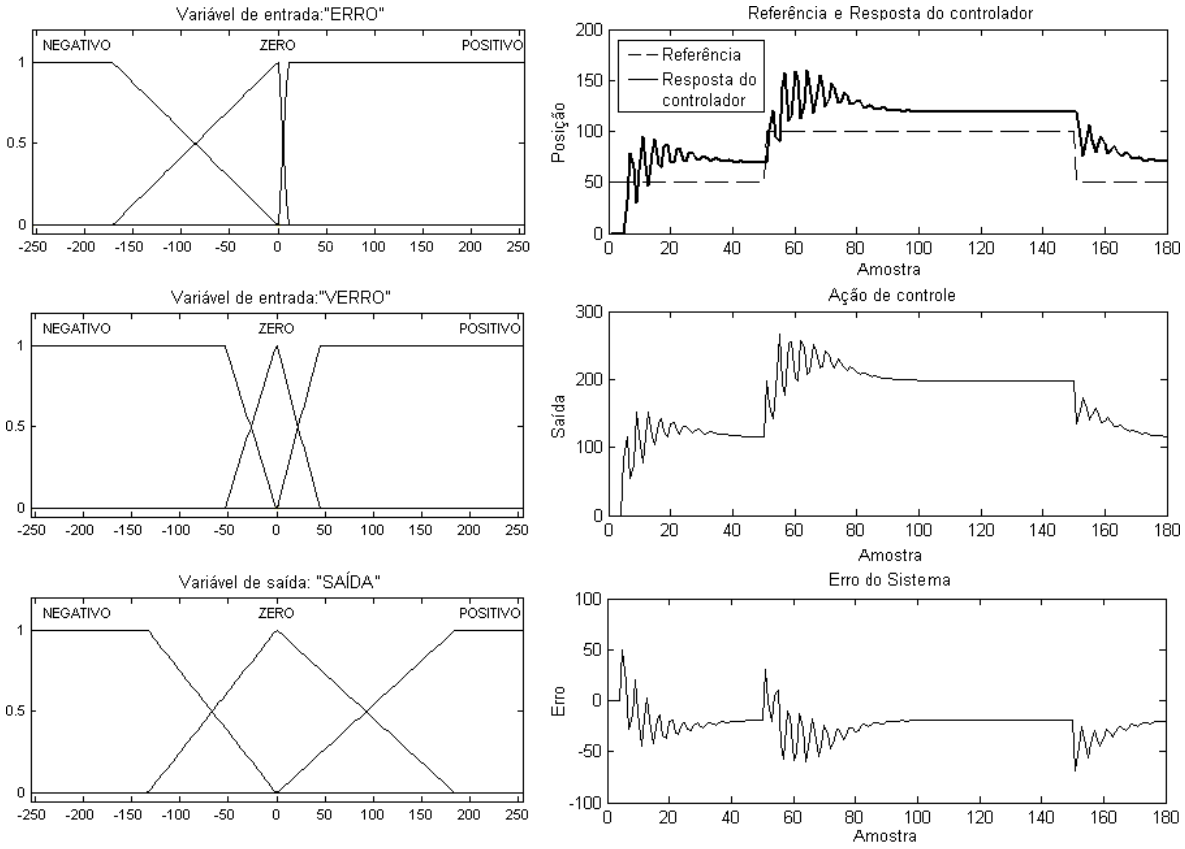
Tabela 5.1 – Resultados de simulação comparativa entre FIS implementado em Matlab e FPGA.

Erro	Variação do Erro	Resposta do Matlab	Resposta do FPGA	Erro (%)
10	30	79.8	79	1.00
30	30	74.5	74	0.67
50	50	50.0	50	0.00
70	30	43.1	43	0.23
100	40	31.4	31	1.27
10	40	78.8	78	1.02
40	40	62.9	62	1.43
90	50	19.2	19	1.04
10	50	80.8	80	0.99
60	50	37.1	37	0.27
40	70	38.8	39	0.52
20	80	50.0	50	0.00

O erro médio quadrático (RMSE) obtido é de 0,77, um valor aproximado, porém com vantagem sobre algumas implementações semelhantes [Uppalapati e Kaur, 2009].

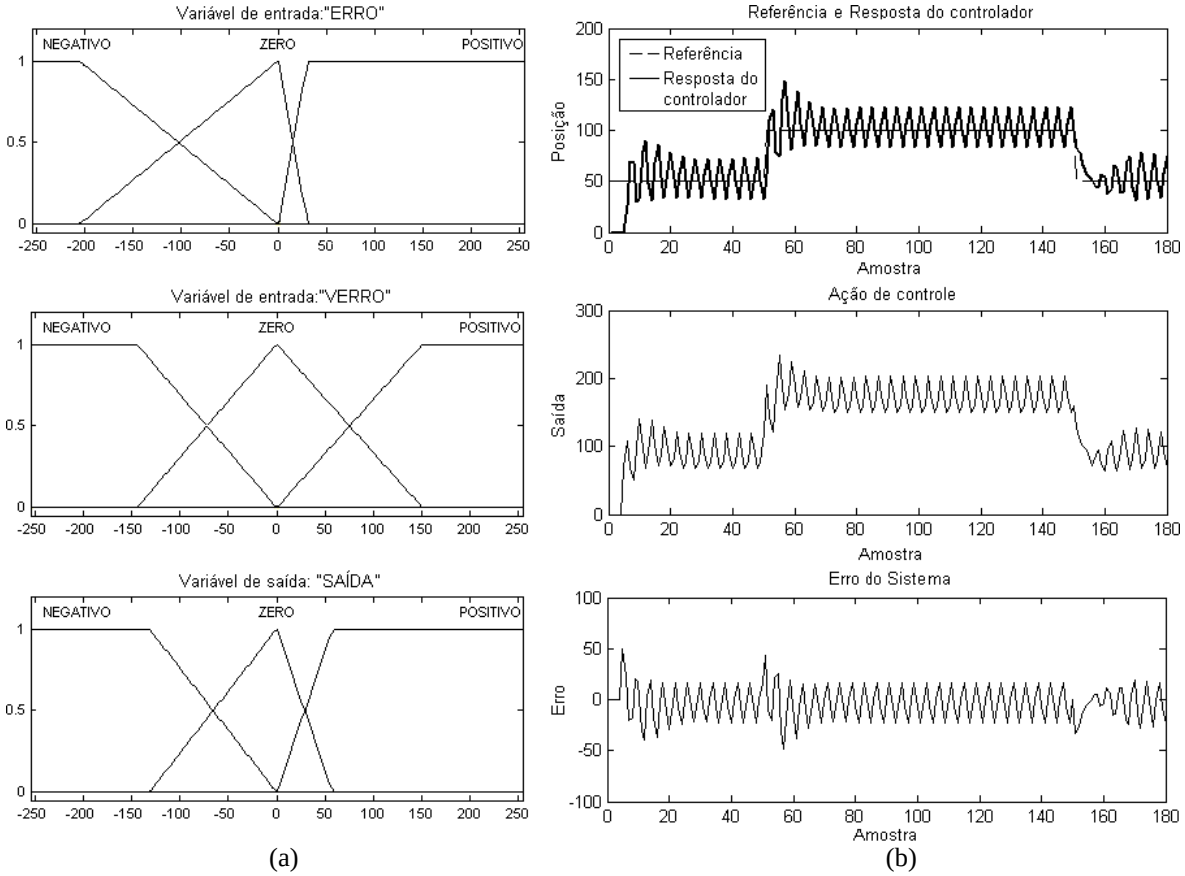
5.3. EVOLUÇÃO DOS CONTROLADORES EM SIMULAÇÃO

Os cromossomos da população inicial são criados aleatoriamente com a codificação de funções de pertinência (descrita no Capítulo 4). Os termos POSITIVO, ZERO e NEGATIVO para as variáveis linguísticas ERRO, VERRO e SAIDA são definidos a partir do código dos cromossomos. As Figuras 5.4, 5.5 e 5.6 mostram exemplos de conjuntos de funções de pertinência de indivíduos da população inicial gerada aleatoriamente (descrito na Seção 4.2) como base para a evolução ao longo das gerações, juntamente com os dados referentes à resposta ao degrau de cada controlador incorporando as características dos indivíduos.



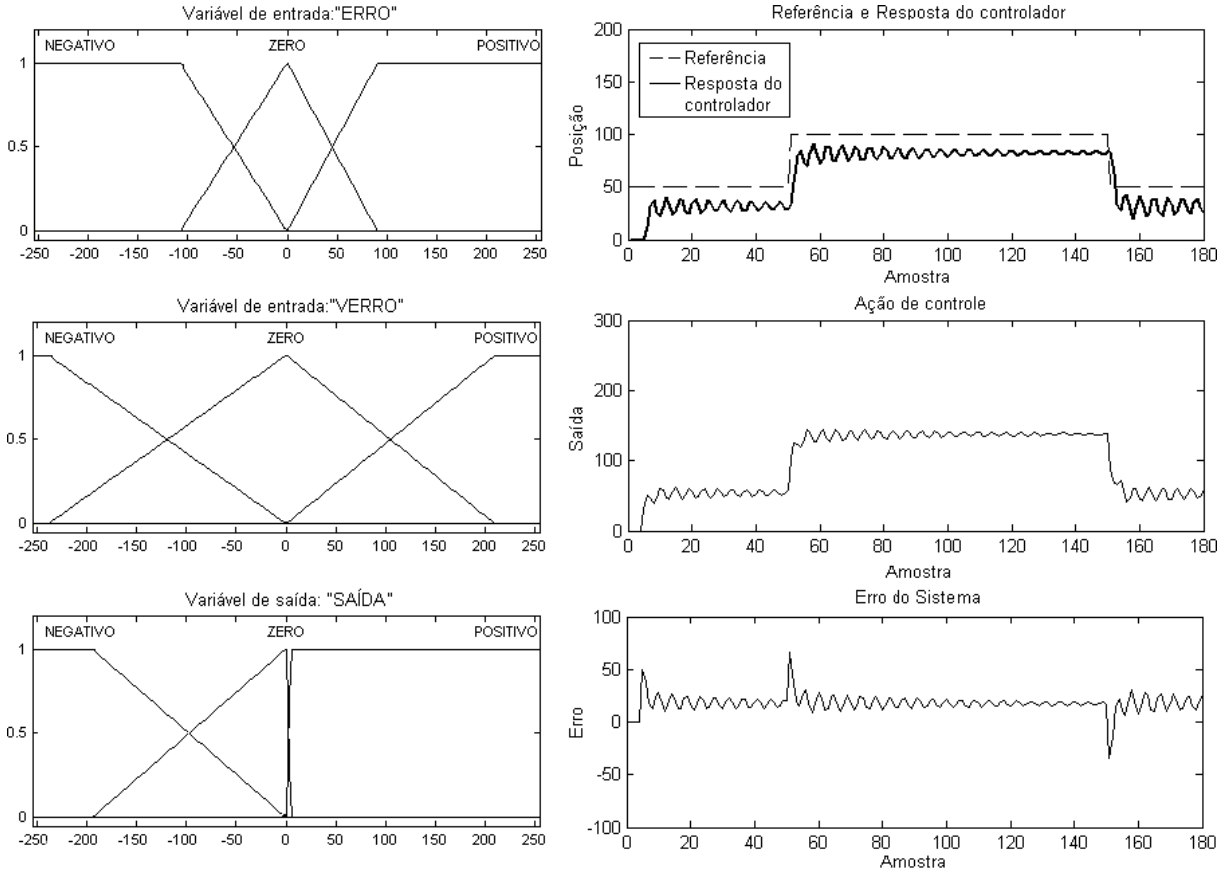
(a) (b)

Figura 5.4 – Exemplo 1 de indivíduo da população inicial: (a) funções de pertinência e (b) resposta ao degrau.



(a) (b)

Figura 5.5 – Exemplo 2 de indivíduo da população inicial: (a) funções de pertinência e (b) resposta ao degrau.



(a) (b)

Figura 5.6 – Exemplo 3 de indivíduo da população inicial: (a) funções de pertinência e (b) resposta ao degrau.

Dois problemas principais podem ser observados nos exemplos de resposta aos controladores da população inicial: grandes oscilações e grande erro de regime permanente. Os problemas apresentados são atribuídos à formação aleatória das funções de pertinência. A sintonia evolutiva por algoritmos genéticos aplicada aos controladores visa minimizar esses problemas apresentados no desempenho dos controladores.

Os problemas de oscilação e erro de regime permanente apresentados pelos indivíduos das populações iniciais estão ligados diretamente com a formação das funções de pertinência dos termos linguísticos, seja por um problema de distribuição ao longo do horizonte de discurso ou pela relação entre termos de variáveis linguísticas distintas. A forma das funções de pertinência influi sobre a formação da superfície de controle, que rege o comportamento da ação do controlador, inclusive na aproximação do ponto desejado.

A estratégia discutida na Seção 4.2 é utilizada para melhorar esta geração inicial de controladores nebulosos. O objetivo é alcançar CNs que apresentem baixos tempos de subida, sobressinal e erro de regime. A função de *fitness* (Equação 4.2) é que indica o quanto cada CN alcança esse objetivo.

Durante todo processo de evolução, as respostas do modelo linear da Equação 5.7 para os CNs associados a cada cromossomo são computados para obter as medidas necessárias para a Equação 4.2. As funções de pertinência são modificadas ao longo das gerações, mas as regras dos controladores nebulosos permanecem fixas conforme a Tabela 4.1. A evolução da função de *fitness* do melhor indivíduo de cada geração ao longo da evolução está representada na Figura 5.7.

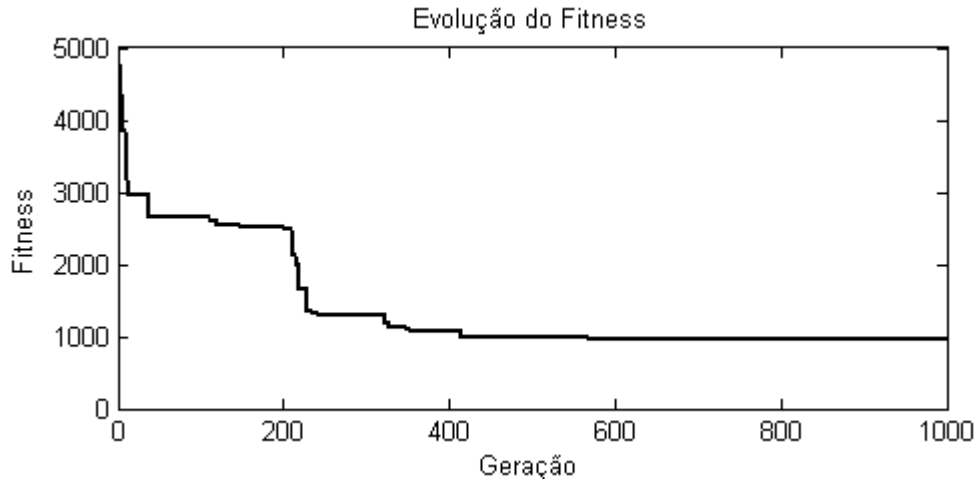
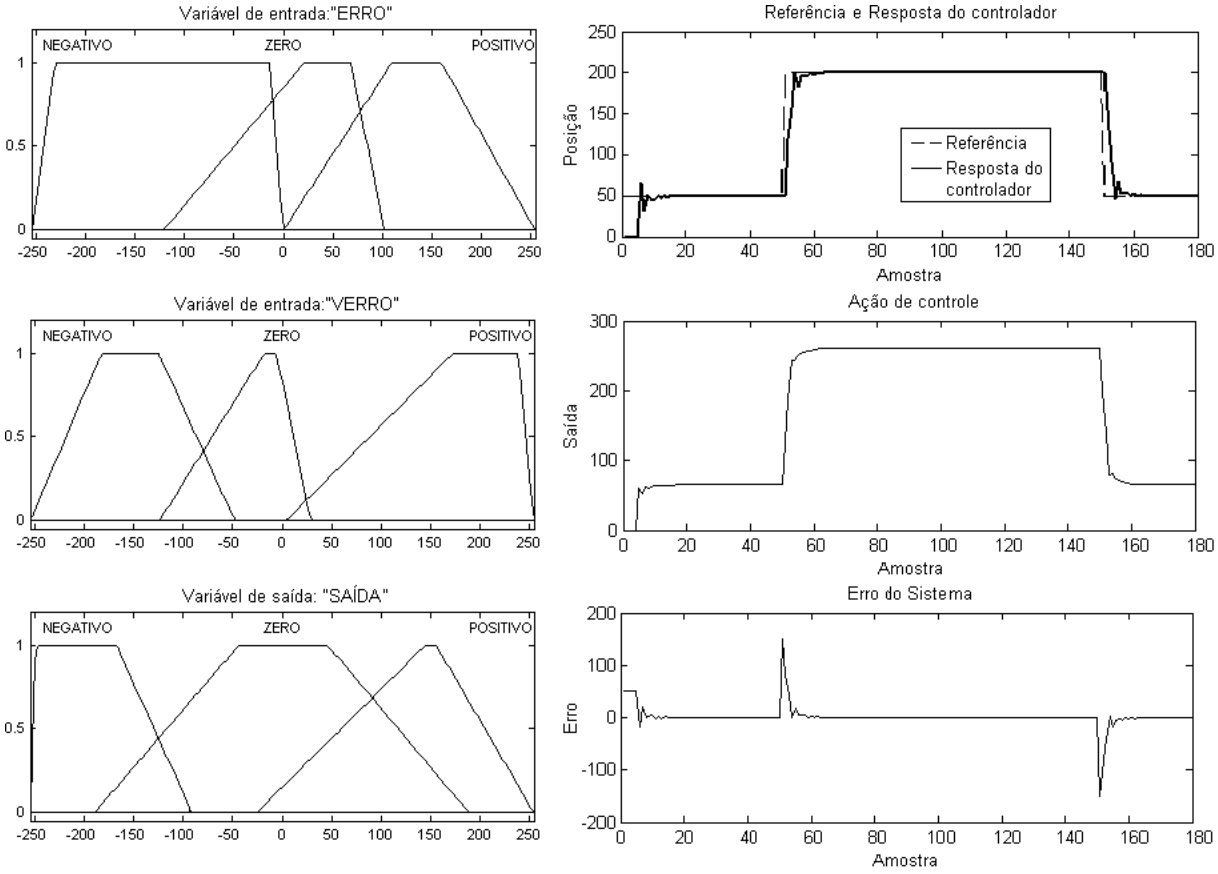


Figura 5.7 – Evolução do valor da função de *fitness* ao longo das gerações.

Os ensaios foram realizados a uma taxa de amostragem de 10 amostras por segundo. Os valores para a saída estão representados entre 0 e 255 que corresponde ao ciclo de acionamento do PWM entre 0 e 100%. O valor de erro é dado na mesma escala do valor de posição lida como descrito na Seção 5.1 e refere-se à diferença entre a referência e o estado do sistema.

O cromossomo com o melhor *fitness* após a evolução tem as funções de pertinência ilustradas na Figura 5.8. Ainda na Figura 5.8 pode-se observar a resposta obtida com o controlador de melhor desempenho após as etapas de evolução. O indivíduo considerado mais apto não apresenta comportamento oscilatório (como na população inicial), responde ao degrau em menor tempo, apresenta sobressinal da ordem de 0,01% do degrau aplicado e não apresenta erro de regime permanente. No caso específico deste ensaio, a amostra foi constituída de sete indivíduos na população e o resultado foi obtido depois de mil gerações. Os parâmetros para a probabilidade de cruzamento e mutação foram fixados em 90% e 5%, respectivamente.



(a) (b)

Figura 5.8 – Exemplo de indivíduo da população final: (a) funções de pertinência e (b) resposta ao degrau.

Comparativamente, pode ser observada claramente uma melhora na resposta do sistema após a evolução. Para se ter ideia da dimensão da diferença no desempenho entre os controladores da população inicial e após a evolução, pode-se observar a Tabela 5.2 que traz os valores dos parâmetros considerados para a formação da função de *fitness*.

Tabela 5.2 – Características relevantes ao *fitness* dos indivíduos.

	População Inicial 1	**População Inicial 2**	**População Inicial 3**	**Controlador Final**
Tempo de Subida	6	9	100	2
Sobressinal	34,13	50	0	0,06
Erro Absoluto	5049,94	10786,76	7362,87	574,25

Uma característica observada no comportamento do algoritmo de busca foi a tolerância quando se altera o ganho do controlador. Os ensaios realizados para a observação dessa característica mostraram que o algoritmo é capaz de fornecer uma solução satisfatória, mesmo com vários fatores de incremento aplicados à saída do controlador nebuloso.

A Figura 5.9 ilustra um exemplo de ensaio realizado com a aplicação de diferentes patamares de referência, onde não se evidencia degradação de desempenho.

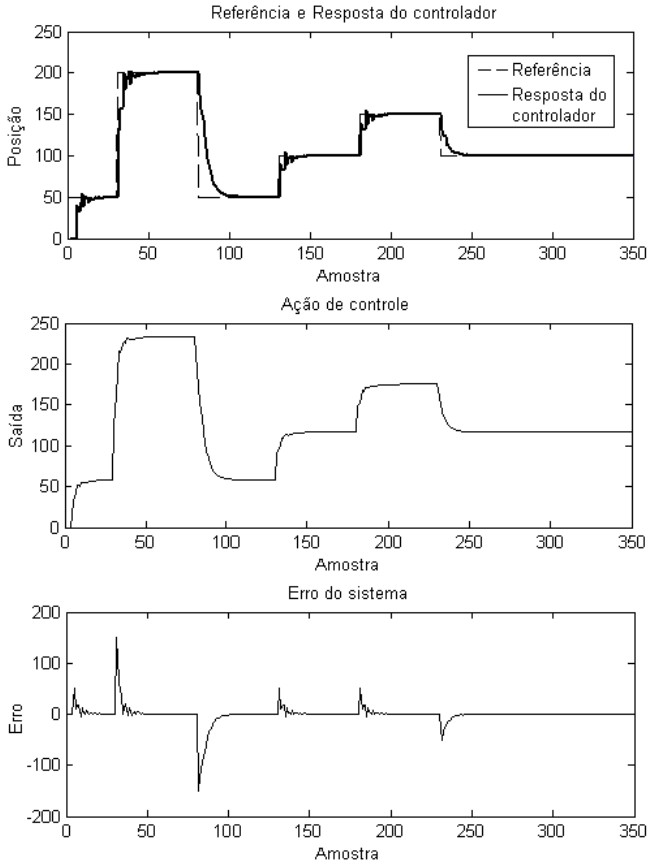


Figura 5.9 – Exemplo de resposta de controlador após evolução.

A partir da base de regras descrita na Tabela 4.1 e das funções de pertinência resultantes da evolução da população de controladores mostrada na Figura 5.8a pode ser elaborada a superfície de controle, como mostra a Figura 5.10.

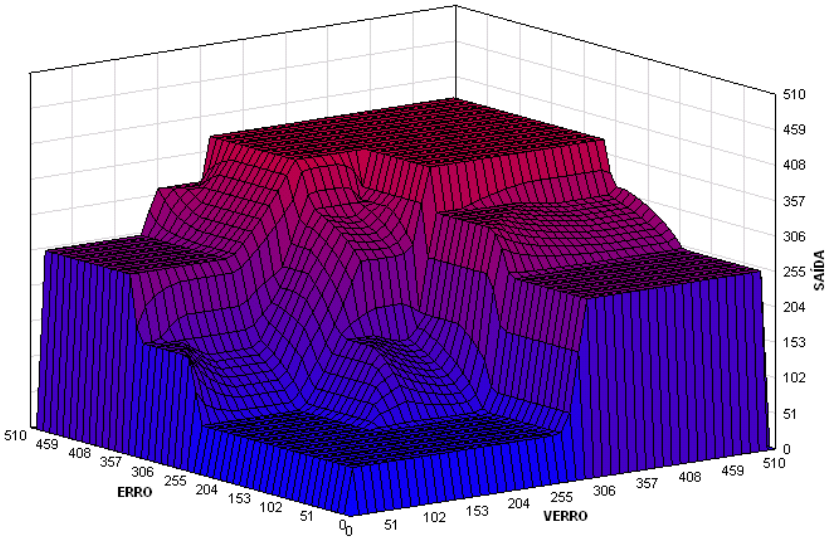


Figura 5.10 – Superfície de controle.

5.4. EXPERIMENTOS COM A PLANTA REAL

Após o processo de evolução das gerações, o indivíduo com melhor qualificação segundo o valor de *fitness* atingido (Equação 4.2) é utilizado para os ensaios sobre a planta real. Os parâmetros das funções de pertinência do indivíduo são transcritos para o código do sistema de inferência nebulosa em VHDL. A Figura 5.11 mostra o comportamento do sistema real (Figura 5.1) com a aplicação do degrau de ensaio.

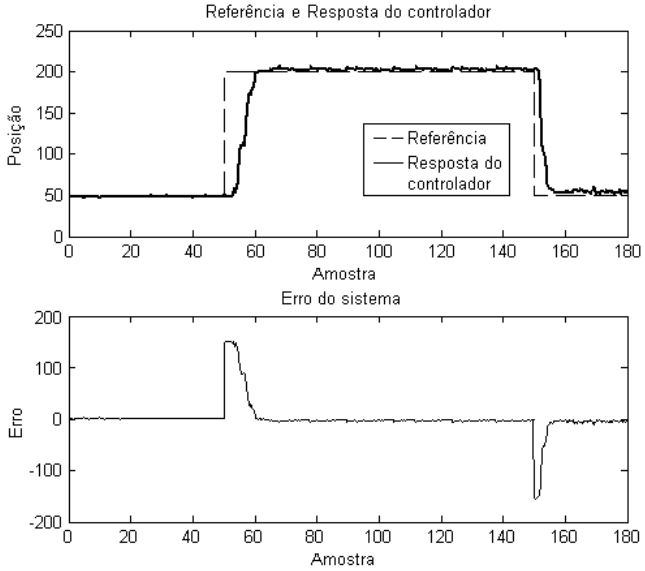


Figura 5.11 – Resposta ao degrau da planta real.

Outros ensaios foram realizados para avaliar o desempenho do controlador em pontos de operação diferentes dos utilizados para evolução por algoritmos genéticos, como podem ser vistos nas Figuras 5.12 e 5.13.

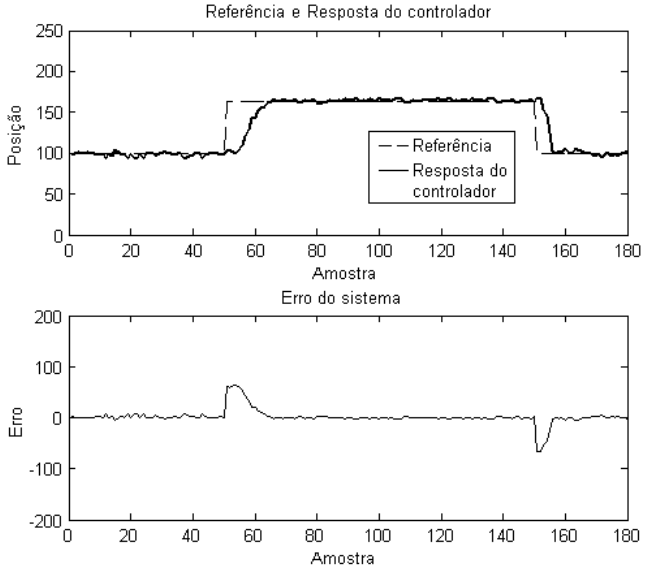


Figura 5.12 – Exemplo 1 de resposta ao degrau da planta real fora do ponto de operação treinado.

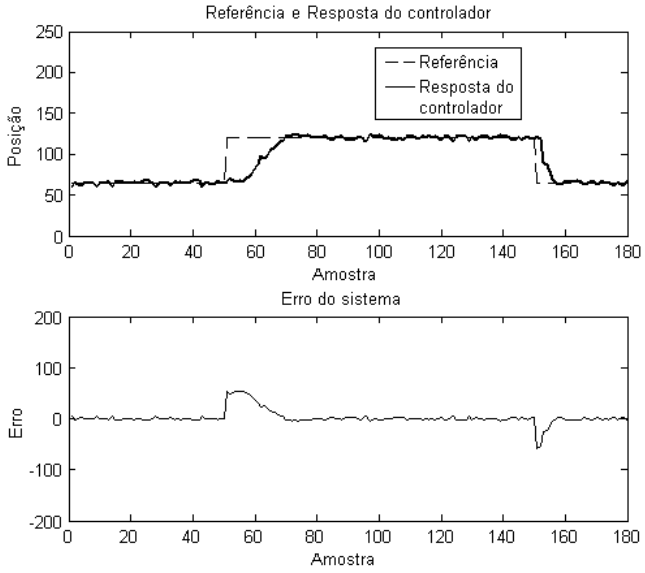


Figura 5.13 – Exemplo 2 de resposta ao degrau da planta real fora do ponto de operação treinado.

Os ensaios com a planta real foram realizados com pontos de partida diferentes do ponto de repouso da planta. Ambos os exemplos 1 e 2 de resposta do sistema à aplicação de um degrau diferente do utilizado para avaliação dos critérios de desempenho e sintonia do controlador apresentam comportamento semelhante. O ponto de partida do sistema também foi alterado.

O sistema também foi submetido a uma variação ainda maior dos parâmetros utilizados para treinamento. A Figura 5.14 ilustra o resultado de um ensaio onde o sinal de referência aplicado é dado em forma de uma rampa.

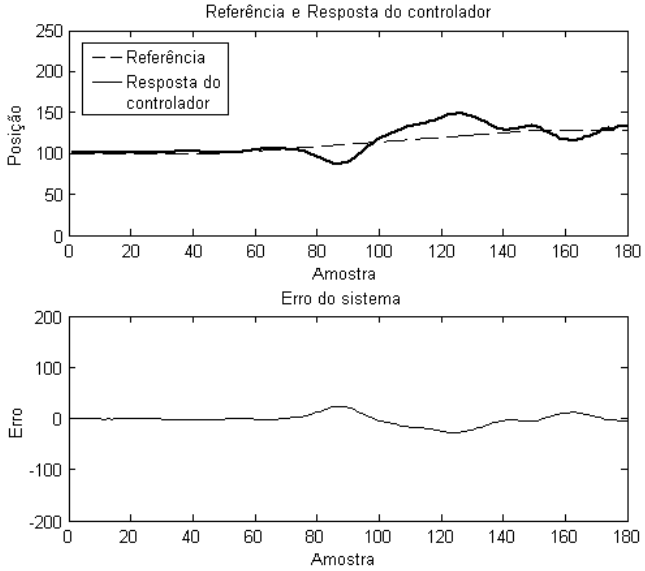


Figura 5.14 – Exemplo de resposta à rampa da planta real fora do ponto de operação treinado.

Quando o sistema foi submetido à aplicação de uma rampa como sinal de referência, foi constatada uma maior dificuldade de estabilização. A dificuldade de estabilização deve-se, principalmente, a dois fatores:

- A etapa de evolução não contemplou a análise de comportamento do sistema diante da aplicação de um sinal do tipo rampa.
- A influência de não-linearidades da planta, não previstas no modelo, sobre o comportamento do sistema real com a aplicação do sinal de rampa.

A título de comparação entre o controlador nebuloso e uma técnica de controle clássico, foi elaborado um ensaio com a aplicação de um degrau como sinal de referência a um controlador nebuloso, sintonizado pelo método descrito no Capítulo 4, com um controlador PID sintonizado pelo método de alocação de polos. Os parâmetros do PID foram ajustados para: k_p=0,158; T_i=0,146; T_d=0,912. Com o resultado da alocação dos polos do sistema foram obtidos: p_1=(0,942 + 2,996i); p_2=(0,942 - 2,996i); p_3=0,500. A Figura 5.15 ilustra os resultados obtidos para a aplicação de um degrau a um controlador nebuloso e um controlador PID. Tendo como referência os critérios apresentados na Equação 4.2, o controlador nebuloso apresenta melhor desempenho. A metodologia de sintonia evolutiva aplicada ao controlador nebuloso apresenta redução no sobressinal e no tempo de subida da resposta do sistema.

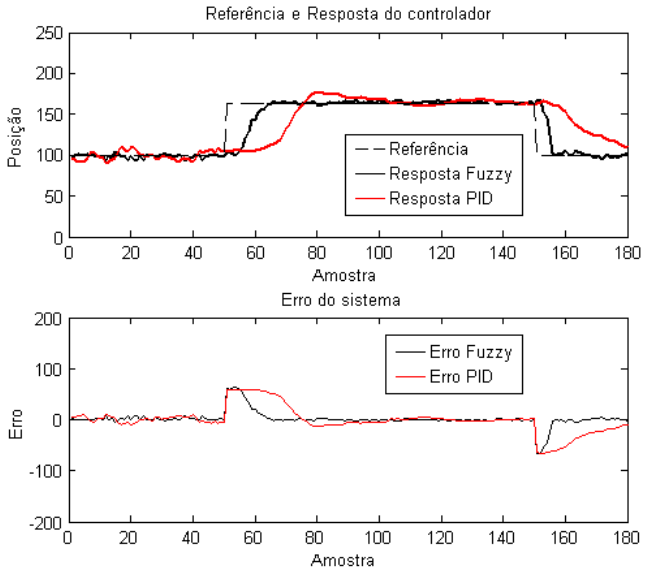


Figura 5.15 – Comparativo entre controlador nebuloso e PID.

No caso dos testes na planta real a ação de controle corresponde ao ciclo de trabalho do PWM aplicado ao circuito de chaveamento do motor propulsor apresentado no Apêndice A. O controlador PID foi implementado em FPGA como apresentado no Apêndice B.

5.5. COMENTÁRIOS FINAIS

Neste capítulo foram apresentados os resultados experimentais do comportamento do algoritmo genético como um algoritmo de busca para uma solução de problema de ajuste das funções de pertinência de controladores nebulosos. Os resultados obtidos tanto em simulação como em ensaios com a planta real demonstram a aplicabilidade da metodologia proposta.

Os experimentos com a planta real demonstraram que a alteração do tipo de sinal aplicado como referência (exemplificado com degrau e rampa) pode provocar oscilações em torno do ponto de operação. Pode-se buscar reduzir essas variações com a implementação de uma etapa de simulação com uma diversidade maior de sinais aplicados ao sistema, ou a utilização em simulação de modelos não-lineares.

Apesar de os FPGAs ainda não terem sido amplamente explorados para a implementação de controladores nebulosos, este trabalho exemplifica a capacidade de realização do projeto e utilização desses componentes como processadores de sistemas de inferência nebulosos.

CAPÍTULO 6

CONCLUSÕES

Neste trabalho foi proposta uma solução para a implementação de um sistema híbrido que envolve técnicas de evolução por algoritmos genéticos e controle nebuloso. Um sistema conhecido como pêndulo amortecido foi utilizado para obter os resultados experimentais. O trabalho engloba, de forma prática, a montagem de planta de ensaio, a identificação do sistema, o desenvolvimento de projetos implementados em FPGA, o controle nebuloso, a simulação matemática de sistemas de controle, a aplicação de algoritmos genéticos evolutivos, a aquisição de dados e a avaliação de desempenho de sistemas de controle.

A identificação do sistema em torno de um ponto de operação específico a partir de um modelo linear de segunda ordem foi utilizada como ponto de partida para o desenvolvimento dos algoritmos. Durante o desenvolvimento e implementação dos algoritmos em VHDL aplicados ao FPGA, alguns pontos pode ser destacados:

- Uma nova filosofia de programação foi desenvolvida para a programação de sistemas que operam com paralelismo.
- Algumas deficiências oriundas da ferramenta de desenvolvimento foram supridas através de artifícios matemáticos e lógicos.
- O desenvolvimento de um controlador nebuloso, com código próprio, e, aplicado em diferentes linguagens, mantendo a flexibilidade para diversas aplicações, representa um ganho significativo para o programa, pois dá suporte a uma linha de pesquisa que está em expansão no departamento, a inteligência computacional.
- O código de evolução por algoritmos genéticos desenvolvidos para este trabalho também pode servir de base para projetos futuros, assim como, já está sendo utilizado em pesquisas paralelas.

Os resultados experimentais obtidos a partir de controladores nebulosos sintonizados por algoritmos genéticos quando comparadas com amostras de resultados de controladores nebulosos configurados arbitrariamente, evidenciam a melhoria do desempenho no controle de sistemas. O algoritmo proposto tem capacidade para fornecer uma solução para sintonia de controladores nebulosos através de um método de busca global, porém heurística. O que não garante que a mesma solução seja encontrada a cada execução para determinado sistema, mesmo partindo da mesma população inicial.

Diante dos resultados obtidos com a sintonia dos controladores nebulosos a partir de um modelo linear estimado para um sistema não-linear fica demonstrado que pode-se obter boa aproximação. Porém, há indicações de que pode-se melhorar ainda mais o desempenho do controlador com o aumento da variedade de sinais de referência aplicados na etapa de treinamento.

A sistemática de projeto utilizada propicia a implementação de controladores distintos, com diferentes números de variáveis linguísticas, quantidades de termos para cada variável linguística, e diferentes bases de regras. A modularidade do algoritmo na etapa de simulação permite a alteração de etapas específicas do projeto, como a alteração da função de fitness, ou mudanças nas definições dos parâmetros de simulação como o ganho de saída do sistema nebuloso de inferência e o período de amostragem do controlador.

Como trabalho futuro é sugerida a aplicação de algoritmos de identificação de sistemas diretamente implementados em FPGA, bem como a implementação de algoritmos evolutivos para adaptação de controladores em teste diretamente em *hardware*, parte que foi realizada neste trabalho com a utilização do Matlab. Também se sugere o levantamento de modelos não-lineares a fim de se obter maior precisão nas etapas de simulação em diversos pontos de operação, bem como a aplicação de outros sinais de referência como rampas e senoides para avaliar os controladores.

Com base na sistemática e metodologia de sintonia utilizada, podem ser realizados ensaios para a comparação de desempenho entre os controladores nebulosos sintonizados por algoritmos genéticos e controladores PID, sendo aplicados diversos pontos de operação. A estrutura mostrada no Apêndice B pode ser utilizada como base para elaboração do projeto de controladores PID em FPGA.

Tomando-se os resultados obtidos torna-se interessante um estudo sobre a aplicação de sistemas de sintonia evolutiva de CNs em tempo real, aplicadas a sistemas embarcados.

REFERÊNCIAS BIBLIOGRÁFICAS

Aguirre, L. A. (2007). "*Introdução à Identificação de Sistemas – Técnicas Lineares e não-lineares aplicadas a sistemas reais*". Editora UFMG, 3 ed. 2007.

Almeida, O. M., Coelho, A. A. R. (2002). "*A Fuzzy Logic Method for Autotuning a PID Controller : SISO and MIMO Case*". 15th IFAC World Congress on Automatic Control, 2002, Barcelona. 15th IFAC, 2002.

Amore, R. d' (1998). "*Contribuições à Síntese Automática de Processadores para Lógica Nebulosa*". São José dos Campos, São Paulo, Brasil, 1998.

Amore, R. d' (2005). "*VHDL: Descrição e Síntese de Circuitos Digitais*". Rio de Janeiro, Brasil, 2005.

Åström, K. J.; Wittenmark, B. (1995). "*Adaptive control*". Addison-Wesley Publishing Company, 1995.

Bandemer, H.; Gottwald, S. (1996). "*Fuzzy Sets, Fuzzy Logic, Fuzzy Methods: with Applications*". Chichester, England, John Wiley & Sons, 1996.

Barr, M. (1999). "*Programmable Logic: What's it to Ya?*" Embedded Systems Programming, June 1999, pp. 75-84.

Barriga, A.; Sánchez-Solano, S.; Brox, P.; Cabrera, A.; Batorune, I. (2006). "*Modelling and implementation of fuzzy systems based on VHDL*". International Journal of Approximate Reasoning, pp. 164-178, 2006.

Beasley, D.; Bull, D. R.; Martin, R. R. (1993). "*An Overview of Genetic Algorithms: Part 1, Fundamentals*". University Computing, 1993.

Beasley, D.; Bull, D. R.; Martin, R. R. (1993). "*An Overview of Genetic Algorithms: Part 2, Research Topics*". University Computing, 1993.

Behmenburg, C. (1993). "*Model reference adaptive systems with fuzzy logic controllers*". Proceedings of 2nd IEEE Conference on Control Applications, Vancouver, BC, Canada, pp. 172-176.

Berstecher, R. G., Palm, R., Unbehauen, H. (1996). "*Direct fuzzy adaptation of a fuzzy controller*". Proceedings of the 13th World Congress of IFAC, San Francisco, CA, USA, vol. K, pp. 49-54.

Chalhoub, N. (2006). *"FPGA-based generic neural network architecture"*. IEEE, 2006.

Coelho, L. S.; Almeida, O. M.; Coelho, A. A. R. (2003). "*Projeto e estudo de caso da implementação de controle nebuloso*". SBA Controle e Automação, 2003.

Fairbanks, M. (2007). *"Redução de variabilidades permite otimizar os processos químicos e recuperar rentabilidades, mas exige investimento nos sistemas de controle"*. Disponível em http://www.quimica.com.br/revista/qd386/automacao5.htm acesso em 11/12/07.

Fischle, K., Schroder, D. (1999). "*An improved stable adaptive fuzzy control method*". IEEE Transaction on Fuzzy Systems, vol. 7, No 1, pp. 27-40.

Floyd, T. L. (2006). "*Digital Fundamentals*". Ed. 9, Prentice Hall, 2006.

Gerksic, S. (2006). *"Advanced control algorithms embedded in a programmable logic controller"*. Control Engineering Practices, 2006.

Goldberg, D. (1989). "*Genetic Algorithms in Search, Optimization and Machine Learning*". Addison-Wesley, 1989.

Halgamuge, S. K.; Hollstein, T.; Kirschbaum, A.; Glesner, M. (1994). "*Automatic Generation of Application Specific Fuzzy Controllers for Rapid-Prototyping*". Proceedings of Third IEEE Conference on Fuzzy Systems, pp. 1638-1641, USA, 1994.

Haykin, S. (1998). "*Neural Networks: A Comprehensive Foundation*". Ed. 2, Prentice Hall, 1998.

Jung, S. (2007). *"Hardware Implementation of a Real-Time Neural Network Controller with a DSP and an FPGA for Nonlinear Systems"*. IEEE Transactions on industrial electronics, 2007.

Kandel, A.; Langholz, G. (1998). "*Fuzzy Hardware: architectures and applications*". England, Kluwer Academic Publishers, 1998.

Khalil, H. K. (1996). "*Nonlinear Systems*". Prentice Hall, 2. Ed, 1996.

Kim, D. (2000). "*An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA System*". Proceedings of IEEE Transactions on Industrial Electronics, Vol. 47, nº3, 2000.

Kissell, T. E. (2006). *"Industrial Electronics: Applications for Programmable Controllers, Instrumentation and Process Control, and Electrical Machines and Motor Controls"*. 3. ed. Prentice-Hall, 2006.

Lago, E.; Jiménez, C. J.; López, D. R.; Sánchez-Solano, S.; Barriga, A. (1998). "*XFVHDL: A Tool for the Synthesis of Fuzzy Logic Controllers*". Proceedings of IEEE Design, Automation and Test in Europe, 1998.

Lee, C. C. (1990). "*Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I*". Proceedings of IEEE Transactions on Systems, Man, and Cybernetics, Vol. 20, Nº2, 1990.

Linden, R. (2006). "*Algoritmos Genéticos*". Brasport, Rio de Janeiro, 2006.

Ljung, L. (1987). "*System Identification: theory for the user*". Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987.

Mamdani, E. H. (1974). "*Application of Fuzzy Algorithms for Control of Simple Dynamic Plant*". Proceedings of the IEEE Control and Science, Vol. 121, pp. 298-316, 1974.

Mamdani, E. H; Gaines, B. R. (1981). "*Fuzzy Reasoning and its Applications*". Academic Press, Londres, Inglaterra, 1981.

Michalewicz, Z. (1994). "*Genetic Algorithms, Data Structures Evolution Programs*". Springer-Verlag, 3. ed, 1994.

Mitchell, M. (1999). "*An Introduction to Genetic Algorithms*". 5 ed. Bradford Book The MIT Press, 1999.

Monmasson, E.; Cirstea, M. N. (2007). "*FPGA Design Methodology for Industrial Control Systems – A Review*". IEEE Transactions on Industrial Electronics, Vol. 54, Nº 4, pp. 1824-1842, 2007.

Moraes, C. C. (2007). "*Engenharia de Automação Industrial*". 2. ed. Editora LTC, 2007.

Nakamura, K.; Sakashita, N.; Nitta, Y.; Shimomura, K.; Ohno, T.; Eguchi, K.; Tokuda, T. (1993). "*A 12b Resolution 200 kFLlPS Fuzzy Inference Processor*". Proceedings of IEEE International Solid-State Circuits Conference, pp. 182-183, 1993.

Natale, F. (2000). *"Automação Industrial"*. 2. ed. Editora Erica, 2000.

Oliveira, D. N.; Braga, A. P. S.; Almeida, O. M. (2010). "*Fuzzy Logic Controller Implementation on a FPGA using VHDL*". 29th North American Fuzzy Information Processing Society Annual Conference, NAFIPS Proceedings, Toronto, Canada, 2010.

Oliveira, D. N.; Braga, A. P. S.; Almeida, O. M. (2010). "*Plataforma de prototipagem rápida de controladores PID e nebulosos em FPGAs*". XVIII Congresso Brasileiro de Automática, CBA2010, Bonito, Mato Grosso do Sul, Brasil, 2010.

Oliveira, D. N.; Braga, A. P. S.; Almeida, O. M. (2010). "*Sintonia Evolutiva de Controladores Nebulosos*". XIV Congresso Latino americano de Automática, Santiago, Chile, 2010.

Oliveira, D. N.; Braga, A. P. S.; Almeida, O. M. (2010). "*Evolutionary Tuning of Fuzzy Controllers*". IX Portuguese Conference on Automatic Control, CONTROLO2010, Coimbra, Portugal, 2010.

Ormondi, A. R.; Rajapakse, J. C. (2006). *"FPGA implementations of neural networks"*. Dordrecht, Germany: Springer-Verlag, pp. 360, ISBN: 978-0-387-28485-9), 2006.

Passino, K. M.; Yurkovich, S. (1998). "*Fuzzy Control*". EUA, Addison-Wesley, 1998.

Patra, J. C. (2006). *"Field Programmable Gate Array Implementation of a Neural Network-based Intelligent Sensor System"*. IEEE ICARCV, 2006.

Pedroni, V. A. (2004). "*Circuit Design with VHDL*". MIT Press, Massachussets, USA, 2004.

Pedrycz, W.; Gomide, F. (2007). "*Fuzzy Systems Engineering Toward Human-Centric Computing*". Wiley-Interscience, 2007.

Rani, S. P. J. V.; Kanagasabapathy, P.; Kumar, A. S. (2005). "*Digital Fuzzy Logic Controller using VHDL*". Proceedings of IEEE Indicon 2005 Conference, Chennai, India, 2005.

Rao, V. B. (1995). "*C++ Neural Networks and Fuzzy Logic*". IDG Books Wordwide, 1995.

Reznik, L. (1997). "*Fuzzy Controllers*". Butterworth-Heinemann Newnes, 1997.

Silva, A. P. A. (2003). "*Tutorial Genetic Algorithms*". Learning and Nonlinear Models – Revista da Sociedade Brasileira de Redes Neurais, Vol. 1, No. 1, pp. 38-48, 2003.

Simões, M. G.; Shaw, I. S. (2007). "*Controle e Modelagem Fuzzy*". Ed. 2, Editora Blücher, São Paulo, Brasil, 2007.

Singh, S.; Rattan, K. S. (2003). "*Implementation of fuzzy logic controller on a FPGA using VHDL*". 22nd International Conference of the North American Fuzzy Information Processing Society - NAFIPS Proceedings, 2003.

Sivanandam, S. N.; Sumathi, S.; Deepa, S. N. (2007). "Introduction to Fuzzy Logic using Matlab". Ed. Springer, New York, 2007.

Sugeno, M. (1985). "*An Introductory Survey of Fuzzy Control*". Information Sciences, No 36, pp. 59-83, 1985.

Tanscheit, R. (1992). "*Controle Nebuloso*". Anais do 9º Congresso Brasileiro de Automática, Vitória, Espírito Santo, Brasil, 1992.

Takagi, T.; Sugeno, M. (1985). "*Fuzzy Implementation of Systems and Its Applications to Modeling and Control*". IEEE Transactions on Systems, Man, and Cybenetics, Vol. SMC-15, No 1, pp.116-132, 1985.

Thiry-cherques, H. R. (2004). *"Modelagem de Projetos"*. 2. ed. Editora Atlas, 2004.

Tocci, R. J.; Widmer, N. S. (2003). "*Sistemas Digitais: Princípios e Aplicações*". Prentice Hall, 8ª Ed, São Paulo, Brasil, 2003.

Togai, M.; Watanabe, H. (1985). "*A VLSI Implementation of Fuzzy Inference Engine: Toward an Expert System on a Chip*". Charles R. Weisbin (Ed.): Artificial Intelligence Applications, CAIA 1985, IEEE Computer Society Proceedings of the Second Conference The Engineering of Knowledge-Based Systems, Miami Beach, Florida, USA, pp. 192-197, 1985.

Uppalapati, S.; Kaur, D. (2009). "*Design and Implementation of a Mandani Fuzzy Inference system on an FPGA*". Proceedings of The 28th North American Fuzzy Information Processing Society Annual Conference. Cincinnati, Ohio, USA, 2009.

Webb, J. W. (2006). "*Programmable Logic Controllers: Principles and Applications*". 5. ed. Prentice-Hall, 2006.

Xiong, N. (2009). "*Learning Flexible Structured Linguistic Fuzzy Rules for Mamdani Fuzzy Systems*". Proceedings of FUZZ-IEEE, Korea, 2009.

Yager, R. R.; Zadeh, L. A. (1992). "*An Introduction to Fuzzy Logic Applications in Intelligent Systems*". England, Kluwer Academic Publishers, 1992.

Yen, J.; Langari, R.; Zadeh, L. A. (1995). "*Industrial Applications of Fuzzy Logic and Intelligent Systems*". IEEE Press, New York, 1995.

Ying, H.; Siler, W.; Buckley, J. J. (1990). "*Fuzzy Control Theory: A Nonlinear Case*". IFAC Automatica, Vol. 26, Nº3, pp. 513-520, Pergamon Press, 1990.

Zadeh, L. A. (1965). "*Fuzzy Sets*". Information and Control, No 8, pp. 338-353, 1965.

Zadeh, L. A. (1968). "*Fuzzy Algorithm*". Information and Control, No 12, pp. 94-102, 1968.

APÊNDICE A:

PLACA DE CONVERSÃO E ACIONAMENTO

A.1. DESCRIÇÃO DA PLACA

A Figura A.1 apresenta o desenho esquemático da placa de circuito impresso utilizada juntamente com a planta. O circuito é composto, basicamente, por um conversor analógico-digital (ADC0808), divisores de tensão para condicionamento de sinal, um circuito de potência isolado com opto-acopladores para acionamento por PWM.

O ADC0808 é um componente de baixa potência para aquisição de dados de oito canais de aquisição que realiza conversão de sinais analógicos, entre os níveis de referência, para sinais digitais através do método de aproximações sucessivas. O dispositivo elimina a necessidade de incorporar ao projeto elementos para ajuste de escala de conversão, sendo possível o ajuste através dos sinais de controle do dispositivo. A Figura A.1 a seguir ilustra o esquema de ligação do conversor.

Para realizar o acionamento do motor foi realizada abordagem de acionamento por PWM (Pulse Width Modulation).

O acionamento do motor por PWM, apesar de trabalhar com modelagem complexa para o motor, facilita a implementação tendo em vista que o controlador é totalmente digital. Sendo reduzido o número de canais necessários para o acionamento.

A frequência e o ciclo de trabalho do PWM são ajustados por software no FPGA.

Como o controlador não possui a capacidade de fornecer a potência necessária para o acionamento diretamente em seus pinos, foi elaborado um circuito de acionamento. O circuito é composto de um optoacoplador e um transistor conectado diretamente ao motor. O circuito isola eletricamente as duas partes: controle e acionamento. A Figura A.2 representa o diagrama de interligação entre o controlador e o circuito de acionamento.

A.2. FUNCIONAMENTO DO CIRCUITO

As placas do controlador e de interface são conectadas através de um cabo do tipo fita comumente usado para comunicação de discos rígidos IDE. O conector de 40 pinos possui conexão com 36 pinos de entrada e saída do FPGA, e também aos sinais de +5VDC, +3.3VDC e dois pontos para conexão à referência (GND). Os sinais de controle são providos pelo FPGA, que recebe as leituras e sinais de sincronismo do circuito.

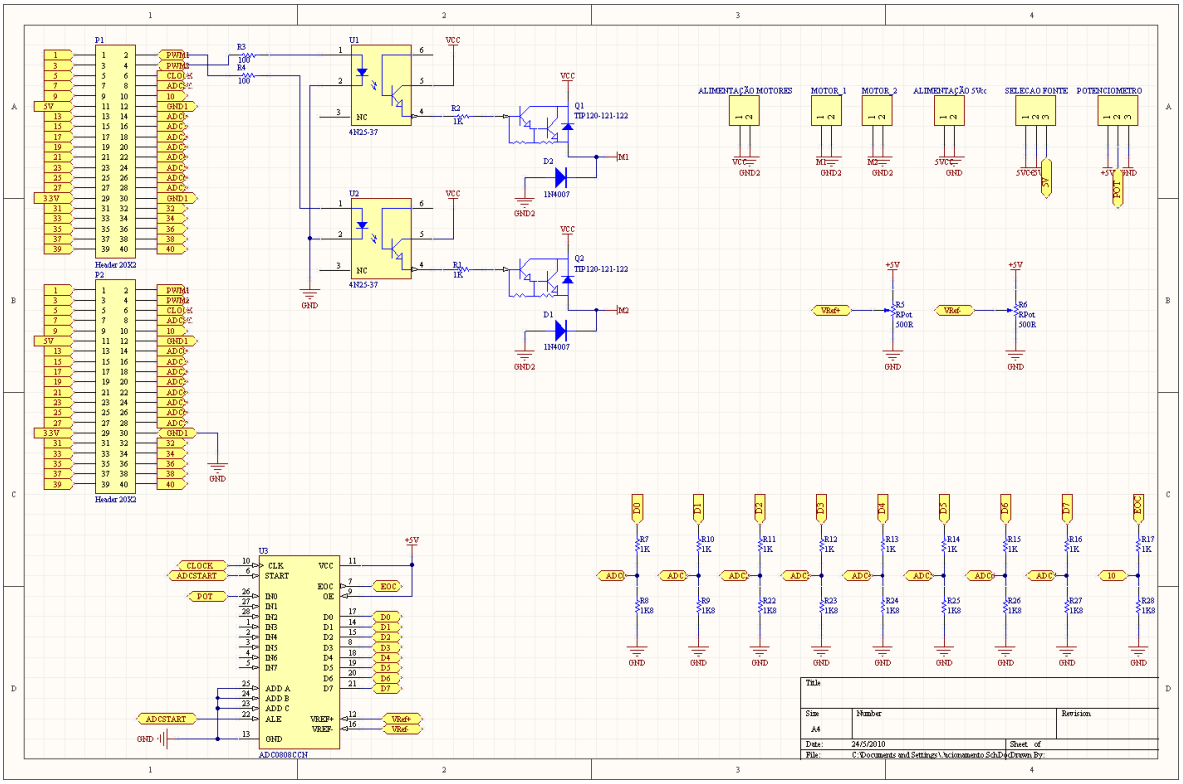


Figura A.1 – Desenho esquemático da placa.

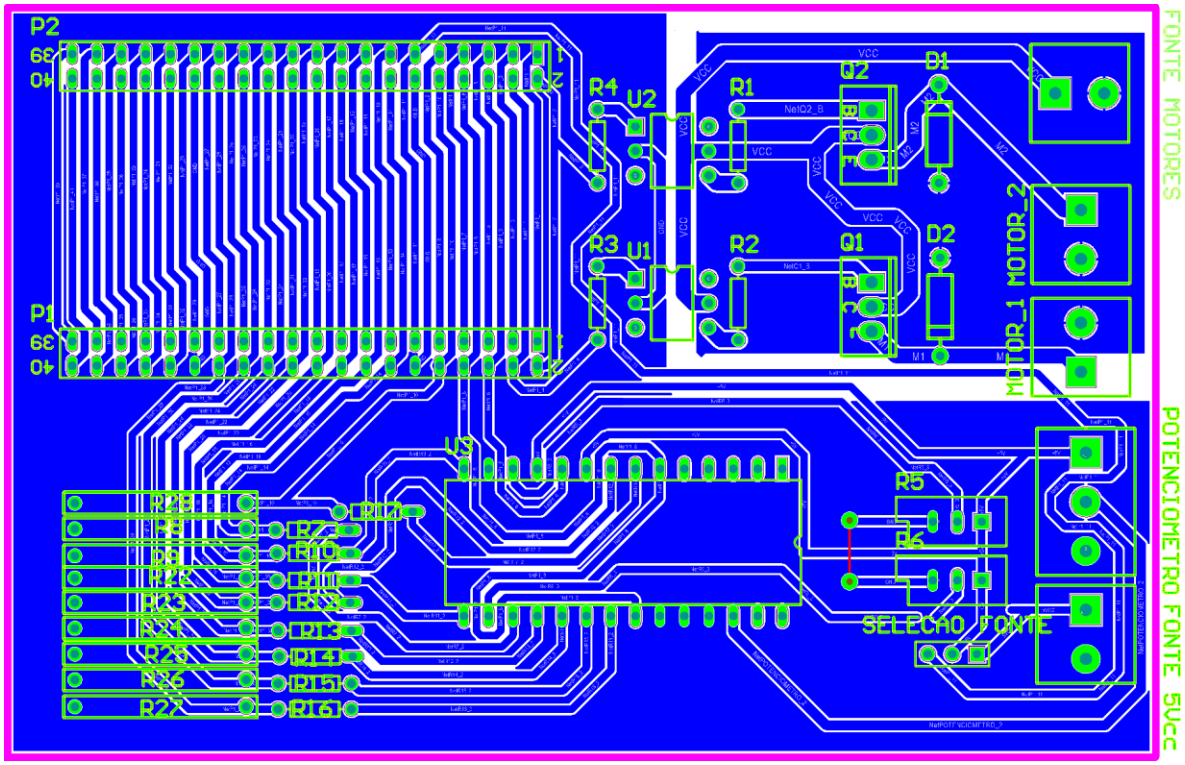


Figura A.2 – Desenho do circuito.

APÊNDICE B:

CONTROLADOR PID IMPLEMENTADO NO FPGA

Discretizando a equação do algoritmo PID de posição representada em (B.1) obtém-se o sinal de controle discreto, m_k, a partir de (B.2). Tendo S_k representado em (B.3).

$$m(t) = k_c e(t) + \frac{k_c}{T_i} \int_0^t e(t)dt + k_c T_d \frac{de(t)}{dt} \tag{B.1}$$

$$m(t) = k_c e_k + k_i S_k + k_d (e_k - e_{k-1}) \tag{B.2}$$

$$S_k = S_{k-1} + e_k \tag{B.3}$$

E os novos parâmetros estão relacionados com os analógicos da forma da equação B.4:

$$\begin{cases} k_p = k_c \\ k_i = k_c \dfrac{\Delta T}{T_i} \\ k_d = k_c \dfrac{T_d}{\Delta T} \end{cases} \tag{B.4}$$

Também pode-se obter o valor da variação da variável manipulada em cada instante ao invés de seu valor absoluto, derivando o algoritmo de velocidade da equação:

$$\Delta m_k = m_k - m_{k-1} \tag{B.5}$$

$$\Delta m_k = e_k (k_p + k_i + k_d) - e_{k-1}(k_p + 2k_d) + e_{k-2} k_d \tag{B.6}$$

A figura a seguir ilustra o diagrama de blocos que compõe o controlador PID. O bloco ‘CONVERSOR_AD’ é responsável por todo o controle do conversor analógico digital, que também fornece o sinal de fim de conversão para ser utilizado como sinal de estímulo para outras etapas do processo. O bloco ‘NORMALIZA’ trata-se de um simples filtro para evitar ruídos de leitura. O bloco ‘GERADOR’ é responsável pela formação dos sinais de referência em forma de uma sequência de degraus. No bloco ‘PID’ está contido todo o equacionamento do controlador em VHDL. Outra implementação em diagrama de blocos com a utilização de *flip-flops* do tipo D também está sendo desenvolvida.

A Figura B.1 mostra o diagrama de blocos elaborado para testes com o controlador PID onde se utilizam blocos de função e blocos de encapsulamento de código em VHDL.

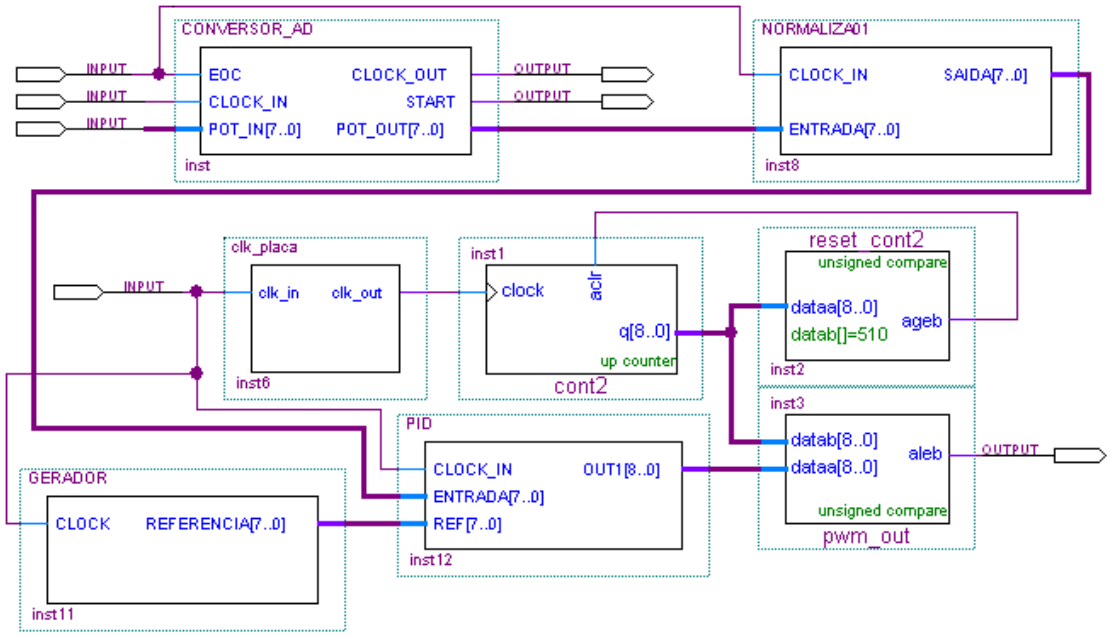


Figura B.1 - Diagrama esquemático do controlador PID.