UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA ELETRÔNICA

JEAN CESAR GONÇALVES DE FREITAS

# CONTROLADOR FUZZY IMPLEMENTADO EM FPGA APLICADO AO CONTROLE DE SERVOMOTORES

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2019

JEAN CESAR GONÇALVES DE FREITAS

# CONTROLADOR FUZZY IMPLEMENTADO EM FPGA APLICADO AO CONTROLE DE SERVOMOTORES

Trabalho de Conclusão de Curso de Graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2 – TCC2, do curso Superior de Engenharia Eletrônica, do Departamento Acadêmico de Eletrônica (DAELN), da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para obtenção do título de Engenheiro em Eletrônica.

**Orientador :** Prof. Dr. Marcio Rodrigues da Cunha

CAMPO MOURÃO
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
*Campus* Campo Mourão
*Coordenação de Engenharia Eletrônica*

TERMO DE APROVAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO INTITULADO

CONTROLADOR FUZZY IMPLEMENTADO EM FPGA APLICADO AO CONTROLE DE SERVOMOTORES

DO DISCENTE

JEAN CEZAR GONÇALVES DE FREITAS

Trabalho de Conclusão de Curso apresentado no dia 01 de julho de 2019 ao Curso Superior de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, Campus Campo Mourão. O discente foi arguido pela Comissão Examinadora composta pelos professores abaixo assinados. Após deliberação, a comissão considerou o trabalho aprovado.

_____

Prof. Eduardo Giometti Bertogna
UTFPR

_____

Prof. Roberto Ribeiro Neli
UTFPR

_____

Prof. Marcio Rodrigues da Cunha
UTFPR

O termo de aprovação original, assinado, encontra-se na coordenação do

**Resumo**

Neste trabalho descrevem-se o projeto de um controlador inteligente do tipo *Fuzzy* em hardware FPGA para servomotores comumente utilizados em manipuladores robóticos. O controlador foi primeiramente projetado em ambiente virtual, e por fim implementado em um kit de desenvolvimento FPGA, onde apresentou um controle eficiente e de acordo as especificações do projeto, mostrando que o casamento entre um controlador do tipo *Fuzzy* e hardware FPGA criam um sistema de controle flexível.

**Palavras Chaves**: Logica Fuzzy, FPGA, VHDL, Servomotor, Manipulador Robótico.

**ABSTRACT**

In this work we describe the design of a Fuzzy-type intelligent controller in FPGA hardware for servomotors used in robotic manipulators. The controller was first designed in virtual environment, and finally implemented in an FPGA development kit, where it presented an efficient control and according to the project specifications, showing that the marriage between a Fuzzy type controller and FPGA hardware creates a system of flexible control.

**Keywords**: Fuzzy Logic, Servomotor, Robotic Manipulator, FPGA, VHDL

“Não creio que haja emoção mais intensa para um inventor do que ver suas criações funcionando. Essa emoção faz você esquecer de comer, de dormir, de tudo”

Nikola Tesla

## Sumário

# LISTA DE FIGURAS

# LISTA DE SIGLAS E ABREVIAÇOES

| | |
|---|---|
| I.A | Inteligência artificial |
| IFR | International Federation of Robotics |
| FPGA | Field Programmable Gate Array |
| VHDL | VHDSIC Hardware Description Language |
| PI | Controle proporcional integrativo |
| PID | Controle proporcional integrativo e derivativo |
| CPU | Central Processing Unit |
| GPU | Graphics Processing Unit |
| RAM | Random Access Memory |
| ADC | Analógico-Digital |
| PWM | Pulse Width Modulation |
| N | Negativo |
| P | Positivo |
| MN | Muito Negativo |
| MP | Muito Positivo |
| DN | Devagar Negativo |
| DP | Devagar Positivo |
| Z | Zero |

# LISTA DE TABELA

# 1 INTRODUÇÃO

Em 1924, Roy J. Wensley, engenheiro elétrico da Westinghouse, desenvolveu a primeira unidade de controle supervisionada. Desde então, os robôs vêm alcançando sucesso e tomando cada vez mais espaço dentro da linha de produção industrial mundial, realizando tarefas consideradas perigosas ou impossíveis para seres humanos. (SOUSA, 2011)

Os robôs utilizados na indústria, na sua na grande maioria, são do tipo manipuladores fixos a uma superfície. Este modelo, o primeiro tipo de robô eletrônico autônomo de uso industrial, construído a pedido da General Motors em 1961, chamava-se Unimate. Este robô trabalhava pegando peças aquecidas de metal e posicionando-as nos chassis dos carros, pesava 1.800Kg e obedecia a comandos gravados em fitas magnéticas. (AYRES, 2007)

Com o uso de métodos de Inteligência Artificial, cada vez mais comum nos robôs industriais, o desenvolvimento da robótica acelerou de tal forma que hoje se tornou ferramenta indispensável para o setor produtivo. Seu uso se disseminou para além dos limites da linha de produção fabril, empregando robôs em diversos setores produtivos, tais como o setor agrícola, o setor da construção civil, o setor de serviços, o setor de transportes, dentre outros.

Hoje, estas ferramentas são capazes de substituir praticamente qualquer trabalhador braçal. O Japão, por exemplo, é considerado o pais que mais utiliza destas ferramentas, com cerca de 306 robôs por grupo de 10 mil trabalhadores, segundo dados da International Federation of Robotics de 2014. (FERRAZ, 2018).

Mas, para que estas ferramentas realizem diferentes funções, necessitam de diferentes níveis de complexidade de controle. Uma importante ferramenta para o projeto dos controladores de ferramentas robóticas foi desenvolvida em 1965, por Lotfi Asker Zadeh (ZAH-da), professor em Berkeley - Universidade da Califórnia, chamada lógica *Fuzzy.* (ABAR, 2004).

Lotfi Zadeh visualizou a necessidade da criação da lógica *Fuzzy*, pois considerava que os recursos tecnológicos baseados na lógica booleana não eram eficientes para resolver problemas de automação de atividades relacionadas à natureza industrial, biológica ou química, onde o método *Fuzzy* pode assumir infinitos valores entre o intervalo de 0 a 1, enquanto a lógica booleana em regra assume apenas valores de 0 ou 1. (ABAR, 2004).

Hoje, manipuladores modernos podem ser projetados de diferentes maneiras, com a utilização de controle em malha aberta ou com controladores PI ou PID em malha fechada, com uso de microcontroladores ou microprocessadores.

Uma forma interessante de se construir o hardware de um robô industrial é com um dispositivo chamado FPGA (*Field Programmable Gate Array*), inventado em 1984, por Ross Freeman e Bernard Vonderschmitt, co-fundadores da empresa Xilinx. O FPGA é um tipo de hardware que pode ser reconfigurado de acordo com as necessidades do usuário, tornando os robôs industriais mais versáteis para linhas de produções que constantemente passam por reformulação e reposicionando de seus robôs. (LOURENÇO, 2011)

A evolução das ferramentas robóticas depende do estudo de métodos alternativos de controle e de hardware, buscando uma maior flexibilidade, a diminuição de custos de produção e uma maior qualidade das ferramentas.

# 2. OBJETIVO

Essa proposta pretende articular um trabalho envolvendo o estudo e a implementação de técnicas de IA, no caso Logica *Fuzzy*, e de lógica programável (FPGA e VHDL) para aplicação em servomotores. Tem o objetivo de utilizado do casamento entre estas duas técnicas para buscar criar um sistema mais flexível e competitivo em relação aos métodos convencionais.

Portanto, o escopo deste trabalho é desenvolver o controlador para o acionamento de servomotores utilizados em manipuladores robóticos fixo em superfície, utilizando controlador *Fuzzy* implementado em um hardware FPGA.

## 2.1. Objetivos específicos

O controlador a ser projetado buscará controlar servomotores em sua velocidade de arranque e de chegada, acelerando progressivamente até chegar a uma velocidade adequada, podendo ter este controle de velocidade facilmente alterado de acordo com as necessidades da aplicação.

A utilização de um hardware FPGA, tem como objetivo principal tornar o controlador mais flexível em relação à quantidade de servomotores que poderem ser controlados por ele, permitindo controlar mais ou menos servomotores sem necessariamente majorar o tempo de resposta do controlador.

## 2.2. Justificativa

O uso cada vez mais comum de manipuladores robóticos, demostra que este tipo de equipamento está ganhando mais destaque dentro da indústria, sendo necessárias que eles estejam em constante evolução, que sejam buscadas novas

tecnologias de construção e desenvolvidas novas pesquisas interligando áreas da mecânica, elétrica, eletrônica, computação e inteligência artificial.

# 3. ESTADO DA ARTE.

Atualmente, manipuladores robóticos são largamente utilizados em diversos setores da indústria. Com o aumento da produção destes manipuladores robóticos e o alto custo da mão de obra humana, a tendência é o aumento do uso dos manipuladores robóticos, motivando o interesse mundial por pesquisas sobre estas ferramentas que abrange os campos da Elétrica, Eletrônica, Mecânica, Computação e Inteligência Artificial.

Hoje, a maior parte dos manipuladores robóticos são implementados em processadores ou microcontroladores comerciais. O trabalho de conclusão de curso CONTROLE NEURO-FUZZY PARA UM MANIPULADOR ROBÓTICO produzido por Eduardo Delinski dos Santos e Felipe Escobar, por exemplo, mostra a utilização de um kit de desenvolvimento Arduino™ com núcleo ATMega 328, utilizando este microcontrolador para gerenciar um controlador *Fuzzy* aplicado a um manipulador robótico. O núcleo ATMmega 328 consegue atender perfeitamente as suas necessidades, porém, seu o desempenho fica cada vez mais comprometido conforme aumenta a quantidade de elementos a serem controlados. Por isto, o projeto proposto pretende utilizar um hardware FPGA, que pode ser convertido em um hardware dedicado de grande eficiência, trabalhando com processamento paralelo, diferentemente do processamento que em regra é feito pelos microcontroladores convencionais.

Em relação ao sistema de controle, boa parte da indústria utiliza em seus robôs controle do tipo PI ou PID. A Lógica *Fuzzy* tem ganhando destaque em diversos setores industriais, mostrando-se como uma alternativa eficiente para o projeto dos controladores. Logo, este será o método a ser implementado neste projeto.

## 3.1. Fundamentos e constituição de um servomotor

Manipuladores robóticos, na maioria das vezes realizam movimentos mecânicos através de seus servomotores.

Os servomotores são unidades eletromecânicas que proporcionam o posicionamento de uma engrenagem de acordo com um sinal de entrada. São constituídos por um atuador, um sensor, um circuito de controle e uma caixa de redução.

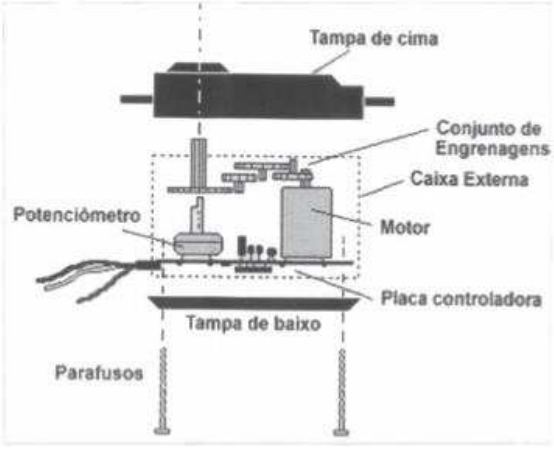
O atuador em regra constitui-se de um motor elétrico de corrente continua, mas alguns servomotores possuem um motor elétrico de corrente alternada.

O sensor utiliza-se de um pequeno potenciômetro acoplado a caixa de redução, que através da variação de tensão informa a posição angular do atuador.

O circuito de controle é constituído de componentes eletrônicos discretos ou em modelos mais sofisticados por circuitos integrados com um controlador PID (controle proporcional integrativo e derivativo), que posiciona o atuador na posição correta. (SILVEIRA, 2017)

A caixa de redução, acoplada ao atuador, é formada por um conjunto de engrenagens figura 1 que criam uma redução ao motor, ampliando drasticamente o torque do atuador, mas sacrificando sua velocidade e angulação, por isto, que muitos servomotores têm angulação te trabalho de no máximo 180 graus.

**Figura 1 -** Componentes internos de um Servomotor




Fonte: https://image.slidesharecdn.com/partesdeunservomotor-111121123301-phpapp02/95/partes-de-un-servo-motor-1-728.jpg?cb=1321879615

## 3.2. Unidade de controle e processamento de um manipulador robótico.

O processamento das tarefas de um robô pode ocorrer de diversas formas, mas basicamente o sistema é constituído de uma parte física “hardware” e uma parte lógica “software”. A parte de hardware é composta por um mecanismo de processamento como um microcontrolador ou um microprocessador, enquanto, a parte de software é responsável por interpretar os dados de entrada dos sensores espalhados pelo manipulador e campo de trabalho. A forma de interpretação desses dados varia de acordo com a funcionalidade do manipulador.

## 3.3. Logica Fuzzy

A lógica *Fuzzy*, também conhecida como lógica nebulosa, surgiu com o intuito de introduzir um método para que o computador possa trabalhar de forma mais natural e precisa, diferentemente da lógica *Booleana* que atribui valores extremos representados por 0 e 1. A lógica *Fuzzy* funciona introduzindo valores intermediários entre os extremos.

Por exemplo, para descrever a altura de uma pessoa em lógica booleana, a mesma pode representar uma pessoa com variáveis numéricas, alta (1) ou baixa (0), mas muitas vezes estes valores são insuficientes para se representar uma população. A lógica *Fuzzy* utiliza variáveis linguísticas que são elementos simbólicos, sendo possível através de uma associação *Fuzzy*, atribuir infinitos valores entre uma grandeza a ser mensurada, por exemplo, é possível representar indivíduos de uma população como baixos, medianos, altos e muito altos, e desta forma o computador pode fornecer uma resposta intermediaria entre eles. (ABAR, 2004)

Hoje, a lógica *Booleana* ainda domina a maior parte da indústria, pois o fato de processar informações de forma imprecisa como na lógica *Fuzzy* não agradou os projetistas. Porém, a lógica *Fuzzy* vem mostrando resultados muito satisfatórios e com certas vantagens como:

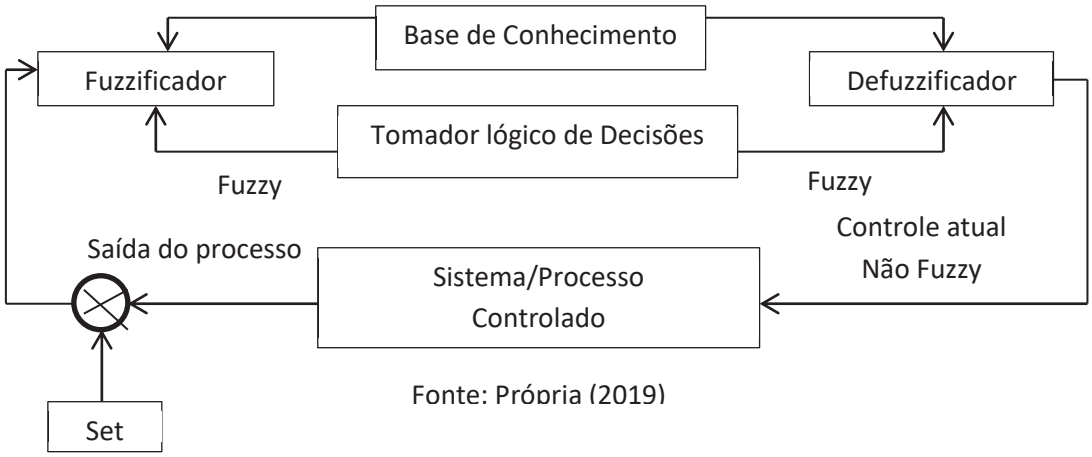
- Informações e conhecimentos dos operadores humanos podem ser inseridos na lógica *Fuzzy*.
- Útil quando as variáveis de informações não são definidas como termos exatos.
- Diminuição no tempo de desenvolvimento do algoritmo.
- Em regra, redução no custo computacional.

### 3.3.1. Controladores Fuzzy

Um sistema *Fuzzy* é uma coleção de variáveis linguísticas de entrada, uma coleção de variáveis linguísticas de saída, unidas por um conjunto de regras lógicas. Esse conjunto de regras tem como base de criação a incorporação do conhecimento subjetivo dos operadores humanos, que com o passar do tempo de trabalho adquiriram um bom conhecimento sobre o sistema a ser controlado.

Controladores *Fuzzy* trabalham em três etapas de processo figura 2, sendo elas: Fuzzyficação, Inferência e Defuzzyficação. Estas etapas criam variáveis linguísticas com um certo grau de pertinência através de parâmetros do mundo real. Desta forma, diferentes entradas acionam variáveis linguísticas com diferentes valores de pertinência gerando ao final do processo de Defuzzyficação um valor absoluto que retorna na malha de controle.

**Figura 2: Diagrama de um controle Fuzzy**




Fonte: Própria (2019)

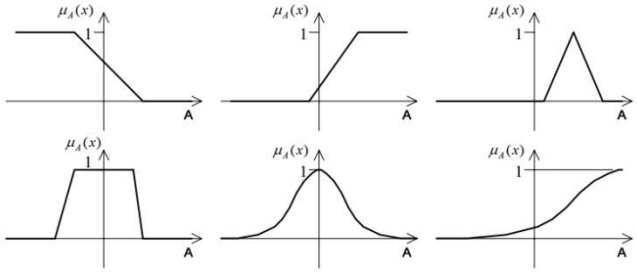
### 3.3.2. Fuzzyficação

O método *Fuzzy* traz ao computador uma forma de pensar mais parecida à forma de pensar humana. Uma pessoa que esteja numa função de operador de processos não precisa de um valor exato e definido para uma variável, por exemplo, velocidade. Tal operador consegue classificar a informação de velocidade em conjunto, do tipo Baixa, Média, e Alta, esses valores representam valores "*Fuzzificados*" dos valores exatos de velocidade, ou seja, a etapa de *Fuzzificação* cria variáveis linguísticas com certos graus de pertinência definidos por operadores humanos que possuem conhecimento prévio sobre o sistema. (TRIERWEILER, 2003)

### 3.3.3. Funções de pertinência.

São funções que definem o grau de pertinência. Geram uma curva de infinitas possibilidades. Os valores dados aos graus de pertinência são representados por funções geralmente de formas simples figura 3 como funções do tipo triangulares, trapezoidais ou gaussiana. A escolha do tipo de função é feita dependendo do nível de precisão desejado para o sistema e o custo computacional que o projetista está disposto a usar.

**Figura 3 -** Formas comuns utilizadas para função de pertinência na lógica fuzzy




Fonte: Disponível em <http://computacaointeligente.com.br/artigos/fundamentos-da-logica-fuzzy/>. Acesso em 07 abril 2019

Os valores para os graus de pertinência devem estar contidos entre os valores mínimos e máximos da função. Por questão de simplificação, o valor de máximo é representado geralmente pelo valor 1 e o valor mínimo por 0. Em termos práticos, utilizando a função de pertinência triangular, dada pela equação 1, a função assume o valor máximo, ou seja, 1 quando o sistema se encontra no ponto b, e assume o valor mínimo quando se encontra no valor a e c.

$$F(x)\begin{cases}0, & x < a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & x > c\end{cases} \qquad (1)$$

### 3.3.4. Conjuntos Fuzzy

Um conjunto é a união de algo, podendo ser definido como uma lista, uma coleção, uma classe de objetos, dentre outras definições. Na matemática, um elemento pode ter duas características relacionadas aos conjuntos, o elemento pode ou não ser pertencente a um conjunto. Partindo deste princípio, na lógica booleana binaria, um elemento em um conjunto pode ser definido como 1 ou 0, ou seja, pertencente ou não pertencente a um conjunto. (TRIERWEILER, 2003).

Dentro da lógica *Fuzzy*, um elemento não precisa ser definido de forma tão extrema dentro de um conjunto. Na lógica *Fuzzy* um elemento pode ser parcialmente pertencente a um conjunto, pois dentro da lógica *Fuzzy*, a função de inclusão de um elemento pode ser definida por qualquer valor dentro do intervalo de [0,1].

Um elemento estar ou não estar inserido em grupo na lógica *Fuzzy*, não acontece de forma tão abrupta, sua inclusão ou exclusão dentro de um grupo pode ser feita de forma gradativa, é o chamado grau de pertinência do conjunto *Fuzzy*.

Este grau de pertinência representa a incerteza ou a imprecisão, característica essa que diferencia o sistema *Fuzzy* de outros sistemas convencionais.

Para criar esta inclusão no sistema *Fuzzy* são utilizadas algumas operações do sistema clássico como: UNIÂO, INTERSEÇÂO E COMPLEMENTO.

### 3.3.5. União de grupo fuzzy

No conjunto *Fuzzy*, a união representa o encontro de dois conjuntos *Fuzzy*, ou seja, é no mínimo o valor de pertinência do maior dos dois conjuntos.

Por exemplo, se dois conjuntos *Fuzzy* A e B possuem cada um quatro elementos, a união dos dois conjuntos representada por $A \cup B$ será um conjunto representado pelo valor máximo comparando elemento por elemento dos dois conjuntos.:

### 3.3.6. Complemento de grupos fuzzy

Para o conjunto *Fuzzy*, a operação em conjunto complemento, corresponde a todos os elementos que não corresponde ao conjunto. Análogo à álgebra booleana, a operação complemento em *Fuzzy*, corresponde à operação NOT booleana.

### 3.3.7. Interseção de grupos Fuzzy

A interseção de grupos *Fuzzy* representa a parte comum entre dois ou mais grupos, ou seja, é no mínimo menor que qualquer dos conjuntos individuais. O grau de pertinência é o mínimo ou o menor entre os dois grupos, comparando-se elemento por elemento.

Relacionando à lógica booleana, a interseção *Fuzzy* representa na lógica booleana a operação AND.

### 3.3.8. Regras FUZZY

O processo de regras *Fuzzy* é baseado em um conjunto de regras com expressões do tipo "*Se-Então*" muito comum na maioria das linguagens de programação. Estas regras aceitam apenas afirmações das quais é possível dizer se são verdadeiras ou falsas. Estas regras ditam quais ações o controlador *Fuzzy* irá tomar. A parte "Se" da regra diz qual a condição da ação, enquanto a parte "Então" diz qual ação será tomada. O sistema de regras *Fuzzy* foi assim definido devido à simplicidade de aplicação e entendimento.

Como o processo *Fuzzy* envolve várias regras para uma mesma variável, o conjunto de regras interpretam o valor de entrada e atribuem um valor a uma variável. de pertinência para o vetor de saída.

### 3.3.9. Defuzzificação

A etapa de Defuzzyficação realiza o processo inverso da Fuzzyficação, ou seja, transforma os valores de pertinências devidamente Fuzzyficados em uma grandeza absoluta que pode ser compreendida pelo sistema. Algumas vezes, após o processo de Defuzzyficação, é preciso normalizar a resposta para que ela possa ser compreendida pelo sistema . (RESIGNET 2011)

Existem diversos modos de se realizar a Defuzzyficação e a escolha assim como a do processo de Fuzzyficação leva em conta o custo computacional, e a precisão de saída.

Os métodos mais comuns de Defuzzyficação são: Centro de área, Centro de Máximo e Média de máximo.

### 3.3.10. Centro da Área (C-o-A)

Este método calcula o centro de área criado através das variáveis de entrada e utiliza este valor como saída para o controle *Fuzzy*. Este método se torna falho quando duas funções de pertinência não se encontram.

### 3.3.11. Centro de Máximo (C-o-M)

Este método fornece uma saída representada pela média ponderada dos valores máximos das entradas do controlador *fuzzy*, ou seja, a área do conjunto todo não interfere na saída.

### 3.3.12. Média do Máximo (M-o-M)

Trata-se do método mais simples de Defuzzyficação, pois a saída representa apenas a média aritmética dos valores máximos dos valores de entrada.

## 3.4. FPGA (Field Programmable Gate Array)

FPGA, a abreviação de *Field Programmable Gate Array*, trata-se de um chip reprogramável que trabalha em nível de portas lógicas. Constitui-se de um circuito integrado programável em campo, ou seja, um hardware que pode ter suas funções lógicas configuradas e reconfiguradas pelo usuário conforme o seu desejo, respeitando as limitações do chip utilizado, diferentemente de um microcontrolador convencional, que vem com suas funções lógicas de fábrica e o usuário deve se adaptar a elas, podendo alterar apenas a sua programação.(CURVELLO, 2017)

A base do funcionamento de FPGA são seus elementos lógicos, que são um agrupamento de portas lógicas. Estas portas lógicas organizadas podem realizar uma infinidade de operações e até armazenar dados, fazendo a função de uma memória.

O FPGA é dotada da capacidade de organizar grandes circuitos lógicos dentro de seus blocos internos. Estes blocos são interligados durante a compilação de acordo com o código desenvolvido, criando assim um hardware totalmente dedicado.

Os blocos dentro da FPGA podem ser organizados de forma a serem totalmente independentes, ou um bloco pode enviar após todo um processamento, apenas um elemento de informação a outro bloco.

Cada elemento lógico desempenha uma função lógica dentro da lógica booleana, podendo ser do tipo AND, OR, XOR, NOT etc. Quanto mais elementos lógicos forem agrupados, mais complexo será o seu hardware, podendo chegar a uma CPU, GPU, RAM, entre outros blocos, dependendo do seu arranjo.

### 3.4.1. Linguagem VHDL (VHSIC Hardware Description Language).

Um FPGA possui uma linguagem um pouco diferente da linguagem de programação convencional, a programação de uma FPGA na verdade é um processo de síntese, pois as linhas de código darão forma a um circuito computacional.

Para facilitar a criação de hardware, o exército americano na década de 80 criou a linguagem VHDL (VHSIC Hardware Description Language). Originalmente, está linguagem surgiu para descrever o comportamento das ASICs, mas que devido ao sucesso foi logo exposta ao domínio público para desenvolvimento e posterior padronização pela IEEE (Institute of Electrical and *Electronica* Engineers) em 1987, amentando drasticamente sua popularização e uso. (NELSON, 2010)

Códigos VHDL são semelhantes a códigos do tipo C/C++, mas descreve o comportamento elementos lógicos comportamentos este que é inferido a FPGA durante a compilação, suas linhas de códigos dizem ao compilador como agrupar os elementos lógicos disponíveis dentro da FPGA.

A linguagem VHDL é muito utilizada em projetos, pois seu custo de produção e tempo de desenvolvimento tendem a ser muito menor se comparado à produção de circuitos dedicados.

# 4 METODOLOGIA

## 4.1 Simulação

Para se chegar a um sistema de controle *Fuzzy* mais adequado, foram realizadas simulações no software *Matlab*.

O software *Matlab* trabalha com processamento em equacionamento matricial, e sua ferramenta *Simulink* possui uma interface facilitada que permite trabalhar com uma linguagem de alto nível em interface em bloco, e com uma linguagem derivada da linguagem C/C++ através de seu prompt de comando. (SALVADOR, 2016)

O *Matlab* possui diversas ferramentas, dentre as quais, serão utilizadas três, o *Simulink*, *Fuzzy Logic Designer* e o *System Identification Toolbox*, pois estas ferramentas possibilitam construir a simulação de um controlador mais adequado às necessidades e também permite comparar a resposta do controlador *Fuzzy* a um controlador PID.

### 4.1.2. Função transferência do servo motor

Para a criação do sistema simulado, foi necessário obter a função de transferência do servomotor. Esta função de transferência foi obtida através de dados coletados do próprio servomotor através de testes e com auxílio da ferramenta *System Identification Toolbox*.

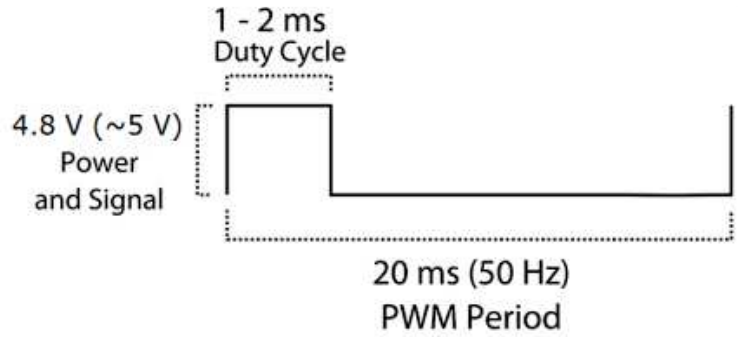
A ferramenta do *Matlab*, *System Identification Toolbox* pode ser acessada digitando o termo Ident na janela de comando do *Matlab*, está ferramenta possibilita construir modelos matemáticos para sistemas dinâmicos utilizando-se de valores de entrada e saída. A ferramenta gera uma interpolação de pontos criando uma equação geradora aproximada. Para encontrar a função transferência do servo

motor utilizado, a variável de entrada utilizada foi o tempo e de saída foi à posição angular do servo motor.

A coleta dos dados de entrada e saída, necessários para a ferramenta *System Identification Toolbox*, foi feita utilizado de um kit Arduino™. Decidiu-se utilizar de um kit Arduino™ pela sua simplicidade de programação, baixo custo e economia de tempo. O kit possui um conversor AD integrado, e permite trabalhar com valores matemáticos em ponto flutuante de forma nativa em suas bibliotecas, possui uma interface com o computador muito intuitiva e é de fácil troca de dados através da porta serial USB do computador.

O servomotor utilizado para os testes, é do modelo sg90. Este tipo de servomotor tem seu posicionamento controlado por um PWM com uma frequência 50 Hz (1 pulso a cada 20 ms) figura 4. Neste servo a posição inicial de zero graus é encontrada fornecendo uma largura de pulso de 1ms, e seu ângulo máximo, 180 graus, fornecendo uma largura de pulso de 2 ms, como mostrada na Figura 4, fornecida pelo datasheet do fabricante, ou seja, para se utilizar os 180 graus que o servomotor é capaz de fornecer, o sistema de controle deve trabalhar com uma largura de pulso de 1ms.

**Figura 4 -** PWM de funcionamento para servo motor sg90



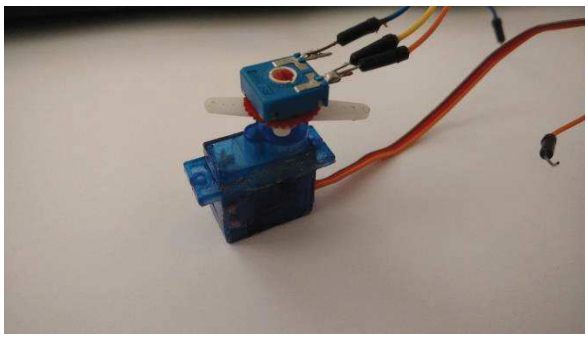

Fonte: < http://akizukidenshi.com/download/ds/towerpro/SG90.pdf>

Outra característica dos servomotores é o seu sistema de controle, que possui um sistema de realimentação que verifica se o valor da posição atual do

servo é realmente o valor solicitado. Esse mecanismo é um sistema de realimentação negativa que através de um potenciômetro acoplado ao eixo do motor compara a tensão deste potenciômetro a tensão fornecida pelo PWM. Esta comparação é feita por um circuito lógico analógico que realiza a conversão da tensão do potenciômetro em tensão do PWM e vice-versa.

O sensor utilizado para a leitura da posição do servo motor foi um potenciômetro acoplado ao servomotor, conforme a Figura 5.

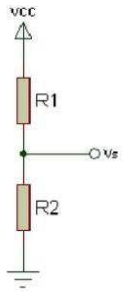
**Figura 5 -** Potenciômetro acoplado ao servo motor



Fonte: Própria (2019)

Como o potenciômetro é construído de forma a gerar um divisor de tensão figura 6, através de alguns cálculos matemáticos, consegue-se transformar o valor de tensão, resultante da divisão de tensão em um valor de posição em graus.

**Figura 6 -** Potenciômetro como um divisor de tensão




Fonte: Própria (2018)

Os potenciômetros lineares possuem taxa linear de variação de resistência em relação ao ângulo de rotação, assim é possível escrever a equação 2, para obter o diferencial do ângulo, equação 3:

$$\frac{R-0}{a-0} = \frac{dR}{da} = K \qquad (2)$$

$$da = \frac{1}{K} dR \qquad (3)$$

Onde:

a: Ângulo de rotação do potenciômetro;

K: Taxa de variação da resistência em função da rotação;

R=R2: Resistência inferior do potenciômetro.

Integrando ambos os lados da equação 3, encontra-se a equação 4:

$$a = \frac{1}{k} R \qquad (4)$$

O valor de R é um valor desconhecido, mas que pode ser calculado considerando o potenciômetro como um divisor de tensão. Aplicando a lei de Ohm onde U=R.I, é possível relacionar as tensões Vcc e Vs, com a resistência nominal do potenciômetro (Rpot) e com a resistência inferior do potenciômetro (R).

Aplicando a lei de Ohm ao potenciômetro é possível observar a seguinte razão, conforme equação 5.

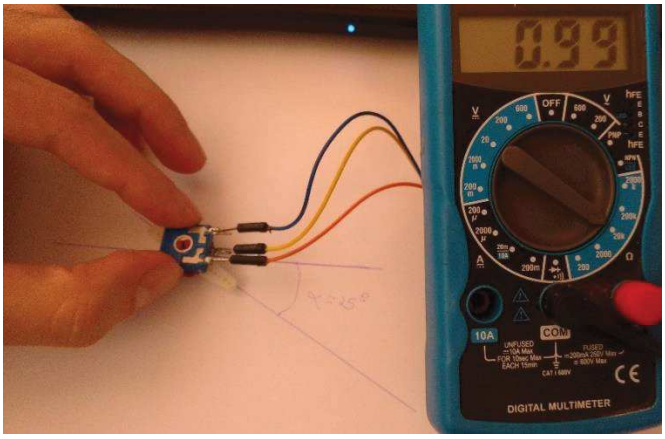
$$\frac{Vs=R2.i}{Vcc=Rpot.i} \quad (5)$$

A relação entre o valor lido no ADC (ValADC) e o máximo valor que o ADC pode retornar ([($2^n$)-1]), onde n é a resolução do conversor ADC a ser utilizado, neste caso 10bits provenientes do conversor do modulo Arduino. A relação obtida anteriormente resulta na Equação 6:

$$R = \frac{Rpot}{K.|(2^n)-1|}.(ValADC) \quad (6)$$

Para encontrar o valor de K é necessário realizar alguns testes práticos com o potenciômetro figura 7, medindo a variação de resistência em relação à variação do ângulo. Utilizando um multímetro posiciona-se o potenciômetro na posição de resistência zero, marcando em uma folha uma linha de base. Após a movimentação do potenciômetro para a posição de resistência 1kΩ, a nova posição foi marcada na folha. Medindo com um transferidor foi encontrada a variação de 1kΩ para cada 25 graus de rotação.

**Figura 7 -** Teste de variação da resistência x ângulo




Fonte: Própria (2019)

Utilizando as informações coletadas e inserindo-as na equação 4, encontra-se o valor de K.

$$\frac{\Delta\Omega}{\Delta\theta} = K \qquad (7)$$

$$\frac{1000-0}{25-0} = 40 \qquad (8)$$

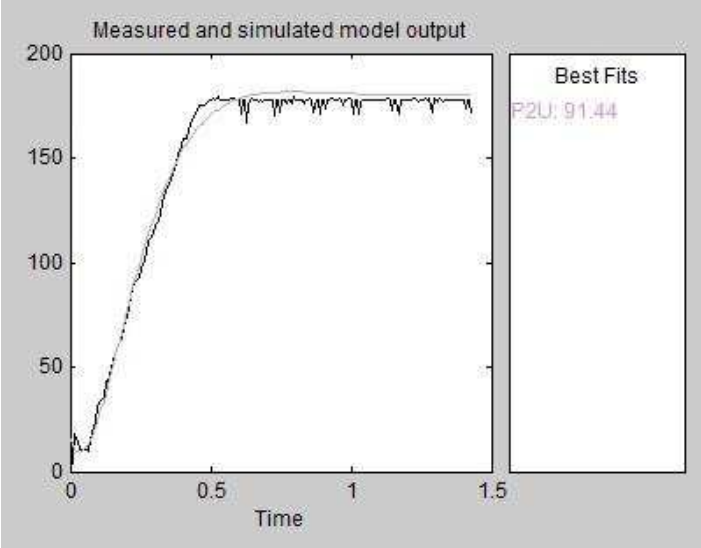
Adicionando K à equação 8 resta saber o valor de Rpot. O potenciômetro usado tem valor nominal de 10 kΩ, porém para maior precisão foi medido seu valor real, encontrando assim o valor de 10380Ω.

Para o valor de n da equação 6, foi utilizado um conversor ADC do modelo ADC0808CCN. Este conversor possui uma resolução de 8 bits. Adicionando todos os valores a equação 6 e adicionando esta equação a programação do Arduino ™, foi possível medir a variação da posição do servomotor em função do tempo.

O tempo de amostragem utilizado na programação foi de 0,002 s, ou seja, foi obtido e salvo o valor da posição do servo a cada 0,002 segundos, com os valores obtidos, foi criada uma curva da posição em relação ao tempo. Está curva foi então

interpolada pela ferramenta *System Identification Toolbox* do *Matlab* que retornou uma equação geradora com 91,44% de precisão em interpolação, como mostra a Figura 8.

**Figura 8 -** Gráficos de interpolação




Fonte: Própria (2019)

### 4.1.3. Controlador Fuzzy no Matlab

Após encontrar a função transferência do servomotor, é possível fazer os testes do controlador *Fuzzy* em ambiente virtual. O *Matlab* possui uma ferramenta para modelagem *Fuzzy* chamada *Fuzzy Logic Designer*. Esta ferramenta pode ser acessada digitando o termo “Fuzzy” na janela de comando do *Matlab*.

O controlador *Fuzzy* implementado utiliza dois parâmetros de entrada, o erro da posição (*erro*) e a velocidade instantânea (*Velo*) do servo motor, e apenas um parâmetro de saída que é o incremento da posição (*posição*).

O erro da posição é encontrado pelo sensor, que no caso é o potenciômetro acoplado ao eixo do motor que envia um valor de tensão variável ao conversor ADC.

A diferença entre a posição desejada pelo operador e a posição em que o servo motor está no momento medido é o parâmetro principal de entrada para o controlador *Fuzzy*.

O parâmetro secundário de entrada para o controlador é a velocidade do servo motor. Esta velocidade é encontrada pela equação 10. Estes dois parâmetros foram escolhidos, pois o controlador deve encontrar a posição ideal do servo motor, obedecendo a um controle de velocidade, pois o objetivo do controlador é iniciar e terminar o movimento do servo motor de forma suave, e ter um movimento intermediário acelerado.

### 4.1.4. Função de pertinência erro

A variável de entrada erro tem uma variação possível de -180 até 180 graus, referentes ao menor e maior erro possível respectivamente que o sistema pode atingir.

O tipo função de pertinência escolhida para o controlador *Fuzzy* foi do tipo triangular. Esta função de pertinência possui algumas vantagens em relação ao uso de outras curvas. (AFUSO, 2012)

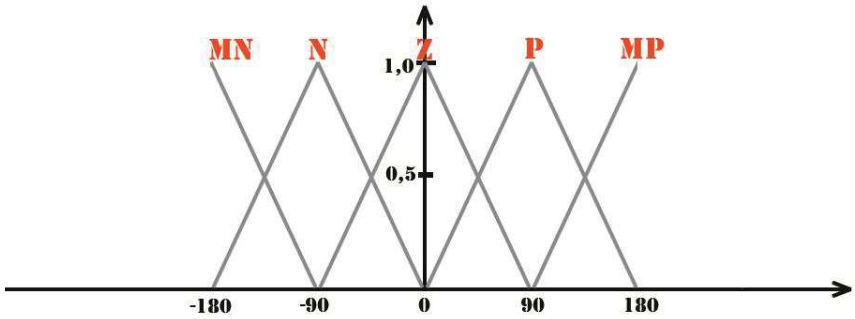
- São simples de ser implementadas;
- Apresentam um custo computacional relativamente baixo;
- São muito eficientes.

Para o controlador *Fuzzy*, foram definidos cinco conjuntos figura 9, ou termos linguísticos, que representam as proposições *fuzzy* para a variável *erro, sendo elas:*

- O erro é muito negativo (MN),
- O erro é negativo (N),
- O erro é zero (Z),
- O erro é positivo (P), e

- O erro é muito positivo (MP).

**Figura 9 -** Conjuntos linguísticos para a variável de entrada erro.




Fonte: Própria (2019)

O grau de pertinência de cada termo linguístico foi definido através de testes práticos realizados pelo controlador no ambiente de simulação. A utilização de quantidades impares de termos linguísticos, facilita a implementação matemática do sistema.

### 4.1.5. Função de pertinência Velocidade

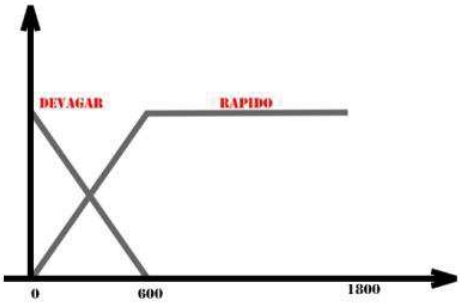
A variável de entrada *Velocidade* representa a velocidade instantânea que o servo motor está desempenhando. Para gerar um movimento suave e evitar alterações bruscas de velocidade, foi decidido que quando o servo motor está em baixa velocidade, o aumento da aceleração será baixo, quando a aceleração do servo motor for considerada pelo operador alta, o aumento da aceleração será alto.

Através de alguns cálculos e testes práticos, a velocidade angular do servomotor pode assumir um intervalo de 0 até 1800 ω dentro do pior caso possível de medição do sistema em um tempo de 0,01s.

Através de testes dentro do ambiente de simulação, foi escolhido utilizar dois tipos de funções de pertinência, uma do tipo triangular e outra do tipo trapezoidal. Os termos linguísticos escolhidos foram figura 10:

- Devagar, quando a velocidade é baixa (D), e
- Rápido, quando a velocidade é alta (R).

**Figura 10 -** Conjuntos linguísticos para a variável de entrada Velocidade



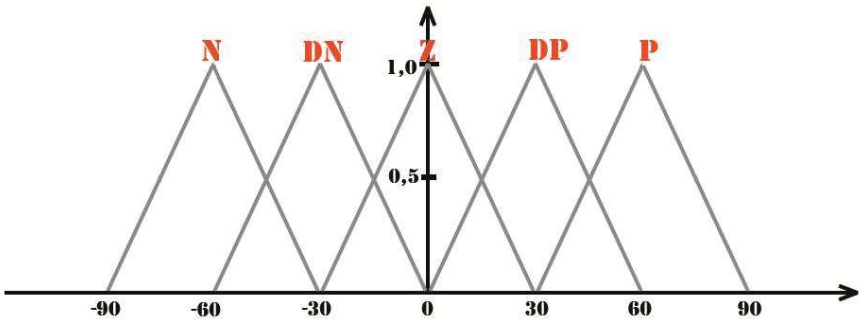

Fonte: Própria (2019)

### 4.1.6. Função de pertinência posição

Para facilitar a implementação em VHDL sem utilizar ponto flutuante, a variável de saída se encontra entre os extremos de -100 até 100, assim a saída ficará representada por números inteiros, o que permite uma fácil implementação sem perder precisão no controlador *Fuzzy*. As variáveis linguísticas escolhidas foram figura 11:

- Variação da posição é negativa (N),
- Variação da posição é devagar e negativa (DN),
- Variação da posição é Zero (Z),
- Variação da posição é devagar e positiva (DP), e
- Variação da posição é positiva (P).

**Figura 11 -** Conjuntos linguísticos para a variável de Saída Posição.




Fonte: Própria (2019)

### 4.1.7. Regras de Defuzzyficação

No processo de Defuzzyficação, uma grande vantagem do controlador *fuzzy* é a sua fácil implementação dos conhecimentos e desejos do operador ao controlador. Como dito, o movimento do servomotor deve ser iniciado de forma suave, assim que o servo motor assumir uma velocidade considerável, o movimento é então acelerado ao máximo, e ao fim do movimento o servo motor deve apresentar uma desaceleração suave.

Assim, utilizando das especificações exigidas pelo operador é possível criar todas as regras do tipo "*Se-Então*". Para realizar este movimento, as seguintes regras de Defuzzificação foram escolhidas, conforme a tabela 1.

**Tabela 1:** Regras para a tensão de saída posição.

| Velocidade/Erro | **MN** | **N** | **Z** | **P** | **MP** |
|---|---|---|---|---|---|
| **Devagar** | DN | DN | Z | DP | DP |
| **Rapido** | N | N | Z | P | P |

Fonte: Própria (2019)

As intersecçoes das linhas com as colunas, na Tabela 1, fornecem o valor desejado de saída, dentro das possiveis combinaçoes de entrada.

As regras são todas do tipo “E” que fornece um valor desejado de saída, para as possíveis situações de entrada. Por exemplo, se a entrada do erro foi muito negativo (MN) e a velocidade foi Devagar (D), o servo está iniciando o seu movimento, então o valor de incremento da saída será devagar negativa (DN). Quando a entrada do erro for negativa (N) e a velocidade for rápida (R), assim o servo motor já está em seu período intermediário de velocidade, então o incremento da posição da saída será Negativa (N).

### 4.1.8. Dados da simulação

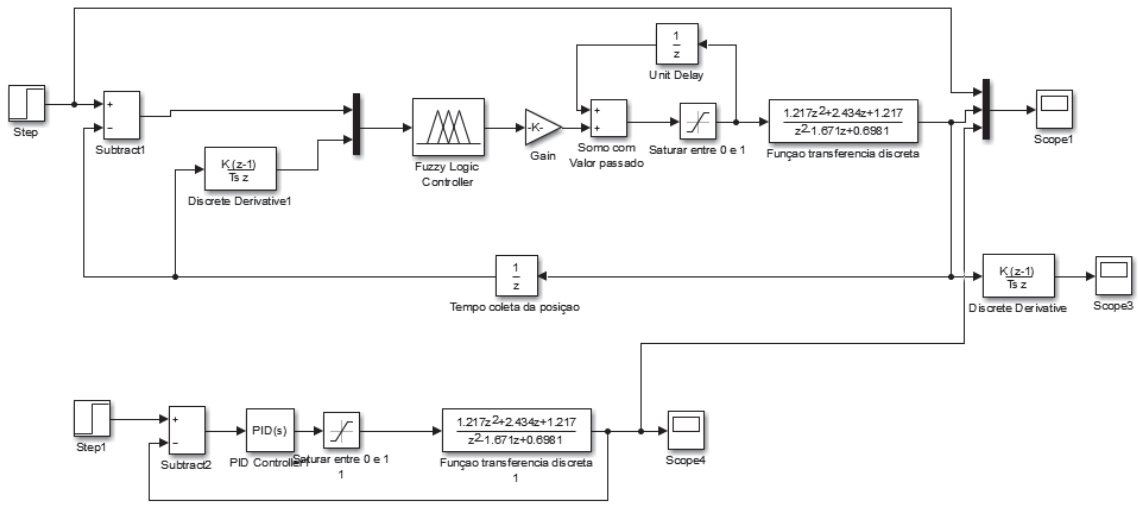
Para realizar a simulação, foi utilizada a ferramenta S*imulink*. O S*imulink* é uma ferramenta para modelagem, simulação e análise de sistemas dinâmicos. Sua interface primária é uma ferramenta de diagramação gráfica por blocos e bibliotecas customizáveis. O software oferece integração com o resto do ambiente do *Matlab*, incluindo a ferramenta *Fuzzy Logic Designer* que foi utilizada para gerar o controlador *Fuzzy*. O *Simulink* é amplamente usado em teoria de controle e processamento digital de sinais para projeto e simulação multi-domínios.

Cada parte do sistema a ser simulado é representada por um bloco dentro do *Simulink*. O controlador *Fuzzy*, que foi definido anteriormente, é simulado dentro do *Simulink* através do bloco dedicado *Fuzzy* chamado *Fuzzy logic controller*. Este bloco faz a integração do *Simulink* com a ferramenta *Fuzzy*.

O sistema a ser criado pode ser integralmente simulado na ferramenta *Simulink*, entradas, realimentação e controlador *Fuzzy*. Alguns atrasos foram adicionados à realimentação, para representar o tempo de amostragem que posteriormente serão definidos na implementação em software. O mesmo acontece com a saída do controlador *Fuzzy*, que foi ligado a um ganho que também representa o tempo de amostragem que posteriormente será definido.

Na simulação também foi adicionado um controlador PID simples, que tem por único objetivo, a comparação da resposta do controlador *Fuzzy* com a resposta do controlador PID. O sistema completo está representado na figura 12.

**Figura 12 -** Sistema completo do servo motor no Simulink

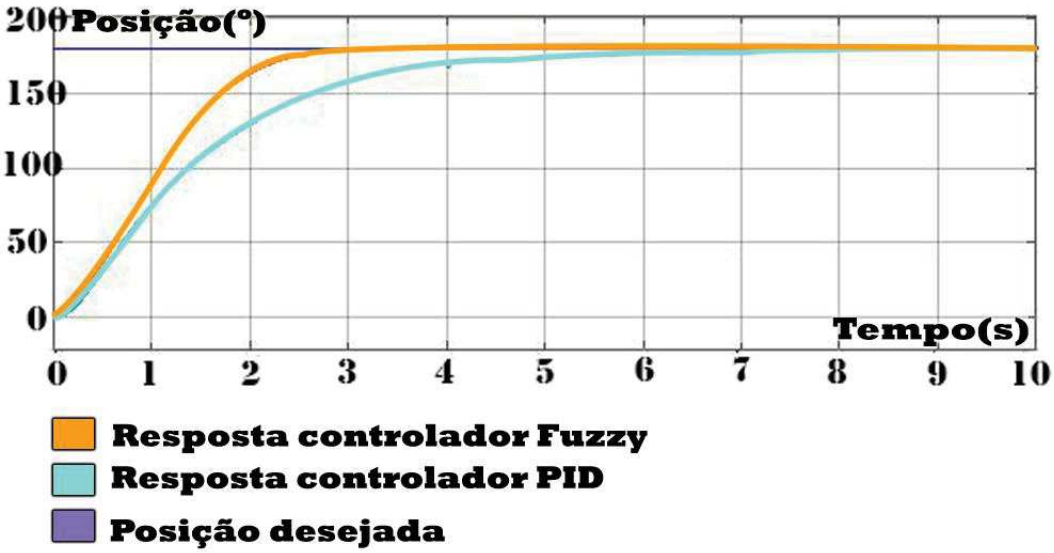



Fonte: Fonte: Própria (2019)

A simulação apresentou um resultado muito satisfatório, pois a curva que representa a movimentação do servo motor apresentou exatamente o movimento desejado e sua resposta foi mais rápida se comparado ao controlador PID simulado.

Na figura 13 a curva em vermelho representa a resposta do controlador *Fuzzy* implementado, a curva em azul a resposta do controlador PID, a linha azul marinho, é a linha de base, que mostra a posição desejada para o sistema, Que neste caso foi de 180˚.

**Figura 13 - Resposta Fuzzy X PID (Gráfico ângulo pelo tempo)**




Fonte: Fonte: Própria (2019)

## 4.2. Programação VHDL

O Hardware escolhido para implementar o sistema de controle foi um kit de desenvolvimento FPGA do modelo Altera Cyclone IV. Trata-se de uma placa de desenvolvimento com diversos periféricos disponíveis. O kit faz uso do ambiente de desenvolvimento e programação da própria ALTERA – Quartus II/Prime (Versão WEB Edition, que pode ser baixada gratuitamente no site da ALTERA), o Quartus II utiliza a linguagem de programação VHDL.

O kit, além de possuir uma grande quantidade de elementos lógicos, fornece um barramento de 32 pinos de I/O e 10 chaves seletoras que serão utilizados no projeto.

O ambiente de desenvolvimento da placa, o Quartus II, fornece algumas bibliotecas matemáticas para a programação em VHDL, porém não possui biblioteca de cálculos com ponto flutuante. Para contornar esta falta e evitar utilizar grande quantidade de elementos lógicos, durante a realização de cálculos matemáticos,

estes passam por um processo de potenciação dos termos evitando assim o uso de ponto flutuante.

O Quartus II também fornece uma interface gráfica em blocos, facilitando a criação de blocos lógicos dentro do chip FPGA, gerando a programação e a resposta independente para cada bloco, proporcionando a programação em paralelo, diferente da programação de um microcontrolador comum, que realiza suas funções lógicas em forma sequencial.

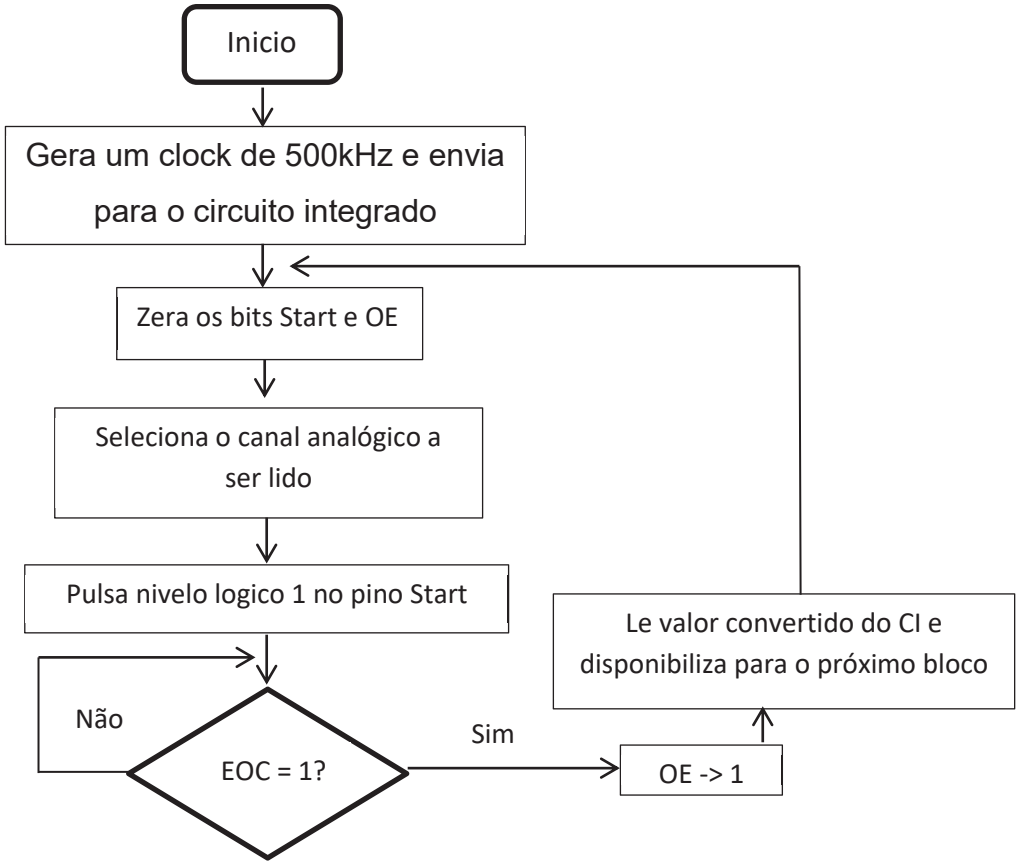
### 4.2.1 Bloco_de_controle

Todo o sistema se inicia pela leitura da posição do servo motor. Está leitura é proveniente do valor de tensão variável do potenciômetro acoplado ao servomotor. Como está tensão é uma grandeza analógica, ela precisa ser convertida em formato digital para que possa ser interpretada pelo kit de desenvolvimento FPGA.

Para realizar a conversão entre as grandezas, foi utilizado um conversor AD do modelo ADC0808CCN de 28 pinos produzido pela *Texas Instruments*, que é um circuito integrado bastante comum e de valor acessível. Este circuito integrado possui 8 entradas analógicas multiplexadas, e apresenta uma saída digital com 8 bits de resolução. Sua comunicação com o kit FPGA é feita através do barramento de I/O de 32 pinos. Para o funcionamento do Circuito integrado, é necessário uma série de comandos externos, estes comandos são realizados pelo bloco de entrada do sistema chamado controle.

O bloco_de_controle envia ao circuito integrado o sinal de start, cria um PWM de 500 Khz necessário para o funcionamento do circuito integrado e envia ao final da conversão o nível lógico alto para o pino *Output enable*, que libera o valor convertido do circuito integrado para a leitura, o funcionamento lógico do bloco de controle fica melhor representado no fluxograma da figura 14, o código em VHDL completo do bloco_de_controle está disposto no Apêndice A. O funcionamento pode ser melhor compreendido através da figura 14

**Figura 14 - Fluxograma controle do conversor ADC**




Fonte: Própria (2018)

### 4.2.2. Bloco Chave

O bloco Chave também se trata de um bloco de entrada comum, sua função é bastante simples, através de duas chaves seletoras disponíveis no kit da Altera, são selecionados ângulos desejados para os servomotores, o código VHDL referente ao bloco Chave está Disposto no apêndice B.

No projeto feito, o valor desejado é repassado igualmente para os três servosmotores utilizados.

### 4.2.3. Bloco ValorEmAngulo

O valor de tensão fornecido pelo potenciômetro e convertido pelo conversor AD, ainda não está pronto para representar o valor em ângulo do potenciômetro, é preciso multiplicar o valor binário fornecido pelo conversor AD pela constante que pode ser encontrada utilizando as especificações do potenciômetro na Equação 6.

Para evitar o ponto flutuante, o valor da constante foi multiplicado por 1000 antes de ser multiplicado pelo valor binário da tensão, após a multiplicação, o valor é então dividido por 1000, resultado em um valor inteiro de ângulo. A resolução de 8 bits do conversor AD com o potenciômetro que trabalha com 270 graus, resulta em valores possíveis de leitura de ângulo que são muito próximos de valores inteiros, então a não utilização de ponto flutuante no programa para o ângulo, não gera problemas, o código VHDL referente ao bloco ValorEmAngulo está disposto no apêndice C.

### 4.2.4. Bloco Erro

O bloco erro é a principal entrada do controlador *Fuzzy*, este bloco fornece a diferença angular entre a posição atual do servo motor, que foi devidamente fornecida pelo bloco ValorEmAngulo, e para posição desejada fornecida pelo bloco Chave, o código VHDL referente ao Bloco_erro está disposto no Apêndice E.

O bloco erro não é o primeiro passo do processo de Fuzzyficação, este bloco apenas fornece um valor compatível de entrada para o processo *Fuzzy*.

### 4.2.5. Bloco VelocidadeAngular

O bloco VelocidadeAngular gera a entrada secundária do sistema *Fuzzy*, assim como o blocoErro. O bloco VelocidadeAngular se limita a gerar a entrada do sistema *Fuzzy*, ainda não iniciando o processo de Fuzzyficação.

A velocidade angular é a derivação da posição em função do tempo Equação 9.

$$\omega = \lim_{\Delta t \to 0} \to \frac{\Delta \theta}{\Delta t} \quad (9)$$

Por se tratar de um sistema digital, o bloco VelocidadeAngular trabalha com um determinado período de amostragem, este período de amostragem é gerenciado pelo bloco CLKdividido.Definido em 0,1s, o período de amostragem se torna suficiente para se fazer a leitura dos três servos motores, e tempo para que o servo conseguir executar alguma variação no movimento mensurável. O bloco então encontra a velocidade angular através da equação da velocidade média equação 10 para um curto período de tempo, neste caso 0,1s.

$$\omega m = \frac{(\theta f - \theta i)}{td - ti} \quad (10)$$

Para se registar o valor inicial, o bloco inicia suas funções no primeiro ciclo de Clock, apenas registrando o valor da posição do servo. A partir do segundo ciclo Clock, o bloco começa a realizar suas atividades matemáticas e ao final registra sua nova posição inicial para o próximo ciclo de clock. Para evitar o ponto flutuante, a divisão por 0,1 que seria a variação do tempo foi substituída por uma multiplicação

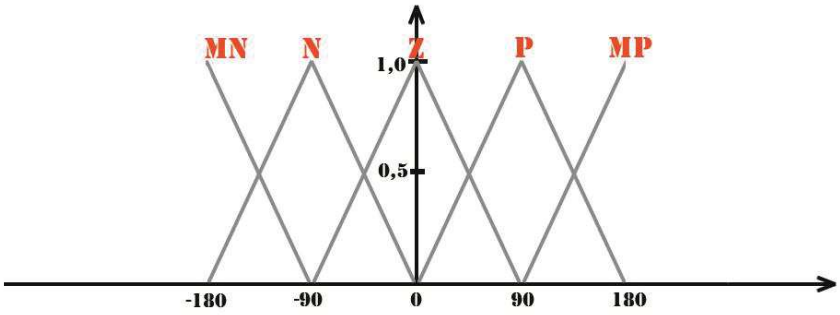
por 10 no numerador. O código VHDL referente ao bloco VelocidadeAngular está disposto no Apêndice E.

### 4.2.6. Bloco FuzzyErro

O bloco FuzzyErro é o primeiro do processo de Fuzzyficação. O bloco transforma os dados de entrada proveniente do bloco Erro em um conjunto *Fuzzy* correspondente.

Como as funções de pertinências escolhidas foram do tipo triangular figura 15 o processo de Fuzzyficação encontra através do valor de entrada, um valor em altura correspondente na função triangular definida. As seguintes funções definem os graus de pertinência da variável de entrada Erro.

**Figura 15 -** Conjuntos linguísticos variável de entrada Erro




Fonte: Própria (2018)

As variáveis linguísticas do tipo triangular podem ser representadas matematicamente da seguinte forma:

$$\text{-180 < erro < -90: } \mu_{erro} = \frac{erro\ -(180)}{90} negativo + \frac{-90-erro}{erro} muito\ negativo \quad (11)$$

$$\text{-90 < erro < 0 : } \mu_{erro} = \frac{erro\ -(90)}{90} negativo + \frac{-0-erro}{erro} zero \quad (12)$$

$$0 < \text{erro} < 90 : \mu_{erro} = \frac{erro\ -(0)}{90} positiva + \frac{90-erro}{erro} zero \quad (13)$$

$$90 < \text{erro} < 180 : \mu_{erro} = \frac{erro\ -(90)}{90} muito\ positiva + \frac{180-erro}{erro} positiva \quad (14)$$

O bloco além de fornecer o valor Fuzyficado da variável de entrada erro, também envia ao próximo bloco uma variável em caractere que informa em qual variável linguística o sistema foi calculado.

Por exemplo, caso variável de entrada erro = 32

$$\frac{32-0}{90} positiva + \frac{90-32}{90} zero \quad (15)$$

$$0{,}355 positiva + 0{,}644 zero \quad (16)$$

Para evitar o ponto flutuante, o numerador antes da divisão sofre uma multiplicação pelo valor 100, está multiplicação é posteriormente corrigida fora deste bloco. O código VHDL referente ao bloco FuzzyErro está disposto no Apêndice F.

### 4.2.7. Bloco FuzzyVelo

O bloco FuzzyVelo assim como o bloco FuzzyErro fazem parte do processo de Fuzzyficação do controlador, o FuzzyVelo gera o grau de pertinência responsável pela variável velocidade.

Para este controlador, a entrada velocidade possui duas variáveis linguísticas, sendo uma do tipo triangular e outra do tipo trapezoidal.

O bloco é responsável por encontrar o valor respectivo da altura na função triangular e na função trapezoidal, nesta até o valor de entrada 600, acima deste

valor na função trapezoidal, o grau de pertinência da entrada velocidade é 1. O processo matemático de Defuzzyficação desta entrada é dado por:

$$0 < velo < 600 \quad \mu_{velo} = \frac{velo-(0)}{600} rapida + \frac{600-velo}{velo} devagar \quad (17)$$

$$Velo > 600 \quad \mu_{velo} = 1 \quad (18)$$

Por exemplo, para uma entrada de velocidade no valor de 340

$$\frac{340}{600} rapida + \frac{600-340}{600} devagar \quad (19)$$

$$0{,}566 rapida + 0{,}433 devagar \quad (20)$$

Para evitar o ponto flutuante neste bloco, o denominador sofre uma multiplicação pelo valor 100, este valor é posteriormente corrigido em um bloco externo a este. O código VHDL referente ao bloco FuzzyVelo está disposto no Apêndice G.

### 4.2.8. Bloco Defuzzy

O bloco Defuzzy é responsável pelas regras de Defuzzyficação, pelas operações do conjunto *Fuzzy* e pelo processo de Defuzzyficação.

O processo de operação em conjuntos *Fuzzy* utilizado foi o processo de interseção. Bloco seleciona entre as duas entradas(erro e velocidade) a parte comum entre elas. Este processo na pratica, seleciona a interseção entre as duas entradas, gerando uma entrada única para o processo de Defuzzyficação. No bloco, este processo é realizado por um conjunto de cinco funções do tipo “IF”:

Com o novo conjunto criado, a programação pode avançar para as regras de inferência, as possibilidades de saída deste controlador foram forjadas de acordo com a necessidade do sistema, pode-se dizer que as vontades do operador foram traduzidas para um conjunto de regras dez regras do tipo “*Se-Então”*:

- **Se** erro é MN **e** velocidade é rápida **então** posição é N
- **Se** erro é MN **e** velocidade é devagar **então** posição é DN
- **Se** erro é N **e** velocidade é rápida **então** posição é N
- **Se** erro é N **e** velocidade é devagar **então** posição é DN
- **Se** erro é Z **e** velocidade é rápida **então** posição é Z
- **Se** erro é N **e** velocidade é devagar **então** posição é Z
- **Se** erro é P **e** velocidade é rápida **então** posição é P
- **Se** erro é P **e** velocidade é Devagar **então** posição é DP
- **Se** erro é MP **e** velocidade é rápida **então** posição é P
- **Se** erro é MP **e** velocidade é Devagar **então** posição é DP

Os operadores *Fuzzy* na programação são realizados por um conjunto de instruções. Basicamente o grupo de entrada já Fuzzyficado no blocoErro, vem acompanhado do por uma variável de controle chamada Valorcontrole, é através desta variável de controle que utilizando de quatro comandos do tipo “If” o sistema consegue identificar e realizar as regras de inferencia.

O processo de Defuzzyficação é realizado pelo método do centro de máximo, diferente do método do centro de gravidade utilizado pelo *Matlab*, os dois processos resultam em resultados muito semelhantes, mas com custo matemático necessário para o método do centro de máximo é significativamente menor.

No método do centro de máximo, o valor a ser calculado é uma média ponderada de todas as ações de controle local. O valor de saída resultante representa a média feita sobre o grau de pertinência das variáveis. O código VHDL referente ao bloco Defuzzy está disposto no Apêndice H.

Por exemplo, considerando as variáveis de entra Erro e Velocidade:

$$\text{Erro} = 0{,}355positiva + 0{,}644zero \quad (21)$$

$$\text{Velocidade} = 0{,}566rapido + 0{,}433devagar \quad (22)$$

Assim:

**Se** erro = positiva **e** velocidade = rápido **então** posição é positiva

$$m_{P\cap R}(\mu) = \min\left[m_p(\mu);\ m_r(\mu)\right] = \min[0{,}355\ ;\ 0{,}566] = 0{,}355 \qquad (23)$$

**Se** erro = positiva **e** velocidade = devagar **então** posição é Devagar positiva

$$m_{P\cap D}(\mu) = \min\left[m_p(\mu);\ m_D(\mu)\right] = \min[0{,}355\ ;\ 0{,}433] = 0{,}355 \qquad (24)$$

**Se** erro = zero **e** velocidade = rápido **então** posição é zero

$$m_{Z\cap R}(\mu) = \min[m_Z(\mu);\ m_R(\mu)] = \min[0{,}644\ ;\ 0{,}566] = 0{,}566 \qquad (25)$$

**Se** erro = zero **e** velocidade = devagar **então** posição é zero

$$m_{Z\cap D}(\mu) = \min[m_Z(\mu);\ m_D(\mu)] = \min[0{,}644\ ;\ 0{,}433] = 0{,}433 \qquad (26)$$

Fazendo-se a Defuzzyficação pelo método do Centro de Máximo:

$$Posição = \frac{0{,}355(60)+0{,}355(30)+0{,}566(0)+0{,}433(0)}{0{,}355+0{,}355+0{,}566+0{,}433} = 18.69 \qquad (27)$$

### 4.2.9. Bloco PWM

O bloco VHDL PWM fornece a saída que efetivamente vai ser enviada ao servo motor, ou seja, sua tarefa é fornecer ao servo motor uma largura de pulso que coloque o servo motor na posição desejada e quando desejada.

O bloco gera uma PWM com largura de pulso variável, o bloco necessita realizar duas contagens, uma contagem coloca o PWM em alta, esta contagem é

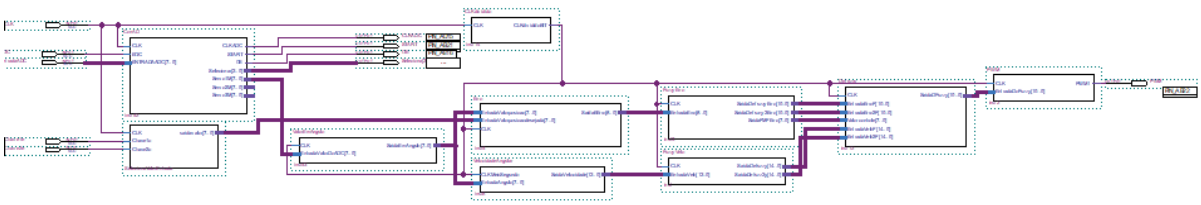
fixa, que no caso são 1000000 contagens, esta contagem cria o período exigido pelo fabricante do servo motor que é de 20Ms. A segunda contagem é a contagem variável, pois esta contagem que efetivamente determina a posição do servo motor, uma função do tipo if determina os limites que a largura de pulso pode ser incrementada e esta mesma função realiza o incremento. Para realizar este incremento, seguindo os testes de tempo de resposta feitos no *Matlab*, o incrementa com o valor passado deve ser feito dentro de um tempo de amostragem de 0,1s, então esta função IF realiza o incremento da largura de pulso com o valor fenecido pelo controle *Fuzzy* a cada 50Mil contagens. O código VHDL referente ao bloco PWM está disposto no Apêndice I.

### 4.2.10. Planta completa

A planta completa dos blocos do circuito FPGA é mostrada na figura 16, cada bloco é executado de forma independente, característica presente graças à estrutura programada que será compilada no hardware FPGA.

Está planta da figura 16 representa o controle de apenas um servo motor, para que possa ser controlado dois ou mais servomotores, é necessário apenas reproduzir toda a planta do controlador em quantos forem os servomotores, está reprodução não afeta a velocidade de resposta do sistema, visto que o compilador reagrupará os circuitos logico de forma independente para cada planta, a quantidade de servomotores que podem ser controlador, fica então limitada apenas as limitações físicas do hardware, ou seja, a quantidade de elementos lógicos que o circuito FPGA utilizado pode fornecer.

**Figura 16** - Planta completa, controlador Fuzzy em FPGA



Fonte: Própria (2019).

# 5. RESULTADOS OBTIDOS

## 5.1. Respostas do programa em ambiente virtual.

O programa de criação de circuitos digitais Quartus II permite através de sua interação com o software *Modelsim,* simular uma resposta de saída do valor dos elementos lógicos programados, utilizando valores de entradas pré-determinados pelo usuário.

A simulação apresenta algumas limitações, principalmente em relação ao tempo máximo de simulação possível, então para que seja possível acompanhar o resultado dos cálculos que o circuito deve apresentar, algumas alterações precisaram ser feitas sobre o código original, estás modificações afetam apenas o tempo de amostragem, não interferindo no valor Deffuzificado que se quer analisar. Basicamente o bloco CLKdividido foi removido dando lugar ao CLK de 50MHz disponível da placa de desenvolvimento.

O tempo de processamento do programa VHDL feito para o controlador Fuzzy é de apenas dois ciclos de Clock, pois utiliza um ciclo no processo de Fuzzyficação, trabalhando com dois blocos em paralelo, e um ciclo para o processo de Defuzzyficação.

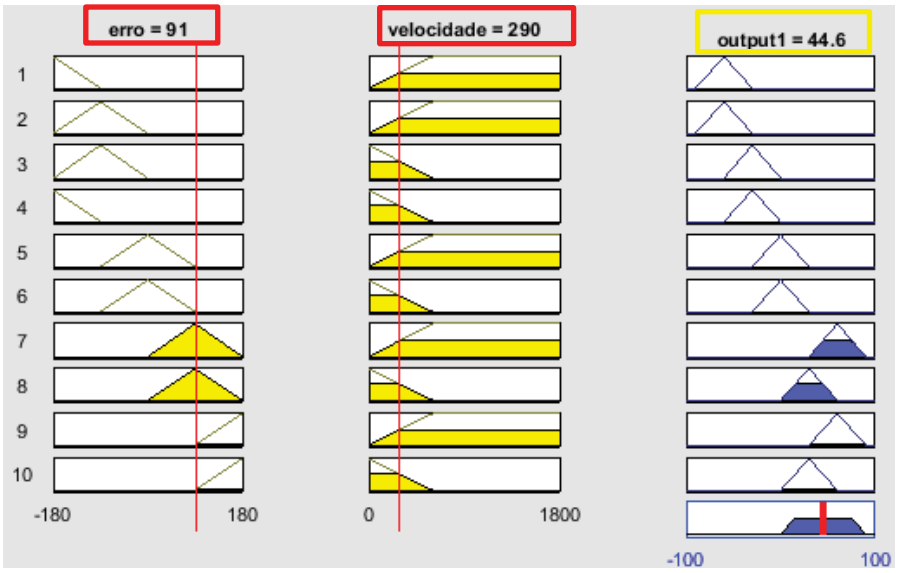
Para verificar a precisão da resposta de saída do controlador *Fuzzy* na simulação fornecida pelo *Modelsin,* esta resposta foi comparada ao valor de saída fornecido pela ferramenta *Fuzzy Logic Designer* do *Matlab,* utilizando-se dos mesmos parâmetros de entrada e as mesmas regras. Como os métodos de Defuzzificação utilizados pelo programa implementado e pelo que o *Matlab* são diferentes, as respostas apresentam uma leve divergência de valores, diferença esta que não gera grandes alterações no funcionamento do controle do servomotor.

Comparando as respostas mostradas nas figuras 16 e 17 em amarelo com entradas mostradas nas figuras 17 e 18 de vermelho onde o Erro = 91 e Velocidade = 290.

Figura 17 - Resposta simulação Modelsin


Fonte: Própria (2019).

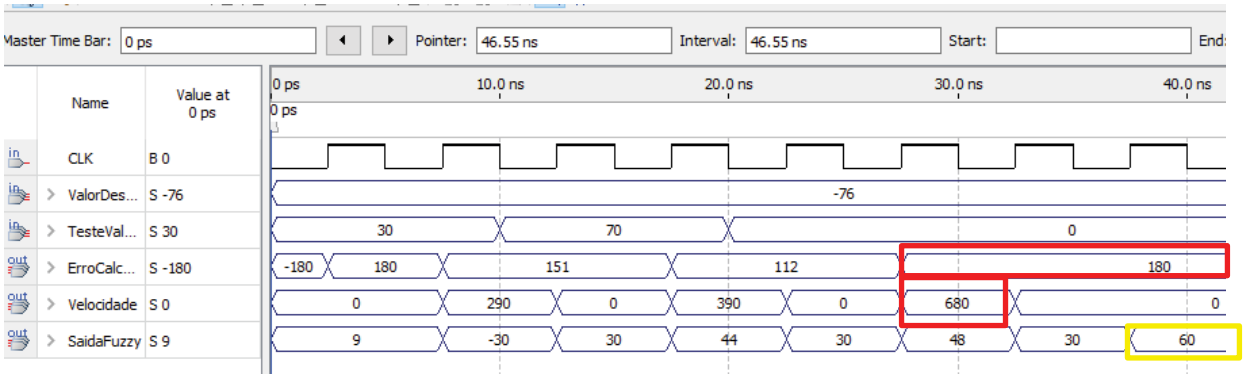
**Figura 18 - Resposta simulação Fuzzy logic controller**




Fonte: Própria (2019).

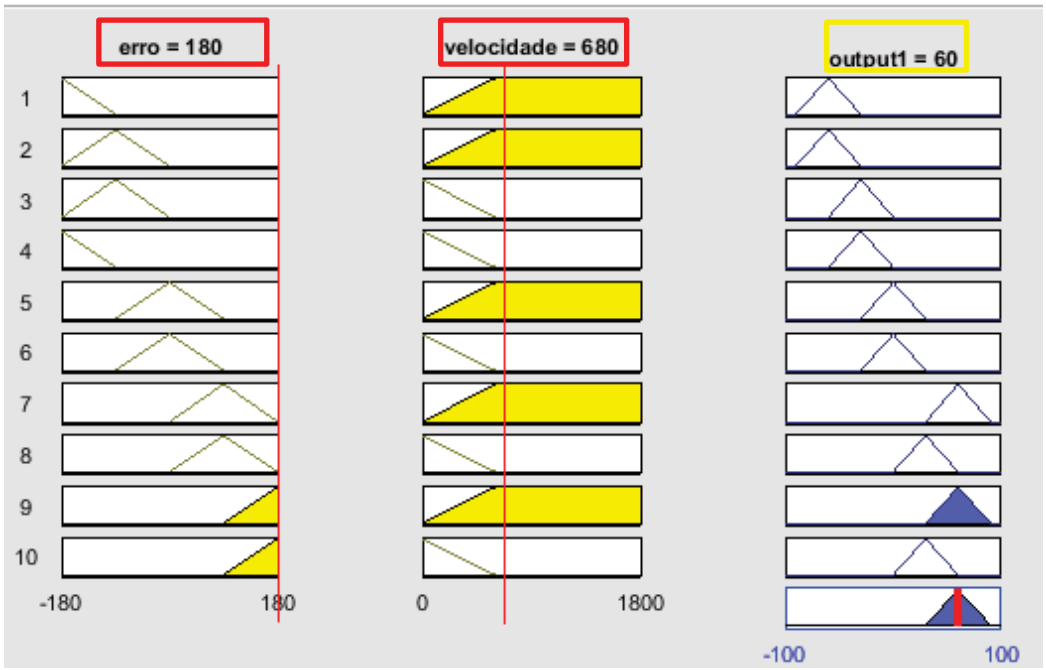
Comparando as respostas mostradas nas figuras 18 e 19 em amarelo com entradas mostradas nas figuras 19 e 20 de vermelho onde o Erro = 180 e Velocidade = 690.

**Figura 19 - Resposta simualçao Fuzzy logic controller**




Fonte: Própria (2019).

**Figura 20 - Resposta simualçao Fuzzy logic controller**



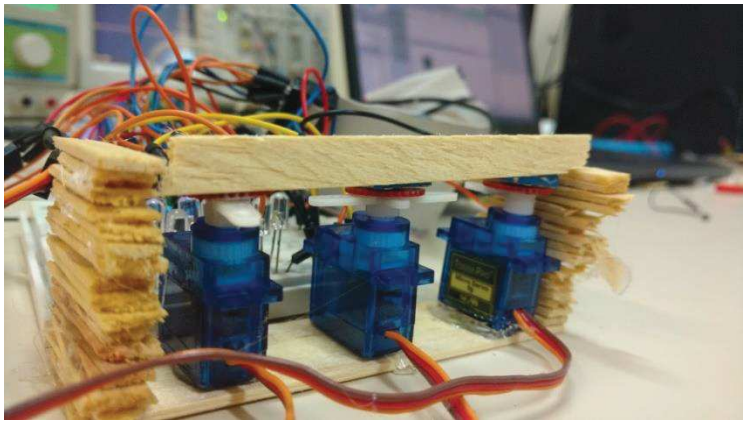

Fonte: Própria (2019).

Como visto o valor Defuzzyficado em ambos os softwares, apresentam um valor de resposta muito semelhantes, mostrando assim a eficácia do código VHDL implementado mesmo sem utilizar ponto flutuante.

## 5.2. Resposta em ambiente físico.

O controlador tem como fundamento controlar os servomotores de forma paralela, então os servomotores que serão utilizados no manipulador foram dispostos em paralelo como mostra a figura 21 de forma a facilitar a compreensão e analise do resultado do controlador *Fuzzy.* A implementação em VHDl facilita a utilização de uma quantidade maior de servomotores, basicamente é preciso apenas replicar o sistema de controle projetado para a quantidade de servos que se deseja, essa multiplicação de blocos independentes em VHDL não altera a velocidade da resposta, tendo em vista que o hardware ira processar as informações de forma paralela em para cada servo motor.

**Figura 21 - Disposição dos servomotores para teste**



Fonte: Própria (2019).

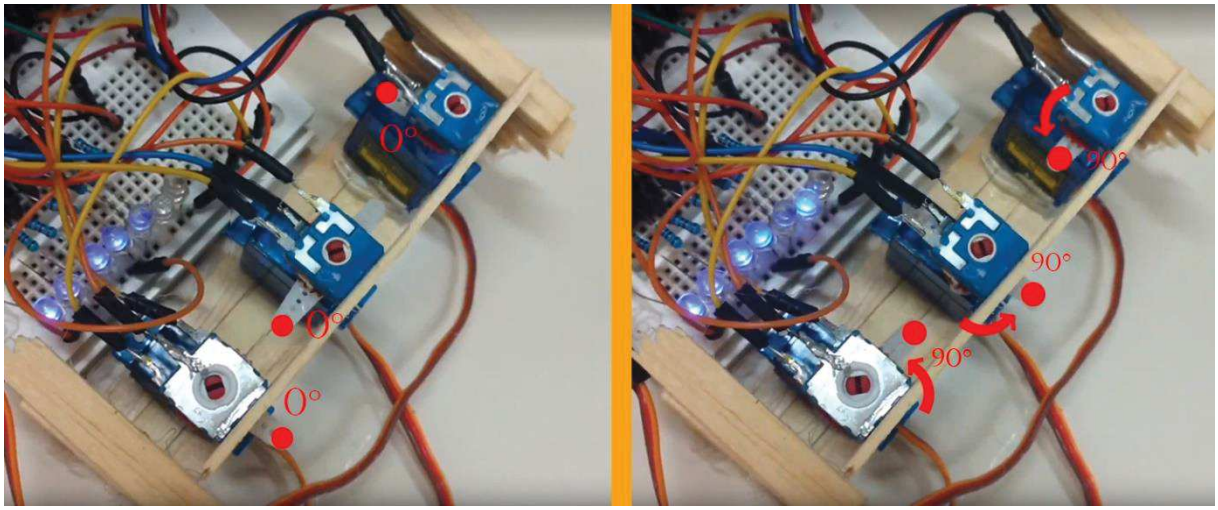
Nesta parte dos resultados, o sistema de controle *Fuzzy* atendeu as suas especificidades, os servomotores apresentaram um movimento suave durante o seu

arranque e durante a sua parada, e seu movimento intermediário foi de forma acelerada.

Como o sistema de conversão analógico digital utilizado é único para os três servos, a resposta inicial de movimentação entre um servo e outro tem um atrasado de 4ms, tempo necessário para que o conversor ADC possa coletar a informação do segundo potenciômetro de forma multiplexada, esse atraso pode ser eliminado utilizando de um conversor ADC independente para cada servo motor, assim o Hardware processaria o controle de cada servo de forma paralela.

Utilizando o controlador em ambiente físico, a resposta em posição e velocidade foi exatamente a desejada, ou seja, os servomotores se movimentaram para a posição desejada demonstrado na figura 22 com o movimento suave, o tempo de movimentação, respeitando as limitações físicas do servomotor, pode ser facilmente alterado apenas alterando o tempo de incremento do valor de Defuzzyficado.

**Figura 22 -** Resposta da posição do servomotor para posição desejada de 90°




Fonte: Própria (2019).

# 6. CONCLUSÃO

Apesar de ainda pouco difundido, o controlador *Fuzzy* neste caso aqui utilizado apresentou um resultado excelente, tanto em ambiente virtual através de simulações, quanto em ambiente físico. Este tipo de controlador, depois de compreendido se demostrou de fácil implementação, ele apresenta um comportamento mais humano sobre o sistema, apesenta um custo computacional muito reduzido se comparado a um controlador PID genérico, e permite inserir facilmente os conhecimentos de algum operador humano experiente sobre o sistema podendo assim reduzir o tempo de desenvolvimento do controlador.

Junto à FPGA, o controlador *Fuzzy*, que graças a seu baixo custo matemático, pode ser facilmente programado em linguagem VHDL mesmo sem utilizar de ponto flutuante para suas operações. Com o uso da FPGA, o sistema de controle se tornou muito flexível em relação à quantidade de elementos a serem controlados, basicamente o projetista deve produzir o controlador para um servomotor, e apenas replicar os blocos lógicos quantos servomotores forem utilizados, seu desempenho se mantém inalterado, tendo em vista que desta forma após compilado o circuito agrupado dentro da FPGA processará os dados de forma paralela, ou seja independente do número de servos a serem controlados, o tempo para que o circuito forneça uma resposta de controle é o mesmo, isto permitiria por exemplo, que apenas uma unidade FPGA, respeitando o seus limites físicos, controle todos os manipuladores robóticos de uma fábrica, sem criar nenhum tipo de atraso de resposta para o controlador.

Conclui-se então que um controlador *Fuzzy* implementado em um circuito FPGA possui grande possibilidade de desenvolvimento em conjunto, se tornando mais competitivo em relação aos métodos convencionais na medida em que aumente o número de elementos a serem controlados.

Esse trabalho deve então ser utilizado como ponto de partida e como fonte de estudo para as técnicas aqui apresentadas. Existe muito a ser melhorado e difundido

sobre está área de desenvolvimento e espero que este trabalho seja útil neste processo.

# REFERÊNCIAS

SOUSA, Rainer. **O fascínio pelos robôs.** 2011. Disponível em: <https://alunosonline.uol.com.br/historia/o-fascinio-pelos-robos.html>. Acesso em: 12 abr. 2016.

AYRES, Marcelo (Ed.). **Conheça a história dos robôs.** 2007. Disponível em: <http://tecnologia.uol.com.br/ultnot/2007/10/01/ult4213u150. jhtm>. Acesso em: 01 maio 2015.

FERRAZ, Rodrigo. **Conheça os 5 países com maior índice de robôs por trabalhador.** 2018. Disponível em: <http://bakertillybr.com.br/conheca-os-5-paises-com-maior-indice-de-robos-por-trabalh>. Acesso em: 14 jun. 2019.

ABAR, Celina (Ed.). **O CONCEITO "FUZZY".** 2004. Disponível em: <http://www.pucsp.br/~logica/Fuzzy.htm>. Acesso em: 06 maio 2015.

LOURENÇO, Luciano... **A História da informática:** Sistemas embarcados e supercomputadores. 2011. Disponível em: <http://www.hardware.com.br/guias/historia-informatica/fpgas.html>. Acesso em: 05 maio 2015.

SILVEIRA, Cristiano Bertulucci. **Servo Motor: Veja como Funciona e Quais os Tipos.** 2017. Disponível em: <https://www.citisystems.com.br/servo-motor/>. Acesso em: 04 fev. 2017

RESIGNET**, UMA INTRODUÇÃO A LÓGICA FUZZY.** São Paulo: Resiget, v. 1, 01 fev. 2011. Semanal. Disponível em: <http://www.logicafuzzy.com.br/wp-content/uploads/2013/04/uma_introducao_a_logica_fuzzy.pdf>. Acesso em: 06 jul. 2016..

SALVADOR, Valter. **O que é o Matlab?** 2016. Disponível em: <https://www.portalgsti.com.br/2016/08/o-que-e-o-matlab.html>. Acesso em: 14 maio 2017.

SANTOS, Eduardo Delinski dos; PIRES, Felipe Escobar. **CONTROLE NEURO-FUZZY PARA UM MANIPULADOR ROBÓTICO.** 2011. 89 f. TCC (Graduação) - Curso de Engenharia Elétrica., Setor de Ciecinas e Tecnlogia, Universidade Federal do ParanÁ, Curitiba, 2011. Cap. 1.

TRIERWEILER, Pedro Antonio. Fuzzyficação. In: WEBER, Leo. **Aplicação da Lógica Fuzzy em Software e Hardware.**Canoas: Daulbra, 2003. Cap. 3. p. 37-39.

# Apêndice A – Código VHDL bloco Bloco_de_controle

```
ENTITY Bloco_de_controle IS
 PORT (  CLK       : IN  bit                         ;
                              EOC       : IN  bit                         ;
                              ENTRADAADC : in  std_logic_vector   (7 downto 0) := "00000000"     ;
                              CLKADC     : OUT bit                        := '0';
                              START      : OUT bit                        := '0';
                              OE         : OUT bit                        := '0';
                              Selecionar : OUT bit_vector     (2 DOWNTO 0)        ;
                              Servo1M    : OUT integer  range 0 TO 255            ;
                              Servo2M    : OUT integer  range 0 TO 255            ;
                              Servo3M    : OUT integer  range 0 TO 255                          );


END ENTITY;

ARCHITECTURE Bloco_de_controle OF Bloco_de_controle IS


BEGIN

process(CLK)
VARIABLE ContadordoADC : INTEGER RANGE       0 TO 2000  :=  0 ;
VARIABLE auxADC       : BIT                      := '0';
VARIABLE ContadordoMUX : INTEGER RANGE       0 TO 60000 :=  0 ;
VARIABLE ValorServo1   : integer range       0 TO 255   :=  0 ;
VARIABLE ValorServo2   : integer range       0 TO 255   :=  0 ;
VARIABLE ValorServo3   : integer range       0 TO 255   :=  0 ;

BEGIN

IF (CLK'EVENT AND CLK = '1') THEN

          ------ CLK do ADC 500khz --
                    ContadordoADC := ContadordoADC +1;
                    ContadordoMUX := ContadordoMUX +1;

                    IF (ContadordoADC = 50) THEN
                    auxADC := not auxADC;
                    ContadordoADC:= 0;
                    END IF;
          ----------------------------

  --------Servo 1 -------------


                    IF (ContadordoMUX > 0 AND ContadordoMUX < 20000) THEN


                              IF (ContadordoMUX > 0 AND ContadordoMUX < 100) THEN
                      Selecionar <= "000";
```

```
            START <='0';
            OE    <='0';
                  end if;

            IF (ContadordoMUX > 100 AND ContadordoMUX < 1100) THEN
                  START <= '1'  ;
            Else

                  START <= '0'  ;
            END IF;
---- LER valor ----

                  IF (EOC /= '0' AND ContadordoMUX > 2100 ) THEN --2100
                  OE    <='1';

                  ValorServo1 := to_integer(unsigned( ENTRADAADC));

                  END IF;


            END IF;


                  IF (ContadordoMUX > 20000 AND ContadordoMUX < 40000) THEN


                  IF (ContadordoMUX > 20000 AND ContadordoMUX < 20100) THEN
            Selecionar <= "001";
            START <='0';
            OE    <='0';
                  end if;

            IF (ContadordoMUX > 20100 AND ContadordoMUX < 21100) THEN
                  START <= '1'  ;
            Else

                  START <= '0'  ;
            END IF;
---- LER valor ----

                  IF (EOC /= '0' AND ContadordoMUX > 22100 ) THEN --2100
                  OE    <='1';

                  ValorServo2 := to_integer(unsigned( ENTRADAADC));

                  END IF;




            END IF;
```

```
                    IF (ContadordoMUX > 40000 AND ContadordoMUX < 60000) THEN


                    IF (ContadordoMUX > 40000 AND ContadordoMUX < 40100) THEN
                  Selecionar <= "010";
                  START <='0';
                  OE    <='0';
                    end if;

                IF (ContadordoMUX > 40100 AND ContadordoMUX < 41100) THEN
                    START <= '1'  ;
                Else

                    START <= '0'  ;
                END IF;
        ---- LER valor ----

                    IF (EOC /= '0' AND ContadordoMUX > 42100 ) THEN --2100
                    OE    <='1';

                    ValorServo3 := to_integer(unsigned( ENTRADAADC));

                    END IF;


                END IF;


END IF;
CLKADC  <= auxADC;
Servo1M <= ValorServo1;
Servo2M <= ValorServo2;
Servo3M <= ValorServo3;
END PROCESS;
END ARCHITECTURE;
```

# **Apêndice B –** Código VHDL bloco ValorDesejado

```
ENTITY chave IS

PORT (   CLK          : IN   BIT                      ;
                       Chave1c       : IN   BIT                      ;
                       Chave2c       : IN   BIT                      ;
                       saidavalor    : OUT  integer RANGE 0 to 180         );


END ENTITY;

ARCHITECTURE chave OF chave IS

BEGIN
process(CLK)

BEGIN

        IF (CLK'EVENT AND CLK = '1') THEN

        IF Chave1c = '0' AND Chave2c = '0' THEN
        saidavalor <= 0;
        end if;
                IF Chave1c = '1' AND Chave2c = '0' THEN
                saidavalor <= 90;
                end if;
                        IF Chave1c = '0' AND Chave2c = '1' THEN
                        saidavalor <= 180;
                        end if;

        END IF;


END PROCESS;
END ARCHITECTURE;
```

# **Apêndice C –** Código VHDL bloco ValorEmAngulo

```
ENTITY ValorEmAngulo IS

PORT (   CLK            : IN bit ;
                        EntradaValorDoADC : IN  integer   RANGE 255 downto 0 := 0  ;
                    SaidaEmAngulo     : OUT integer   RANGE 180 DOWNTO 0 ) ;

END ENTITY;

ARCHITECTURE ValorEmAngulo OF ValorEmAngulo IS

BEGIN
        process(CLK)
        VARIABLE ValorDeP : integer := 9686          ;
        VARIABLE AUXsaida : integer range 0 to 9999;
        VARIABLE AUX2     : integer RANGE 0 TO 9999;


BEGIN
IF (CLK'EVENT AND CLK = '1') THEN

AUXsaida := (EntradaValorDoADC*ValorDeP)/10000;
AUX2     := (EntradaValorDoADC*ValorDeP) REM 10000;


                IF AUX2 >= 6000 THEN  -- APROXIMA CASA DECIMAL --
                AUXsaida := AUXsaida+1;
                END IF;


END IF;

SaidaEmAngulo <= AUXsaida;
END PROCESS;
END ARCHITECTURE;
```

# **Apêndice D –** Código VHDL bloco Erro

```
ENTITY Erro IS

PORT (    EntradaValorposicao        : IN  integer   RANGE 180 downto 0  ;
                                EntradaValorposicaodesejada : IN  integer   RANGE 180 downto 0  ;
                        CLK                    : IN  BIT                    ;
                                SaidaEErro              : OUT integer   RANGE -180 to 180 ) ;

END ENTITY;

ARCHITECTURE Erro OF Erro IS

BEGIN
process(CLK)
BEGIN

        IF (CLK'EVENT AND CLK = '1') THEN

                SaidaEErro <= EntradaValorposicaodesejada - EntradaValorposicao;

        END IF;
END PROCESS;
END ARCHITECTURE;
```

# **Apêndice E –** Código VHDL bloco

```
ENTITY VelocidadeAngular IS

PORT (   CLKMeioSegundo   : IN bit ;
                          EntradaAngulo    : IN  integer    RANGE 180 downto 0  ;
                          SaidaVelocidade  : OUT integer    RANGE -3600 to 3600  ) ;

END ENTITY;

ARCHITECTURE VelocidadeAngular OF VelocidadeAngular IS

BEGIN

process(CLKMeioSegundo)
VARIABLE AUXPrimeirovalor : integer range -180 to 180   ;
VARIABLE contador        : integer RANGE 1 TO 2 := 1;
VARIABLE contadorSaida   : integer  range -3600 to 3600:= 0;


BEGIN
                IF (CLKMeioSegundo'EVENT AND CLKMeioSegundo = '1') THEN


                        IF contador = 1 AND contador < 2 THEN
                        AUXPrimeirovalor := EntradaAngulo;
                        contador := contador+1;
                        END IF;



                                IF contador = 2 THEN
                                --contadorSaida := contadorSaida+1;
                                contadorSaida := ((EntradaAngulo-AUXPrimeirovalor)*10) ;        --- usar
tempo como 0,01 segundo.
                                AUXPrimeirovalor:= EntradaAngulo;
                                contador := 2;

                                        if contadorSaida < 0 THEN
                                                contadorSaida := contadorSaida*(-1);
                                                end if;

                                end if;



                end if;
```

```
SaidaVelocidade <= contadorSaida;
end process;
END ARCHITECTURE;
```

# **Apêndice F –** Código VHDL bloco

```
ENTITY FuzzyErro IS

PORT (   CLK              : IN bit ;
                          EntradaErro        : IN  integer  RANGE -180  to 180   ;
                          SaidaDefuzzyErro    : OUT integer  RANGE -1000 TO 1000  ;
                  SaidaDefuzzy2Erro    : OUT integer  RANGE -1000 TO 1000  ;
                          SaidaFMFErro                : OUT character               );

END ENTITY;

ARCHITECTURE FuzzyErro OF FuzzyErro IS

BEGIN
process(CLK)

        VARIABLE ChardaAux1    : character   ;
        VARIABLE ChardaAux2    : character   ;
        VARIABLE SaidaAUX1     : integer  range -1000 to 1000;
        VARIABLE SaidaAUX2     : integer  range -1000 to 1000;
        VARIABLE ErroAux       : integer range -180 to 180 ;

BEGIN

        IF (CLK'EVENT AND CLK = '1') THEN
        ErroAux := EntradaErro;

-- velocidade
-- /MN = A
--   N = B
--   Z = C
--   P = D
--  MP = E

                IF ErroAux >= -180 AND ErroAux <= -90 THEN
                saidaAUX1 := ((ErroAux - (-180))*100)/9;
                saidaAUX2 := (((-90) - ErroAux)*100)/9;
                        chardaAux1 := 'A';


                --conta
                end if;

                        IF ErroAux > -90 AND ErroAux <= 0  THEN
                        saidaAUX1 := ((ErroAux - (-90))*100)/9;
                        saidaAUX2 := ((0 - ErroAux)*100)/9;
```

```
                    chardaAux1 := 'B';


                -- conta
                end if;

                    IF ErroAux > 0 AND ErroAux <= 90  THEN
                        saidaAUX1 := ((ErroAux - 0)*100)/9;
                        saidaAUX2 := ((90 - ErroAux)*100)/9;
                        chardaAux1 := 'C';


                -- conta
                end if;

                    IF ErroAux > 90 AND ErroAux <= 180  THEN
                        saidaAUX1 := ((ErroAux - 90)*100)/9;
                        saidaAUX2 := ((180 - ErroAux)*100)/9;
                        chardaAux1 := 'D';


                    -- conta
                    end if;




        END IF;
        saidaFMFErro     <= chardaAux1 ;
        saidaDefuzzyErro  <= saidaAUX1  ;
        saidaDefuzzy2Erro <= saidaAUX2  ;

END PROCESS;
END ARCHITECTURE;
```

# **Apêndice G –** Código VHDL bloco FuzzyVelo

```
ENTITY FuzzyVelo IS

PORT (    CLK            : IN bit ;
                    EntradaVelo      : IN  integer  RANGE -3600  to 3600 ;
                    SaidaDefuzzy     : OUT integer range -1000 to 10000 ;
                SaidaDefuzz2y     : OUT integer range -1000 to 10000 ) ;

END ENTITY;

ARCHITECTURE FuzzyVelo OF FuzzyVelo IS

BEGIN
process(CLK)


        VARIABLE SaidaAUX1     : integer range -10000 to 10000 ;
        VARIABLE SaidaAUX2     : integer range -10000 to 10000 ;
        VARIABLE VelocidadeAux : integer range -50000 to 50000 ;

BEGIN

        IF (CLK'EVENT AND CLK = '1') THEN
        VelocidadeAux := EntradaVelo;

-- velocidade


                IF velocidadeAux >= 0 AND velocidadeAux <= 600 THEN
                saidaAUX1 := ((VelocidadeAux - 0)*10)/6;
                saidaAUX2 := ((600 - VelocidadeAux)*10)/6;



                --conta
                end if;

                        IF velocidadeAux > 600 THEN
                        saidaAUX1 := 1000;
                        saidaAUX2 := 0;
                        -- conta
                        end if;
```

```
        END IF;
        saidaDefuzzy  <= saidaAUX1  ;
        saidaDefuzz2y <= saidaAUX2  ;

END PROCESS;
END ARCHITECTURE;
```

# **Apêndice H –** Código VHDL bloco Defuzzy

```
ENTITY Defuzzy IS

PORT (    CLK              : IN bit ;
                    EntradaErroF       : IN  integer  range -1000 to 1000;
                    EntradaErro2F                    : IN  integer  range -1000 to 1000;
                    Valorcontrole                    : IN  character              ;

                    EntradaVeloF                     : IN  integer  range -1000 to 10000;
                    EntradaVelo2F                    : IN  integer  range -1000 to 10000;


                    SaidaDFuzzy        : OUT integer  range -1000 to 1000);

END ENTITY;

ARCHITECTURE Defuzzy OF Defuzzy IS

BEGIN
process(CLK)

VARIABLE DF1 : integer range -10000 to 10000;
VARIABLE DF2 : integer range -10000 to 10000;
VARIABLE DF3 : integer range -10000 to 10000;
VARIABLE DF4 : integer range -10000 to 10000;

VARIABLE Aux1 : integer range -90000 to 90000;
VARIABLE Aux2 : integer range -90000 to 90000;
VARIABLE Aux3 : integer range -90000 to 90000;
VARIABLE Aux4 : integer range -90000 to 90000;


BEGIN
        IF (CLK'EVENT AND CLK = '1') THEN


                    IF EntradaErroF  < EntradaVeloF THEN
                    DF1 := EntradaErroF;
                            ELSE
                            DF1 := EntradaVeloF;
                                    END IF;

                            IF EntradaErroF  < EntradaVelo2F THEN
                            DF2 := EntradaErroF;
                                    ELSE
                                    DF2 := EntradaVelo2F;
```

```
                                        END IF;

                                    IF EntradaErro2F  < EntradaVeloF THEN
                                    DF3 := EntradaErro2F;
                                        ELSE
                                        DF3 := EntradaVeloF;
                                            END IF;

                                        IF EntradaErro2F  < EntradaVelo2F THEN
                                        DF4 := EntradaErro2F;
                                            ELSE
                                            DF4 := EntradaVelo2F;
                                                END IF;

if  Valorcontrole = 'A' THEN
        aux1 := DF1*(-60);
        aux2 := DF2*(-30);
        aux3 := DF3*(-60);
        aux4 := DF4*(-30);

        ELSIF Valorcontrole = 'B' THEN
                        aux1 := DF1*0;
                        aux2 := DF2*0;
                        aux3 := DF3*(-60);
                        aux4 := DF4*(-30);

                ELSIF Valorcontrole = 'C' THEN
                                    aux1 := DF1*60;
                                    aux2 := DF2*30;
                                    aux3 := DF3*0;
                                    aux4 := DF4*0;

                                ELSIF Valorcontrole = 'D' THEN
                                    aux1 := DF1*60;
                                    aux2 := DF2*30;
                                    aux3 := DF3*60;
                                    aux4 := DF4*30;
        end if;


END IF;
SaidaDFuzzy <= ((aux1+aux2+aux3+aux4))/(DF1+DF2+DF3+DF4);

END PROCess;
END ARCHITECTURE;
```

# **Apêndice I –** Código VHDL bloco PWM

```
ENTITY PWM IS

PORT (   CLK          : IN   BIT                    := '1';
                      EntradaDoFuzzy   : IN   integer  RANGE -1000  to 1000 :=  0 ;
                      PWM1          : out  BIT                    := '1');

END ENTITY;

ARCHITECTURE PWM OF PWM IS

BEGIN
process(CLK)

        VARIABLE ContadorSobe  : INTEGER RANGE       0 TO 1000000:= 0       ;
        VARIABLE ContadorDesce : INTEGER RANGE       0 TO 100000 := 0       ;
        VARIABLE ValorDeDescida: INTEGER RANGE    25000 TO 200000 := 50000;
        VARIABLE Amostragem    : INTEGER RANGE       0 TO 5000000:= 0        ;


BEGIN

        IF (CLK'EVENT AND CLK = '1') THEN
        -- preciso de 50HZ

ContadorSobe  := ContadorSobe  +1;
ContadorDesce := ContadorDesce +1;
Amostragem    := Amostragem    +1;


                IF ContadorSobe >= 1000000 THEN --Correto
                        PWM1 <= '1';
                        ContadorSobe  := 0;
                        ContadorDesce := 0;
                END IF;
                        --IF ContadorDesce = 75000 THEN
                        IF ContadorDesce >= ValorDeDescida AND ContadorSobe /= 1000000 THEN
                                PWM1 <= '0';
                                ContadorDesce := 0;
                        END IF;

                --Valor de amostragem
                IF Amostragem = 50000 AND ((ValorDeDescida + EntradaDoFuzzy) > 25000) AND
((ValorDeDescida + EntradaDoFuzzy) < 200000) THEN
                        ValorDeDescida := ValorDeDescida + EntradaDoFuzzy;
```

```
                    Amostragem     := 0;
                    --ValorDeDescida := ValorDeDescida + EntradaDoFuzzy;
                END IF;

        END IF;


END PROCESS;

END ARCHITECTURE;
```

# **Apêndice J –** Código VHDL bloco CLKdividido

```
ENTITY CLKdividido IS

PORT (    CLK           : IN  BIT  ;
                                CLKdivididoBIT  : OUT BIT );


END ENTITY;

ARCHITECTURE CLKdividido OF CLKdividido IS

BEGIN
process(CLK)

VARIABLE ContadorCLK    : integer range 0 to 1500000 := 0;
VARIABLE aux            : bit     :='0';
CONSTANT constantetempo : integer := 1500000;



BEGIN

        IF (CLK'EVENT AND CLK = '1') THEN
                ContadorCLK := ContadorCLK + 1;

                IF ContadorCLK = constantetempo THEN ---verificar
                aux := not aux;
                ContadorCLK := 0;
                END IF;

        END IF;

        CLKdivididoBIT <= aux;

END PROCESS;
END ARCHITECTURE;
```