UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Faculdade de Engenharia e Ciências de Guaratinguetá

JORGE HENRIQUE RODRIGUES ALVES

LPM *Fuzzy* - Biblioteca de Módulos Parametrizáveis *Fuzzy*

GUARATINGUETÁ - SP

2023

Jorge Henrique Rodrigues Alves

LPM *Fuzzy* - Biblioteca de Módulos Parametrizáveis *Fuzzy*

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia e Ciências do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Leonardo Mesquita

GUARATINGUETÁ - SP

2023

Alves, Jorge Henrique Rodrigues
A474L LPM *Fuzzy* - Biblioteca de módulos parametrizáveis *Fuzzy* / Adriano Garufe Nogueira– Guaratinguetá, 2023.
55 f : il.
Bibliografia: f. 55

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia e Ciências de Guaratinguetá, 2023.
Orientador: Prof. Dr. Leonardo Mesquita

1. Lógica difusa. 2. Controladores elétricos. 3. Sistemas difusos. 4. VHDL (Linguagem descritiva de hardware) I. Título.

Luciana Máximo
Bibliotecária/CRB-8 3595

unesp

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

JORGE HENRIQUE RODRIGUES ALVES

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO EM ENGENHARIA ELÉTRICA"** APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Prof. Dr. Daniel Julien Barros da Silva Sampaio
Coordenador

BANCA EXAMINADORA:

Prof. Dr. Leonardo Mesquita
Orientador/UNESP-FEG

Prof. Dr. Samuel Euzedice de Lucena
UNESP-FEG

Profª. Dra. Paloma Maria Silva Rocha Rizol
UNESP-FEG

Janeiro/2023

DADOS CURRICULARES

JORGE HENRIQUE RODRIGUES ALVES

NASCIMENTO	04.05.1997 – São Bernardo do Campo/SP
FILIAÇÃO	Jorge Rodrigues Alves Simone Aparecida Vallim Alvim
2016/2022	Curso de Graduação em Engenharia Elétrica Universidade Estadual Paulista – "Júlio de Mesquita Filho", Campus de Guaratinguetá.

DEDICATÓRIA

Dedico este trabalho a minha família e o corpo docente, em especial meus pais, que me proporcionaram e me guiaram para essa conquista em especial.

AGRADECIMENTOS

Inicialmente, agradeço a Deus por sempre estar ao meu lado e prover todos as oportunidades na qual tive em minha vida, me dando forças para vencer todos os obstáculos.

Aos meus pais por acreditarem em mim e por todo o sacrifício que fizeram para que eu me tornasse quem eu sou hoje, sem eles eu não teria chegado até aqui. Agradeço por vocês sempre me apoiarem e proporcionarem todos os meios para ultrapassar as dificuldades que foram aparecendo. Agradeço a minha irmã por todas as conversas e ser minha melhor amiga.

A Sara, minha companheira de vida e maior confidente, que esteve ao meu lado desde que nos conhecemos. Obrigada por sempre me mostrar a ser uma pessoa melhor e ser uma inspiração na área academia e vida. Por fim, obrigado por me ajudar a revisar esse documento e por ter escutado inúmeras vezes eu explicando esse projeto com muita empolgação, desde o comecinho dele.

Ao Prof. Dr. Leonardo Mesquita, por toda paciência pelas diversas paradas e retomadas, sempre me trazendo uma palavra de tranquilidade e apoio. Agradeço por todo o apoio, orientações e conhecimento. Sou muito grato por você ter aceitado me acompanhar nessa etapa final da minha graduação.

Agradeço a todos os funcionários e professores da UNESP, *campus* de Guaratinguetá por contribuírem ao meu crescimento estudantil, profissional e pessoal.

Por fim, gostaria de deixar meu agradecimento aa todos que de alguma forma impactaram nessa etapa de formação.

RESUMO

A utilização da lógica *fuzzy* e a linguagem de descrição VHDL traz várias utilidades para projetos e indústrias. Logo foram criadas duas bibliotecas para utilização da lógica *fuzzy* I tipo Mamdani, sendo uma de funções e outra de módulos parametrizáveis, respectivamente. Com essas os usuários poderão criar diferentes sistemas fuzzy, sendo aplicados em diferentes projetos, por abrangerem uma ampla gama de funcionalidades. Para a validação desses pacotes, além de testes para cada módulo, criou-se um sistema de controle PID-*fuzzy* para um motor DC, utilizando as bibliotecas criadas. Os resultados foram satisfatórios, demostrando que estas funcionam conforme o esperado.

PALAVRAS-CHAVE: Lógica *Fuzzy*; Controladores *fuzzy tipo* – 1*;* Lógica difusa; PID-*Fuzzy*; VHDL.

ABSTRACT

The use of fuzzy logic in VHDL description language brings several utilities for projects and industries. So, two libraries were created for use of fuzzy logic I type Mamdani, one of functions and another of parameterizable modules, respectively. The users can create different fuzzy systems for apply in different projects, to cover a wide range of functionality. For the validation of these packages, in addition to tests for each module, a PID-fuzzy control system was created for a DC motor, using the libraries created. The results is satisfactory, showing that work as expected.

KEY WORDS: Fuzzy logic; Fuzzy controllers type-1; PID-Fuzzy; LPM fuzzy; VHDL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

FPGA	Arranjo de Portas Programáveis em Campo *(field-programmable gate array).*
VHDL	Linguagem de descrição de hardwares para circuitos integrados de altíssima velocidade (Very High-Speed Integrated Circuits Hardware Description Language).
LPM	Biblioteca de módulos parametrizados (*library of parameterized modules)*.
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos.
SDL	Especificação e Descrição Linguagem (Specification and Description Language).
HDL	Linguagem de descrição de hardware (Hardware description language).
DC	Corrente continua (direct current).
FI**S**	Sistema de inferência fuzzy (fuzzy infer'ence system).
RPM	Rotação por minuto.
PID	Proporcional, integral e derivativo.
PWM	Modulação por largura de pulso (Pulse Width Modulation).

Sumário

1 INTRODUÇÃO

Com o aumento da complexidade dos sistemas, a habilidade de criar declarações precisas sobre seus comportamentos diminui até que um limite seja alcançado. De modo que a precisão e significância se tornem características quase mutualmente exclusivas. Uma forma de tentar contornar essa dificuldade foi apresentada pela teoria dos conjuntos *fuzzy* que agrega uma flexibilidade para representar o mundo real, enquanto a logica *fuzzy* permite que o mundo real crie as declarações através da flexibilidade de representação dessa lógica (Baldwin,1996).

A esse novo conceito apresentou certa resistência a comunidade cientifica na época em que surgiu. Porém após alguns anos, os cientistas e teóricos observaram a gama de possibilidades que foram abertas. Dessa forma, começaram a surgir estudos e projetos oriundos deste conceito em todo o mundo, especialmente no Japão.

Atualmente, é possível observar a lógica *fuzzy* em diversas áreas devido a sua forma de flexibilidade para explorar indecisões do mundo real, como eletrodomésticos e indústria, medicina, transportes, neuro-computação e vários outros.

O *field-programmable gate array* (FPGA), com *Hardware Description Language* (HDL), possibilita o desenvolvimento e simulação de sofisticados circuitos digitais de forma rápida (CHU, 2008). Por isso, com sua pluralidade, ele é muito utilizado para projetos de diferentes tamanhos, tendo assim grande utilidade em várias esferas de desenvolvimento, utilizando-o como um dispositivo de simulação, protótipo ou até final.

Com isso, existe o interesse de criar funções, procedimentos e blocos parametrizáveis para a utilização da lógica *fuzzy* no FPGA, de forma a facilitar a utilização deste em diferentes projetos.

1.1 OBJETIVO GERAL

Este trabalho apresenta como objetivo criar módulos para facilitar a utilização da Lógica *Fuzzy* no *field-programmable gate array* (FPGA), através da linguagem de descrição *Very High Speed Integrated Circuits Hardware Description Language* (VHDL).

1.2 OBJETIVOS ESPECÍFICOS

- Fundamentar técnica e cientificamente o estudo por meio de revisão bibliográfica.

- Criar um pacote de função/procedimento, em VHDL, com o intuito de utilizar a lógica *fuzzy* no FPGA.
- Criar uma *library of parameterized modules* (LPM) que são biblioteca de módulos parametrizáveis, no VHDL, com o intuído de utilizar a lógica *fuzzy* no FPGA.
- Desenvolver um projeto para aplicar as ferramentas criadas nesse trabalho de graduação.

2 REVISÃO BIBLIOGRÁFICA

2.1 LÓGICA *FUZZY*

A teoria dos conjuntos *fuzzy* foi introduzido por Lotfi Zadeh em 1965. Ele foi o responsável pelo começo da ideia de incerteza no campo da engenharia, este seria, portanto, o embrião desta lógica. Mais tarde, Zadeh propôs a ideia de algoritmos *fuzzy* que serviu de fundamento para a Lógica *Fuzzy*. Após essas publicações, em 1972, Michio Sugeno junto com Zadeh escreveu sobre os conceitos de medição e integral difusa. A grande mudança ocorreu em 1974 com Ebraham Mamdani que aplicou a lógica *fuzzy* em controladores pela primeira vez (Tanaka, 1996).

Lógica é o estudo dos métodos e princípios do raciocínio de todas as formas, sendo que existem duas delas: a clássica e a difusa (ou *fuzzy)*. A lógica clássica lida com proposições que a resposta é verdadeira (1) ou falso (0) (Klir;Yuan,1995). A lógica difusa, em termos gerais, trabalha com proposições na qual o valor está entre completamente verdadeiro (1) e completamente falso (0) (Jamsshidi; Vadiee; Ross, 1993).

Dessa forma, a lógica *fuzzy*, como dito anteriormente, abre as possibilidades tendo em vista que lógica clássica é muito "preto no branco", 1 ou 0, sim ou não, bom ou ruim, entre outras possibilidades. Todavia, em nosso dia a dia, as coisas não são assim, ou seja, existem pessoas "menos más" que as outras. Na vida real, há diferentes tonalidades entre o preto e o branco e bem como uma infinidade de números entre o 0 e o 1 existem número reais. Existem também medidas vagas como por exemplo "suba alguns degraus", a palavra "alguns" nessa frase é vaga, podendo ser qualquer valor. Por isso foi criado a lógica difusa (Zadeh;Kacprzyk,1992).

2.1.1 Conjunto *fuzzy*

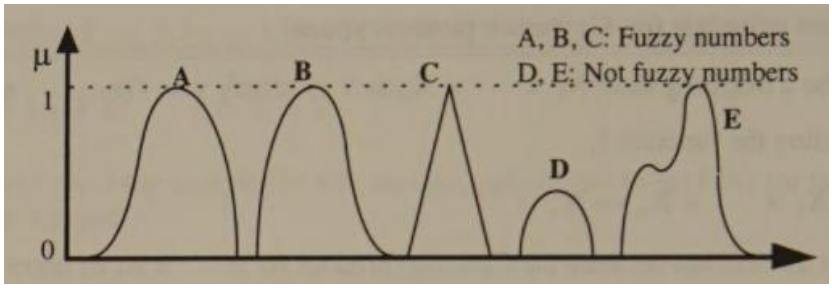
O conjunto *fuzzy* pode ser definido como a atribuição de um valor para todo o universo de discurso, cujo valor é um nível de pertencimento ao conceito apresentado por esse conjunto, normalmente apresentado pelo intervalo de valor entre 0 e 1, sendo mais ou menos compatível ao conceito subjetivo apresentado (Klir;Yuan,1995). Um caso especial de conjunto *fuzzy* é o número *fuzzy*, que deve ter as seguintes características:

- Estar definido nos números reais;
- Ser convexo;

- Função de pertinência deve ser contínua;
- O valor máximo de pertinência deve ser no máximo 1 (Tanaka,1996).

A Figura 1 representa bem o que é considerado número *fuzzy*, nesta percebe-se que as letras A, B e C são números *fuzzy*, pois preenchem todas as características destes mencionados acima. Na letra D, o valor máximo não é 1, logo esse parâmetro não é atingido. Já a letra E, a Figura não é convexa.

Figura 1. Representação de conjuntos *fuzzy* e números *fuzzy.*



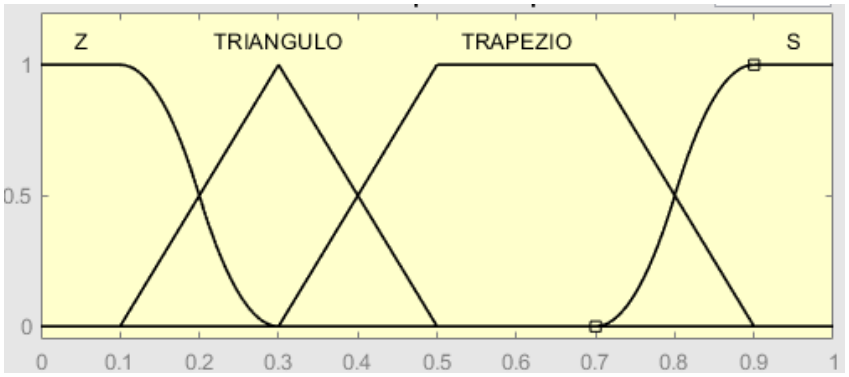
Fonte: Klir, G.; Yuan (1995).

Existem diversos números *fuzzy*, os mais comuns utilizados são:

- Função de pertinência tipo triangulo;
- Função de pertinência tipo trapézio;
- Função de pertinência tipo S, como caso especial da trapezoidal.
- Função de pertinência tipo S.
- Função de pertinência tipo Z, como caso especial da trapezoidal.
- Função de pertinência tipo Z.

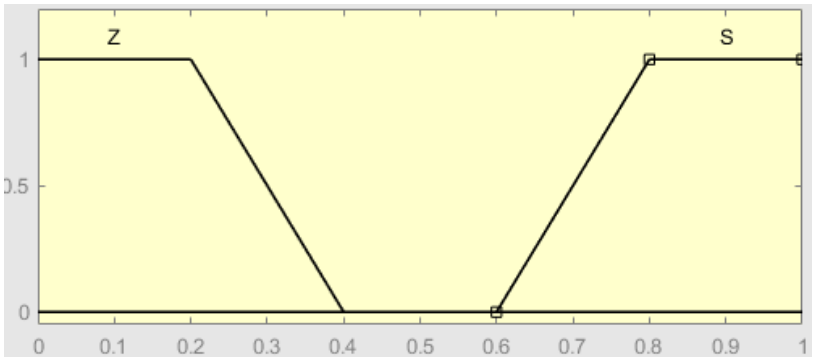
Nas Figuras 2 e 3 estão representadas as funções mencionadas:

Figura 2. Números *fuzzy* função tipo Z, triângulo, trapézio e S.



Fonte: Elaborado pelo autor (2022).

Figura 3. Números *fuzzy* função tipo Z e S, caso especial de trapézio.



Fonte: Elaborado pelo autor (2022).

2.1.2 Operações de conjuntos nebulosos

As principais operações de conjuntos *fuzzy* são união, intersecção e Complemento, estes também presentes na lógica *crisp* comumente utilizadas.

2.1.2.1 União

A resposta da união de conjuntos nebulosos é o maior entre os valores de pertinência para o valor de entrada. Como por exemplo, A U B, onde A e B são conjuntos *fuzzy:*

$$\mu_{A\cup B}(x) = \mu_A(x),\ se\ \mu_A(x) \geq \mu_B(x) \quad (1)$$

$$\mu_{A\cup B}(x) = \mu_B(x),\ se\ \mu_A(x) < \mu_B(x) \quad (2)$$

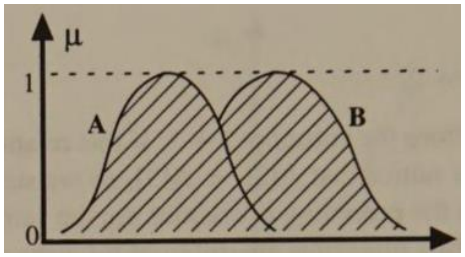
Outra forma de descrever a união dos conjuntos está disposto conforme abaixo:

$$\mu_{A\cup B}(x) = \max\{\mu_A(x), \mu_B(x)\} \quad (3)$$

(Tanaka,1996)

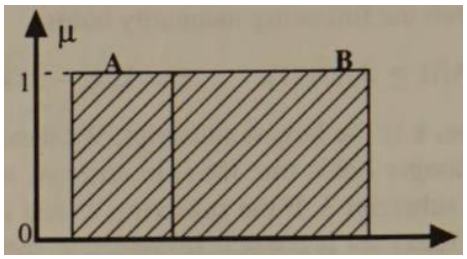
Para uma melhor visualização nas Figuras 4 e 5 estão a representação gráfica da união dos conjuntos *fuzzy* em comparação aos conjuntos *crisp*.

Figura 4. Representação da união dos conjuntos *fuzzy.*



Fonte: Tanaka (1996).

Figura 5. Representação da união dos conjuntos *crisp.*



Fonte: Tanaka (1996).

2.1.2.2 Intersecção

A intersecção de conjuntos nebulosos representa o menor valor de pertinência para o valor de entrada entre os conjuntos. Como por exemplo, A ∩ B, onde A e B são conjuntos *fuzzy:*

$$\mu_{A\cap B}(x) = \mu_A(x),\ se\ \mu_A(x) \leq \mu_B(x) \quad (4)$$

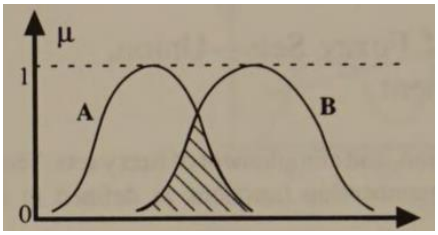
$$\mu_{A\cap B}(x) = \mu_B(x),\ se\ \mu_A(x) > \mu_B(x) \quad (5)$$

Outra forma de descrever a união dos conjuntos está disposto conforme abaixo:

$$\mu_{A\cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} \quad (6)$$

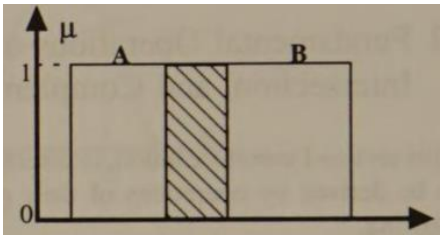
Nas Figuras 6 e 7 estão a representação gráfica da intersecção dos conjuntos *fuzzy* em comparação aos conjuntos *crisp*.

Figura 6. Representação da intersecção dos conjuntos *fuzzy*.



Fonte: Tanaka (1996).

Figura 7. Representação da intersecção do conjunto *crispy*.



Fonte: Tanaka (1996).

2.1.2.3 Complemento

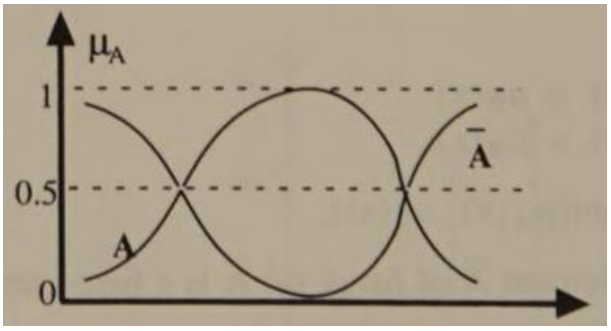
O complemento de conjuntos *fuzzy* é o valor de pertinência que falta para chegar ao valor máximo (normalmente 1), para o valor de entrada. Como por exemplo, Ā, onde A é um conjunto *fuzzy:*

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{7}$$

(Tanaka,1996)

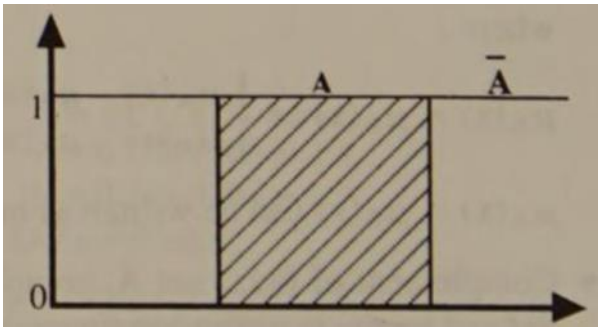
As Figuras 8 e 9 trazem a representação gráfica do complemento do conjunto *fuzzy* em comparação aos conjuntos *crisp*.

Figura 8. Representação do complemento do conjunto *fuzzy*



Fonte: Tanaka (1996).

Figura 9. Representação do complemento do conjunto *crispy*



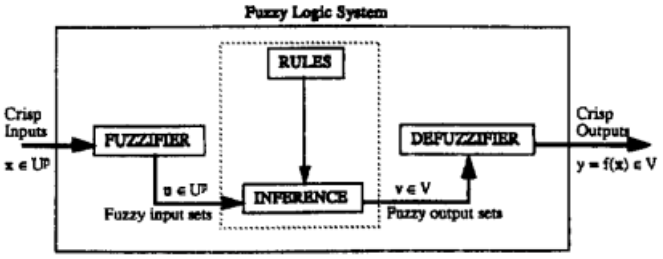
Fonte: Tanaka (1996).

2.1.3 Método de inferência Mamdami

Os sistemas *fuzzy* são os sistemas que-possuem pelo menos uma variável, sendo entrada ou saída. Normalmente estes conjuntos são números *fuzzy* e as variáveis do sistema fuzzy, associadas as variáveis linguísticas (Klir;Yuan,1995).

A Figura 10 contêm o esquema mais usado para controladores e processamento de sinais utilizando a lógica *fuzzy*. Neste é possível observar que existem 4 blocos, *Fuzzificador*, Inferência, Regras e *Defuzzificador*, os sinais de entrada quanto o de saída são números nítidos. (MENDEL; Jerry,1995)

Figura 10. Representação do complemento do conjunto *crispy*.



Fonte: Tanaka (1996).

Existem diferentes métodos de raciocínio para resolução de sistemas *fuzzy*, o mais utilizado foi o proposto por Mahmood Mamdani, este possui uma estrutura simples, com operações de intersecção (mínimo) e união (máximo), utilizou-se como interface de regras o formato *"IF-THEN"* (*SE-ENTÃO*) (Tanaka,1996).

Para resolução de um sistema pelo método de inferência Mamdami, deve-se seguir os seguintes passos (conforme explicado no livro *Fuzzy Sets and Fuzzy Logic – Theory and Applications* dos autores Klir, G. J. e Yuan, B.):

- Passo 1: Definir as variáveis de entrada e saída, inserindo seus conjuntos *fuzzy*, que normalmente são ligadas a um representante linguístico. Importante lembrar que os conjuntos difusos quase sempre são números difusos e as formar mais comuns são mostradas na Figura 2.

- Passo 2: Nesse passo será realizado a *fuzzificação*. O propósito desse é encontrar o grau de pertencimento do valor de entrada da variável que está representado pelas funções de *fuzzificação*, que abrangem o universo por completo. Estas funções são definidas pelos conjuntos *fuzzy*.

- Passo 3: Montagem da base de regras de inferência *fuzzy*, elas podem ser determinadas tanto por especialistas humanos no assunto ou por dados empíricos que através de métodos de aprendizagem retratam o universo de aplicação.

  O formato das regras, nesse caso, serão o *"IF-THEN"* (*SE-ENTÃO*). Para exemplificar temos as duas equações **X e Y**, sendo as variáveis A, B e C, e alguns dos números *fuzzy* são A_1, B_2, C_1.

$$Se\ A = A_1 e\ B = B_1, então\ C = C_1$$

$$Se\ A = A_1 ou\ B = B_1, então\ C = C_1$$

O número de regras máximo pode ser obtido pela multiplicação dos números *fuzzy* das variáveis, como por exemplo havendo 2 variáveis e cada tendo 5 conjuntos *fuzzy*, a quantidade de regras serão 25. Porém pode haver regras que possuem algum conflito ou que não haja uma regra para algum caso.

- Passo 4: Os valores de pertinência encontrados no passo 2 serão combinados com as regras, para que se encontre o grau de pertinência para cada conclusão de cada regra.

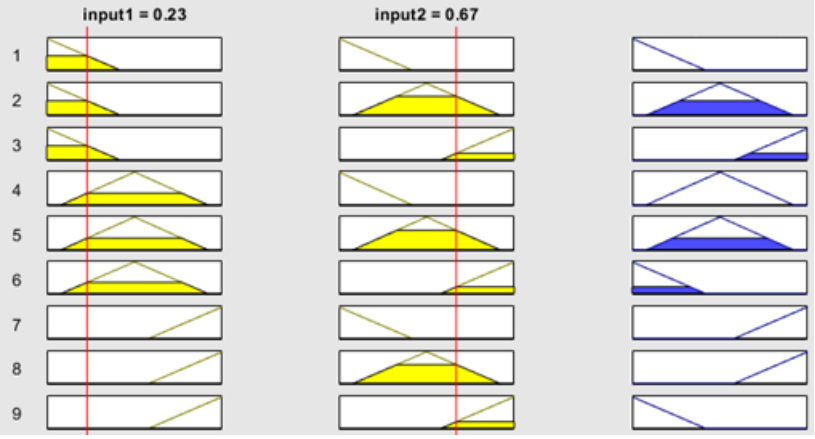
As duas operações são "e" (*and*) e "ou" (*or*), conforme é possível observar nas equações X e Y, a diferença é que com a *and* o cálculo do valor de pertinência da resposta é o menor entre os de entrada, já o *or* é o máximo.

$$\mu_c = \min\{\mu_A(A_1), \mu_B(B_1)\}\ \ (and) \tag{8}$$

$$\mu_c = \max\{\mu_A(A_1), \mu_B(B_1)\}\ \ (or) \tag{9}$$

Na Figura 11 está exposto um exemplo gráfico das regras de um sistema com 2 variáveis de entrada e 1 de saída e a resposta do mínimo entre os valores de pertinência.

Figura 11. Gráfico de representação das regras e da inferência



Fonte: Elaborado pelo autor (2022).

- Passo 5: No último passo está a *defuzzificação*, neste ocorre a convergência das conclusões em um único valor real, este valor é o resultado do sistema. Existem diversos métodos de *defuzzificação*, neste trabalho optou-se pelo desenvolvimento de quatro tipos específicos.

O resultado vem através do cálculo do máximo destas, como por exemplo em um sistema com 2 variáveis e cada possua 2 números *fuzzy*, 4 regras e 1 variável de saída com 3 variáveis (C_1, C_2 e C_3), sendo esses com a função triangulo [(0; 0,25; 0,5); (0,25; 0,5; 0,75); (0,5; 0,75; 1)]. Considerando os maiores valores de cada conclusões como:

$$\mu_{c1} = 0{,}5;\ \mu_{c2} = 0{,}2;\ \mu_{c3} = 0{,}2. \quad (10)$$

Considerando o passo de 0,01, teríamos: $\mu_c(0{,}1)$ até $\mu_c(0{,}25) = 0{,}5$; $\mu_c(0{,}25)$ até $\mu_c(0{,}4) = -2x + 1$; $\mu_c(0{,}4)$ até $\mu_c(1) = 0{,}2$. A Figura 12, representa graficamente o resultado desse exemplo aplicado.

Figura 12. Gráfico de representação da área de resultado do exemplo.

Fonte: Elaborado pelo autor (2022).

Método do centro de área (D_{CA}): O resultado é o valor natural referente ao centro de gravidade da área resultante da inferência. Segue a equação do caso discreto, a função resposta é C e os valores de pertinência são $\{Z_1, Z_2, Z_3, Z_4, ..., Z_n\}$.

$$D_{CA}(C) = \frac{\sum_{k=1}^{n} C(z_k) z_k}{\sum_{k=1}^{n} C(z_k)} \quad (11)$$

Método da média do máximo (D_{MM}): O resultado é o valor natural referente a média do maior valor da resultante da inferência. Segue a equação do caso discreto, a função resposta é C, M refere-se o conjunto na qual estão os valores de pertinência máximo e os valores de pertinência são $\{Z_1, Z_2, Z_3, Z_4, ..., Z_n\}$.

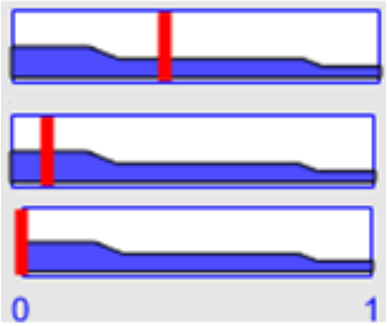
$$D_{MM}(C) = \frac{\sum_{z_k \in M} z_k}{|M|} \quad (12)$$

Método menor do máximo (D_{MeM}): O resultado é o menor valor natural referente ao maior valor da resultante da inferência. Segue a equação do caso discreto, a função resposta é C, M refere-se o conjunto na qual estão os valores de pertinência máximo e os valores de pertinência são $\{Z_1, Z_2, Z_3, Z_4, ... , Z_n\}$.

$$D_{MeM}(C) = \min(M) \quad (13)$$

Para comparação dos diferentes métodos, na Figura 10, o gráfico demonstra as respostas para o método centro de área, o método médio do máximo e o método menor do máximo, sendo respectivamente de cima para baixo.

Figura 13. Gráfico de comparação dos resultados dos métodos DCA, DMM e DMeM para um exemplo.



Fonte: Elaborado pelo autor (2022).

2.2 FPGA E QUARTUS II 13.1

O campo de matrizes de portas programáveis (FPGA), em inglês *Field Programmable Gate Array*, é um dispositivo logico que contém matrizes de células de logicas genéricas e chaves programáveis. Através de linguagens de descrição é possível programar as células genéricas para implementar uma função específica e as chaves para realizar conexões entre as células, assim realizando as funções desejadas pelo usuário, tendo incontáveis possibilidades (Chu, 2008).

Existem diversas linguagens de descrição de hardware, como VERILOG, SDL e VHDL. Este projeto foi desenvolvido utilizando a última citada, por se tratar de domínio público e ser amplamente utilizada. O *Very High Speed Integrated Circuits Hardware Description Language* (VDHL), em português Linguagem de descrição de *hardware* para circuitos integrados de velocidade muito rápida, é uma notação utilizada para ser lida tanto por humanos, quando para a máquina, sendo uma forma de comunicação entre ambos, este suporta o desenvolvimento, manutenção e teste de design de *Hardware* (IEEE Computer Society, 2019).

O *software* utilizado para realizar as programações nesse trabalho é o Quartus II 13.1 da empresa Intel, pois este permite desenvolver a lógica através da linguagem VHDL, realizar simulações e a gravação destes no FPGA.

2.2.1 Subprogramas e pacote

Subprogramas e pacotes são muito utilizados em VHDL, pelo fato de facilitarem a utilizações e repetições de lógicas instituídas previamente.

Subprogramas define valores ou comportamentos existentes, utilizando parâmetros, podendo converter tipos de valores, respostas para os parâmetros utilizados e sua lógica, e definindo partes do processo. Existem 2 tipos de subprograma, a função e o procedimento. A diferença entre os dois está na chamada dos subprogramas, o procedimento é uma função, enquanto a função é uma expressão com um valor retornado (IEEE Computer Society, 2019).

O pacote permite que diferentes unidades de design compartilhem as mesmas declarações, podendo agregar diferentes subprogramas em um pacote. Assim pode-se utilizar

subprogramas do pacote no programa, declarando a utilização do pacote a qual o subprograma pertence (IEEE Computer Society, 2019).

3 MATERIAL E MÉTODO

Para o desenvolvimento das bibliotecas na linguagem de descrição VHDL utilizou-se *o Quartus* II 13.1 da Altera, na qual foi possível escrever e testar as respostas dos módulos usando o pacote ModelSim. Para caráter de validação dos resultados utilizou-se o *Matlab*, comprando os resultados encontrados de cada módulo.

Após a criação das bibliotecas, para verificar a usabilidade destas, utilizou-se o sistema de controle de um motor DC, o console PCT-1 da DEGEM *systems*, para aplicar um controlador PID *Fuzzy*, carregado no FPGA DE2-115 da TERASIC.

3.1 DESENVOLVIMENTO DAS BIBLIOTECAS FUZZY

Nesse projeto foi desenvolvido dois pacotes, um contendo as funções necessárias para implementar cada um dos blocos constituintes de um processador *fuzzy*, e outro pacote de dados contendo que constitui a LPM *fuzzy*.

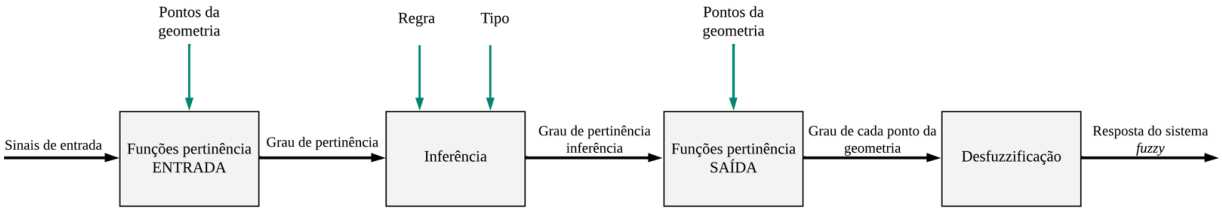
Os blocos para a utilização da lógica *fuzzy* são:

- Funções de pertinência da entrada
- Inferência
- Funções de pertinência da saída
- *Defuzzificação*

Existem diferentes funções e módulos para cada um desses itens, somente quando se utiliza Singletons para as variáveis de saída, que os blocos funções de pertinência da saída e *Defuzzificação* se juntam em uma única função e modulo.

A Figura 14 mostra um esquema de como funciona a montagem do um sistema *fuzzy* utilizando as bibliotecas. Este esquema é abrangente, dependendo da função ou modulo haverá mais entradas, como quando se quer utilizar de modo síncrono.

Figura 14. Esquema de blocos para a utilização da lógica *fuzzy* tipo 1 utilizando as bibliotecas criadas.



Fonte: Elaborado pelo autor (2023)

As características dos pacotes são:

- O programa é feito para entradas com palavras de 8 bits;
- O universo de discurso é 8 bits (inteiro igual a 0 até 255).
- Os graus de pertinência vão de 0 até 100.

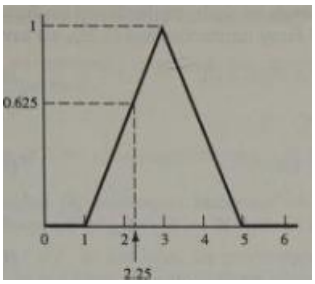
3.2 PACOTE DE FUNÇÕES *FUZZY*

Neste pacote estão descritas as funções e procedimentos para a criação do sistema de logica *fuzzy*. O nome deste pacote é PACOTE_FUNC_*FUZZY*1. As funções e procedimentos serão dívidas nos 4 blocos anteriormente apresentados.

3.2.1 Funções de fuzzificação

As funções de pertinência das entradas, são formatos geométricos que ao inserir o sinal de entrada a resposta é o grau de pertinência projetada nesse formato, como por exemplo a Figura 15, nesta mostra o grau de pertinência de uma entrada em uma função do tipo triangulo.

Figura 15. Representação de uma função de pertinência com uma entrada.

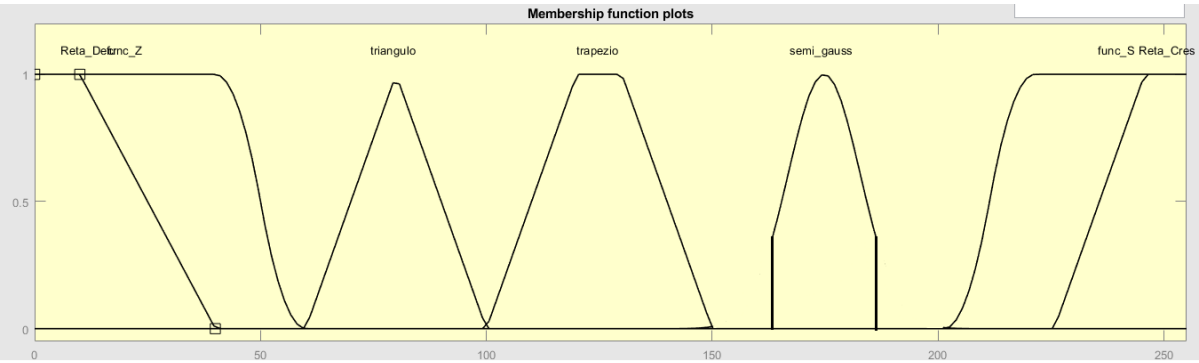


Fonte: Tanaka (1996).

Como podemos observar na Figura 15, o grau de pertinência para uma entrada de valor 2,25 é 0,625, para essa função. Assim funcionam as funções desse bloco.

Existem 7 funções de diferentes formas geométricas, essas são: triangulo, trapézio, função reta decrescente (particularidade da função triângulo), função reta crescente (particularidade da função triângulo), função Z, função S, semi gaussiana. Abaixo na Figura 16 estão representadas as 7 formar.

Figura 16. Representações das funções de pertinência existentes na biblioteca.



Fonte: Elaborado pelo autor (2022).

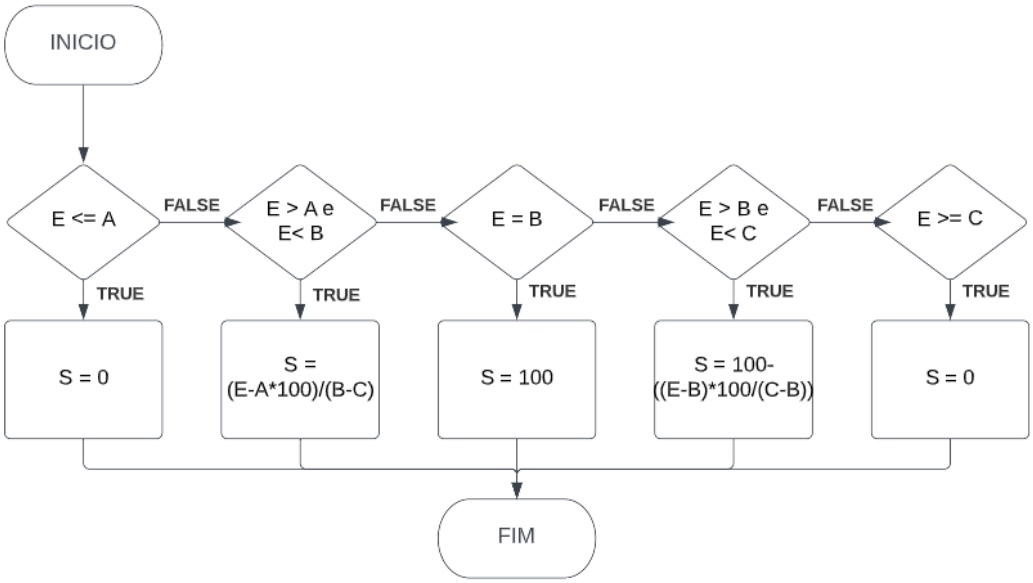
As entradas das funções são o valor no qual quer converter ao universo *fuzzy* e os pontos que descrevem a geometria escolhida, estão são respectivamente dos tipos std_logic_vector e inteiros.

As quantidades de entrada responsáveis pelas geometrias são diferentes para cada tipo. A saída por sua vez é o grau de pertinência do dado de entrada em inteiros.

- Triângulo: 3 entradas, correspondentes a ponto inicial mínimo, ponto máximo de pertinência e ponto final de mínimo.
- Trapézio: 4 entradas, os pontos são inicial mínimo, inicial de máximo, final de máximo e final de mínimo.
- Função Reta decrescente: 2 entradas, pontos são início máximo e final mínimo.
- Função Z: 2 entradas, pontos são início máximo e final mínimo.
- Função Reta crescente: 2 entradas, pontos são início mínimo e final máximo.
- Função S: 2 entradas, pontos são início mínimo e final máximo.
- Função Semi-gauss: 2 entradas, pontos são a média e a variância.

Para o entendimento, na Figura 17 apresenta o diagrama da função triângulo. Sendo E o sinal de entrada; A, B e C os pontos da geometria e o S retorno da função.

Figura 17. Diagrama da função de pertinência: Triângulo



Fonte: Elaborado pelo autor (2022).

3.2.2 Inferência

Neste bloco estão presentes procedimentos referente a inferência do tipo Mamdani, os procedimentos são divididos em dois grupos, para que permita ao usuário realizar de diferentes maneiras este processo, sendo de forma automática (para grandes quantidades de regras) e de maneira isolada (para pequenas quantidades de regras).

A forma automática, trata-se de 4 procedimentos diferentes, que consiste na quantidade de variais de entrada existem, de 2 a 5. Este percorre todas as possibilidades de combinação das variáveis, porém para definir a resposta faz se necessário uma lista na qual nomeamos através de números as variais de saída, porém o 0 representa a ausência de regra. Como por exemplo, existindo duas variais de entrada, com duas funções de pertinência, e duas de saída. Chamaremos as variais de entrada de A e B, para as de saída 1 e 2. As regras definidas são:

$$Se\ entrada1 = A_1\ e\ entrada2 = B_2, então\ saida = 1$$

$$Se\ entrada1 = A_2\ e\ entrada2 = B_1, então\ saida = 2$$

$$Se\ entrada1 = A_2\ e\ entrada2 = B_2, então\ saida = 1$$

A lista de regras será (0,1,2,1), sendo (A_1/B_1 não possui regras; $A_1/B_2 = 1$; $A_2/B_1 = 2$; $A_2/B_2 = 1$).

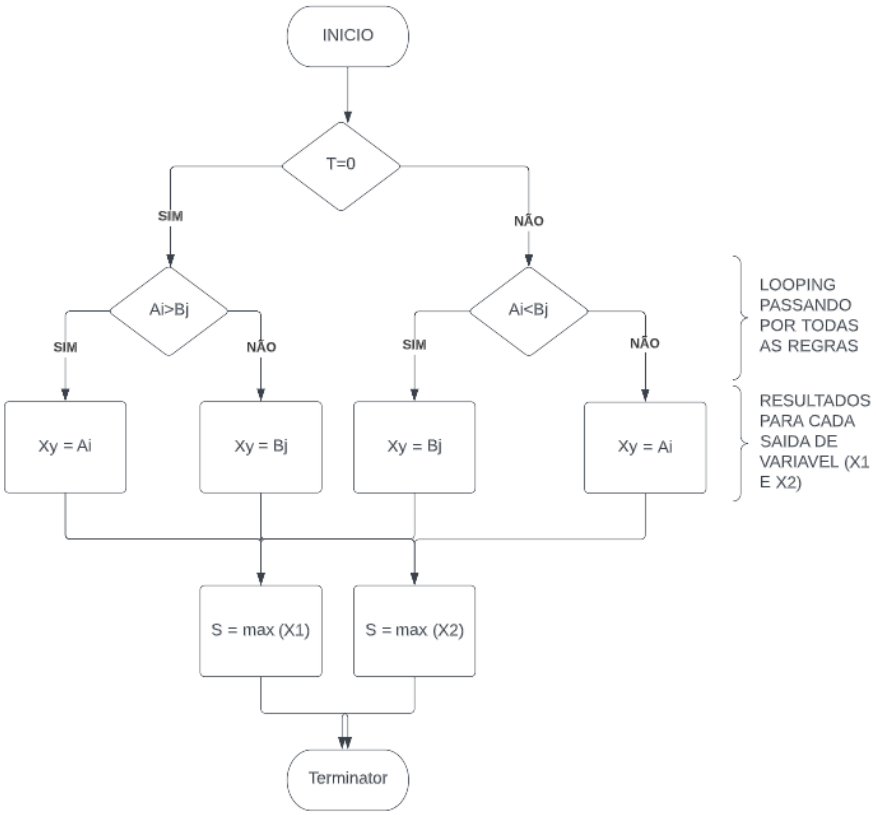
Se no caso fosse três variais com duas funções de pertinência cada, a forma de escrever as regras seriam:

$$(A_1/B_1/C_1;\ A_1/B_1/C_2;\ A_1/B_2/C_1;\ A_1/B_2/C_2;$$

$$A_2/B_1/C_1;\ A_2/B_1/C_2;\ A_2/B_2/C_1;\ A_2/B_2/C_2).$$

Os procedimentos são infer_2, infer_3, infer_4 e infer_5. Na qual o que difere uma da outra é a quantidade de variáveis de entrada referente as variáveis *fuzzificadas*, sendo estas uma lista com os valores de pertinência de cada função das entradas. Logo as variáveis dos procedimentos são variáveis de entrada, regra (trata-se de uma lista conforme explicado), tipo (sendo 0 para AND e 1 para OR) e a saída (que é uma lista com os valores encontrados para cada variável de saída. Segue na Figura 18 o diagrama da inferência conforme o exemplo exposto anteriormente.

Figura 18. Diagrama Inferência do tipo Mamdani para 2 variáveis de entrada e 2 de saída.



Fonte: Elaborado pelo autor (2023).

A segunda maneira de realizar a inferência é através de dois procedimentos interligados, **inf_reg** e **inf_max**, o primeiro é utilizado para a criação da regra utilizando AND

(mínimo) ou OR (máximo), após criado todas as regras, é juntado em listas as respostas para cada variável de saída, na qual o segundo procedimento realizará o cálculo do máximo para a resposta do grau de pertinência para cada variável de saída.

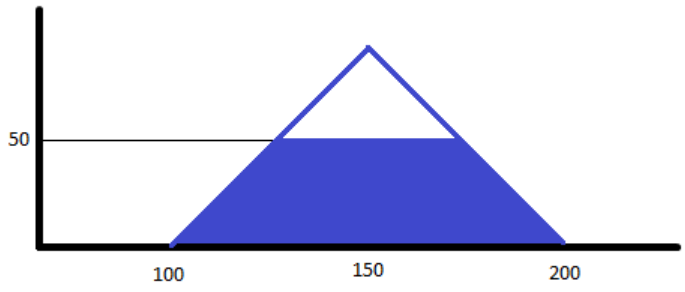
3.2.3 Variáveis de saída

As variáveis de saída podem possuir diferentes funções de pertinência, sendo as previstas pela biblioteca, triângulo, trapézio, função reta decrescente (particularidade da função triangulo), função reta crescente (particularidade da função triangulo), função Z, função S e Singleton. Esta última corresponde a um ponto que percorre de 0% até 100% de pertinência, as outras já foram explicadas anteriormente, pois são iguais as funções de entrada.

Estas funções de pertinência são expostas através de funções (exceto o Singleton, pois o processo deste será explicado na sessão 5.1.4), na qual a resposta é uma lista de valores de pertinência correspondente a função por todo o universo de discurso, sendo que o valor máximo correspondente é o valor encontrado no bloco inferência, para facilitar o entendimento a Figura 19 mostra um exemplo da função triangulo, na qual o valor de pertinência é 50%.

Neste é possível observar que para os pontos menores que 100 o valor é 0%, para valores entre 100 e 125 os valores correspondem a função de reta crescente, entre 125 e 175 é 50%, de 175 a 200 trata-se da função de reta decrescente e a partir de 200 é 0%.

Figura 19. Representação gráfica dos valores de pertinência para a função triângulo, sendo a entrada 50%.



Fonte: Elaborado pelo autor (2023).

As entradas são os pontos para a criação da geometria, conforme as de entrada, e a pertinência encontrada pelo bloco inferência.

3.2.4 Defuzzificação

O último bloco é a *Defuzzificação*, na qual a saída é a resposta da lógica *fuzzy*, sendo esse um número natural, que para nosso universo de discurso está entre o 0 e 255. Existem diferentes métodos de *defuzzificação*, iremos abordar média do máximo, menor do máximo, centro de área e método das alturas (utilizado na variável *Singleton*). Foram nomeados os procedimentos da seguinte maneira, respectivamente, ***Def_MediaMaximos***, ***Def_baixoMaximos***, ***Def_centroide*** e Def_singletons**.**

Nos três primeiros métodos, as variáveis dos procedimentos são o vetor das listas dos valores de pertinência encontrados de cada variável de saída e a resposta do tipo *std_logic_vector.*

Para o *Singleton*, o bloco *defuzzificação* e variáveis de saída se tornam um único procedimento, as variáveis são a lista com os valores de pertinência de cada variável de saída, uma lista com todos os pontos Singleton e a resposta do tipo std_logic_vector. A resposta é a média ponderada dos pontos. Conforme a função abaixo, VP trata do valor de pertinência.

$$D_S = \frac{\sum_{k=1}^{n} Ponto.VP_n}{\sum_{k=1}^{n} VP_n} \quad (14)$$

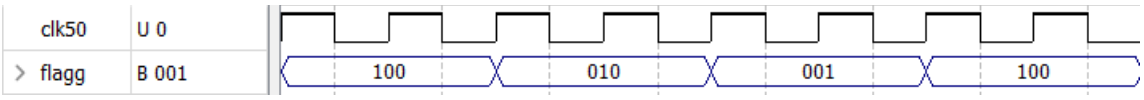
3.3 PACOTE DE MÓDULOS PARAMETRIZAVEIS *FUZZY*

O pacote de módulos parametrizáveis *fuzzy* (*LPMfuzzy*) está localizado na biblioteca *PACOTE_LPM_FUZZYI*. Como o pacote de função, este também possui os quatro blocos, porém existe mais dois módulos que é utilizado para utilizar na forma síncrona. Este pacote pode ser usado de forma síncrona ou assíncrona, conforme o usuário desejar.

Os módulos de funções de *fuzzificação*, inferência, variáveis de saída, inferência de maneira isolada e *defuzzificação Singleton*, possui as mesmas entradas, porém acrescidas com o sinal de *clock*, o flag utilizado para controle e o a posição desse flag desejado. Além dessas existe o modulo de *defuzzifucação*, que além dessas três entradas citadas anteriormente, possui o vetor das listas dos valores de pertinência encontrados de cada variável de saída, o tipo de *defuzzificação* (entre os três métodos previamente citados) e a resposta do tipo *std_logic_vector.*

Para a utilização de forma síncrona é necessário o modulo de controle chamado **contr**, que seleciona os flags um por um, de forma sequencial. Estes flags consistem em uma palavra de tamanho definido pelo usuário, na qual o bit 1 é rotacionado em cada dois pulso do relógio, a Figura 20 mostra o funcionamento.

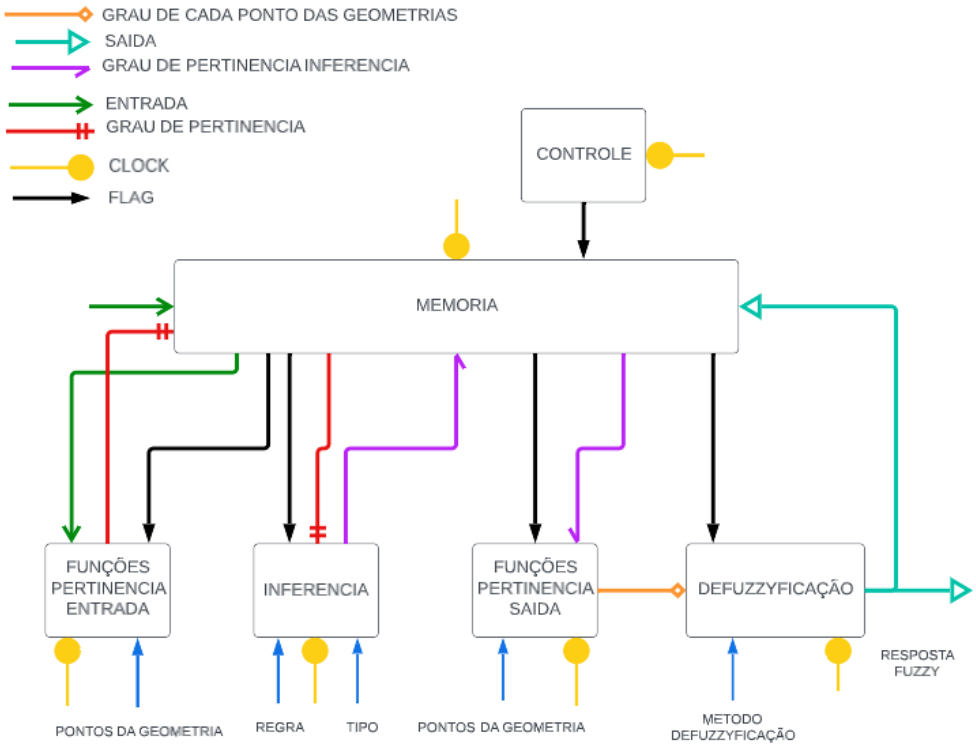
Figura 20. Representação do funcionamento do módulo controle.



Fonte: Elaborado pelo autor (2022).

Outro módulo é de uma memória RAM chamada de **memor_ram**, que serve para armazenar os resultados de cada passo do processo, assim ocorrendo algum interrompimento é possível saber o último valor daquela etapa e em qual *flag* estava acionado. Na Figura 21 está representado o funcionamento da forma síncrona com o controlador e a memória.

Figura 21. Diagrama do funcionamento da resolução da lógica *fuzzy* com controle e memória.



Fonte: Elaborado pelo autor (2022).

3.4 APLICAÇÃO – CONTROLADOR *FUZZY* MOTOR

Com a intenção de validar a utilização do LPM *Fuzzy*, submeteu-se uma aplicação na qual usam-se as bibliotecas criadas e assim comparando com o modelo com *MatLab*, conhecer o funcionamento e validar sua eficácia.

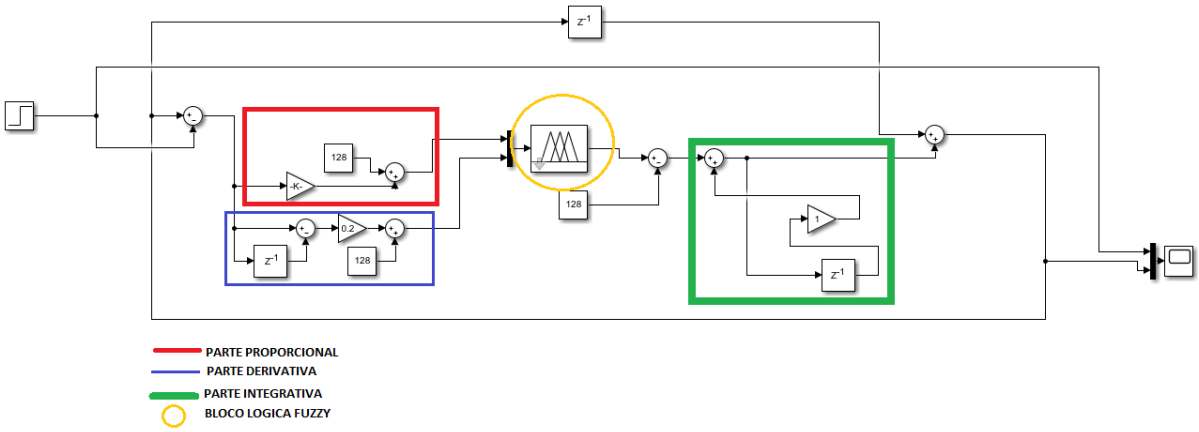
A aplicação é um Controlador PID-Fuzzy para um motor DC. Utilizando o sistema de controle de velocidade de um motor DC, um FPGA e o Quartus II 13.1 foi possível criar esse controlado, para efeito de comparação criou-se um modelo no Simulink. Os equipamentos utilizados são:

- FPGA ALTERA UNIVERSITY PROGRAM “DE0’ – TERASIC;
- DC MOTOR SPEED CONTROL SYSTEM CONSOLE PC-1 – DEGEM SYSTEM
- QUARTUS II 13.1
- *SIMULINK*

3.4.1 Simulink

Na Figura 22 está descrito o modelo de controle PID-*Fuzzy* implementado no *Simulink*. É possível observar na imagem que existem quatro setores separados, sendo estas partes do controle Proporcional, Derivativo, Integrativa e *fuzzy*.

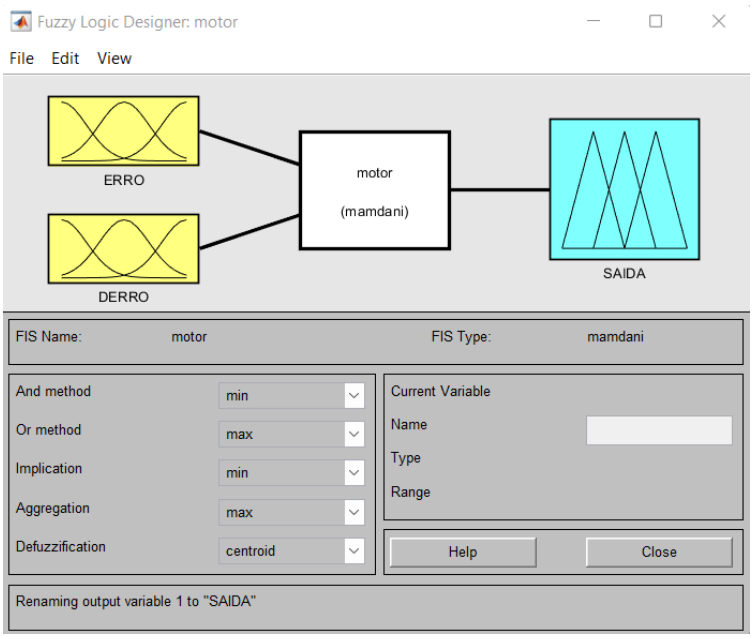
Figura 22. Diagrama do funcionamento da resolução da lógica *fuzzy* com controle e memória.



Fonte: Elaborado pelo autor (2022).

O *Simulink* disponibiliza um bloco para utilizar o controle *fuzzy* pelo toolbox *fuzzy* do *Matlab*, porém para utilizá-lo é necessário a criação do sistema de inferência no *Matlab* e após isso inseri-lo no bloco. A melhor maneira é realizar pelo FIS *Edictor do Fuzzy Logic Toolbox*, pois neste é possível realizar todo o processo de forma didática. A Figura 23 mostra o editor aberto, nesta página é possível escolher se o tipo de inferência será Mamdani ou Sugeno (Mamdani o escolhido), também quantas variais de entrada (erro e "derro") e saída (Saída), o tipo de *defuzzificação* (centroide), entre outras coisas. Após cria-se as funções de pertinência das variáveis de entrada e saída, conforme as Figuras 23, 24, 25,26, respectivamente.

Figura 23. Página do FIS Edictor, com as informações do exemplo.



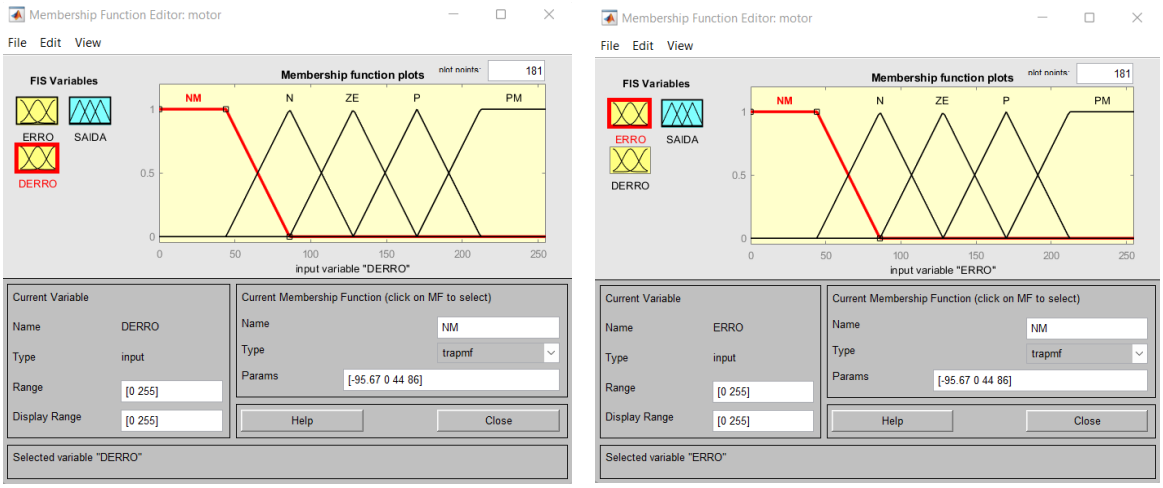
Fonte: Elaborado pelo autor (2022).

As variáveis de entradas são o erro entre a resposta encontrada na leitura da velocidade do motor e a velocidade que se espera e a variação do erro, conforme as equações abaixo.

$$erro = motor - set \tag{15}$$

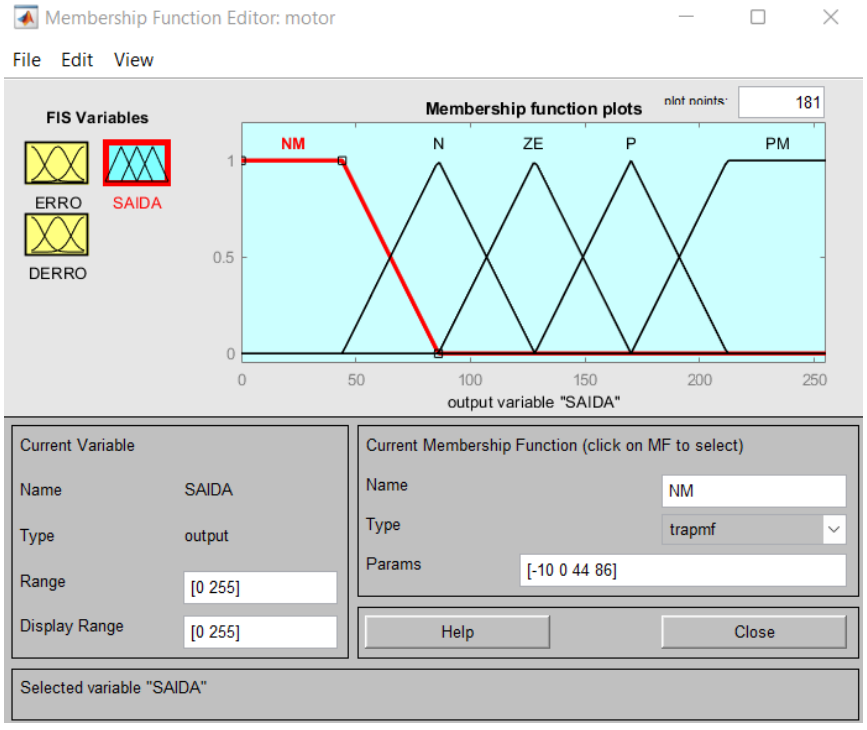
$$derro = erro - erro\_anterior \tag{16}$$

Figura 24. Páginas das funções de pertinência das variáveis de entrada.



Fonte: Elaborado pelo autor (2022).

Figura 25. Página das funções de pertinência da variável de saída.



Fonte: Elaborado pelo autor (2022).

Os membros das variáveis são negativos mínimos (NM), Negativo (N), Zero (ZE), Positivo (P) e Positivo máximo (PM). Para facilitar a comparação utilizamos o universo do discurso sendo de 0 até 255. Na tabela 01 está representado as regras para a inferência em questão.

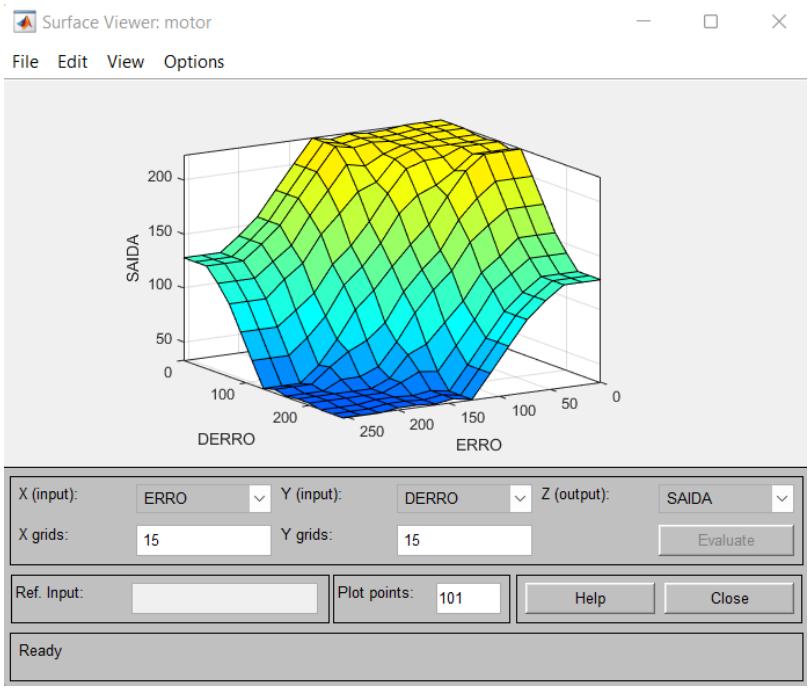
Tabela 1. Regras de inferência do controlador *fuzzy*.

		Erro				
		NM	N	Z	P	PM
Variação do erro	NM	PM	PM	P	ZE	N
	N	PM	PM	P	ZE	N
	Z	PM	P	ZE	N	NM
	P	P	ZE	NM	NM	NM
	PM	ZE	N	NM	NM	NM

Fonte: Elaborado pelo autor (2023).

O *Matlab* consegue fornecer a superfície de saída das regras, na qual facilita o entendimento do processo conforme a Figura 26.

Figura 26. Página do gráfico da superfície de resposta para o modelo *fuzzy* do controle do motor.



Fonte: Elaborado pelo autor (2022).

Na saída do controlado *fuzzy* está o bloco de controle integrativo que consiste no seguinte cálculo.

$$Saida\ Motor = Saida\ Inf + Saida\ Inf\_anterior \quad (17)$$

3.4.2 Fpga/Quartus

O projeto de controle do motor utilizando o sistema de controle de velocidade do motor DC, o programa realizado no *quartus* e o FPGA tem como objetivo realizar a leitura da velocidade do motor e fazer com que chegue na velocidade desejada. O esquema de blocos dos componentes criados para a realização desse projeto é mostrado na Figura 27. Existem 5 blocos desenvolvidos para tratar os sinais recebidos e realizar os cálculos necessários para a realização do controle *PID-Fuzzy* do motor. Estes são:

- Frequencímetro: Encontra a velocidade em RPM do motor através do tacômetro presente no console da *Degem* em unidade, dezena, centena e milhar e disponibiliza no display. Este bloco é um contador de quantas vezes os furos da placa passa a cada 1 segundo, por possuir 60 furos, a quantidade é igual ao valor de rotações por minuto.

- Controle: Calcula o erro e a variação do erro através do valor encontrado no frequencímetro e o valor desejado (set). Sendo esse escolhido através de *switches* conforme a Figura 27.

Figura 27. Valores da velocidade (rpm) que se quer dependendo dos sinais dos switches.

```
    case set is                 --selecionar o valor da saida dependente do valor da selecao
    when "000" => z := 200;     -- set igual a '000' ent?o valor de saida igual a 200
    when "001" => z := 400;     -- set igual a '001' ent?o valor de saida igual a 400
    when "010" => z := 500;     -- set igual a '010' ent?o valor de saida igual a 500
    when "011" => z := 750;     -- set igual a '011' ent?o valor de saida igual a 750
    when "100" => z := 900;     -- set igual a '100' ent?o valor de saida igual a 900
    when "101" => z := 1000;    -- set igual a '101' ent?o valor de saida igual a 1000
    when "110" => z := 1500;    -- set igual a '110' ent?o valor de saida igual a 1500
    when "111" => z := 2000;    -- set igual a '111' ent?o valor de saida igual a 2000
end case;
```

Fonte: Elaborado pelo autor (2022).

- Sistema *Fuzzy:* Esse bloco é constituído com o LPM *fuzzy*, utilizando as mesmas informações explicadas na sessão 6.1. Neste sistema é adotado como frequência do controlador, 5Hz, pois com essa frequência é possível obter valore mais confiáveis do tacômetro e ajuda na observação de cada etapa do processo, por se tratar de uma demonstração foi decidido utilizar essa frequência.

- Teste_ex: Este calcula o valor de saída do sistema, com a porção integrativa.

- PWM: Envia um sinal pwm para o sistema degem, através do valor encontrado no Test_ex. A tabela1 mostra a relação da tensão de alimentação do motor com a velocidade do motor, esses valores foram encontrados de forma empírica.

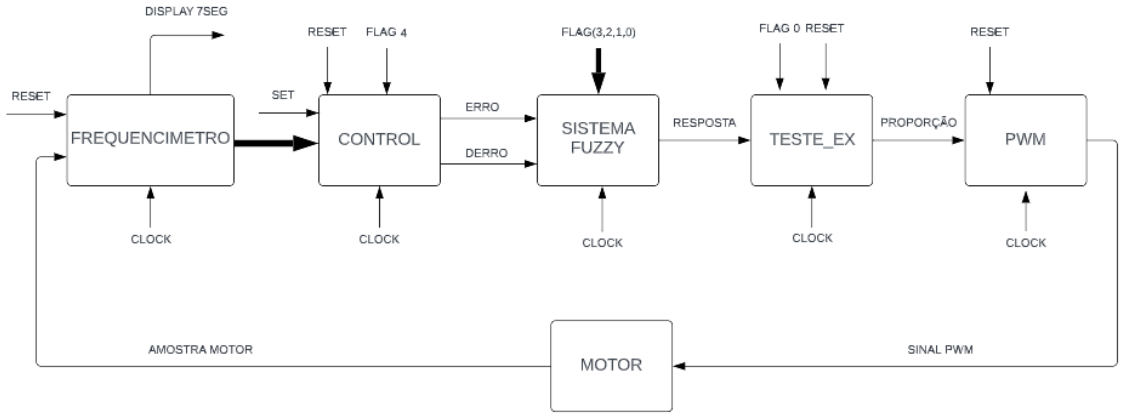
Tabela 2. Relação da tensão e velocidade do motor.

Tensão alimentação - motor	Velocidade [rpm]
1,6 V	0
2,39 V	200
3,03 V	400
3,30 V	500
4,07 V	750
4,57 V	900
3,30 V	500
4,07 V	750
4,57 V	900
4,82 V	1000
6,33 V	1500
7,76 V	2000

Fonte: Elaborado pelo autor (2023).

O pulso PWM nessa aplicação tem a finalidade de atribuir a tensão desejada no motor DC, através da variação da largura do pulso da onda quadrada é possível modular potência incidente no motor. A onda quadrada possua seu ciclo de máximo e de mínimo pela frequência, se nesse período 50% do tempo for de máximo (*duty cycle*), a potência entregue será a metade do total, seguindo essa lógica se o *duty cycle* for de 90%, a potência entregue será de 90% da potência total. Utilizando esse conceito é possível controlar a velocidade do motor dependendo da resposta encontrada no sistema PID-*fuzzy*.

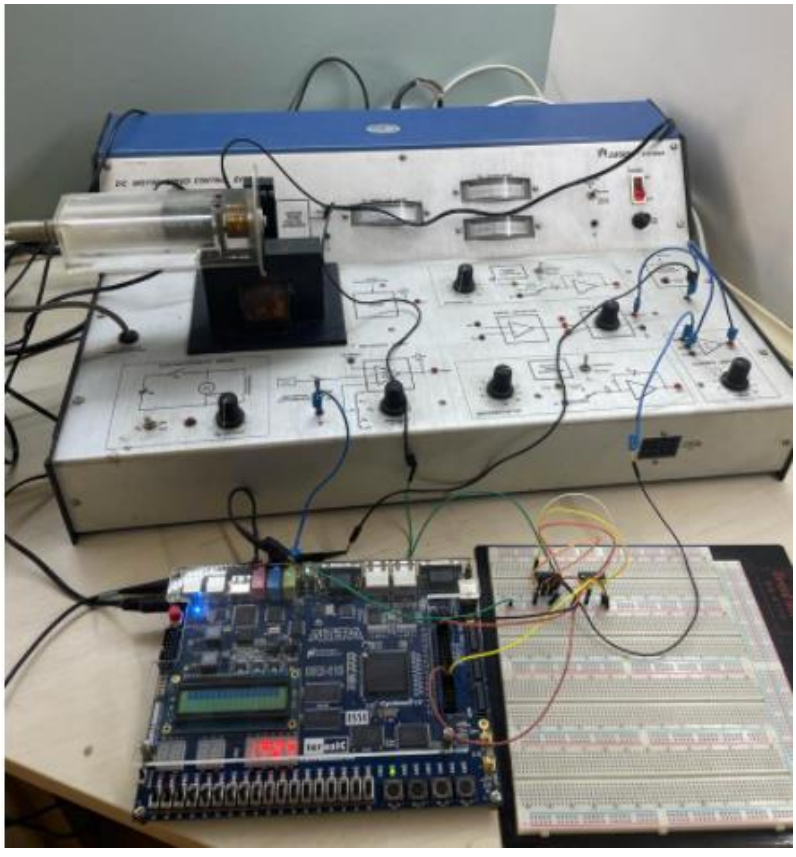
Figura 28. Diagrama do funcionamento do controle PID do motor DC.



Fonte: Elaborado pelo autor (2022).

Segue abaixo, na Figura 29, o protótipo do circuito implementado. No sistema existem amplificadores, que foram utilizados para ajustar os valores obtidos do tacômetro e da saída do FPGA.

Figura 29. Fotografia do esquema montado do controlador *PID-Fuzzy*.



Fonte: Elaborado pelo autor (2023).

4 SIMULAÇÕES E RESULTADOS

4.1 MÓDULOS *FUZZY*

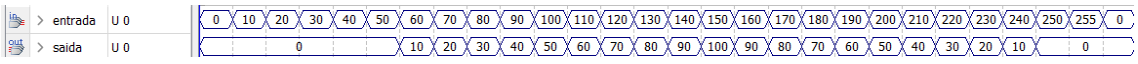
Para a validação das bibliotecas, será testado cada bloco separadamente, inserindo vetores de teste e verificando as respostas encontradas em comparação com as esperadas. Importante observar que o projeto realizado não utiliza informações do tipo real (em digital, essa informação é tratada como sendo do tipo ponto-flutuante), assim os valores encontrados são inteiros, o que se for comparado ao *MatLab* ou outro sistema que utiliza a notação de ponto-flutuante, haverá um erro que será mínimo e não faz parte do escopo do projeto.

4.1.1 Módulos das funções de Pertinência - Entrada

Com o objetivo de padronizar os pontos de entrada e varrer todo o universo amostral adotou valores de 0 até 250, com passo de 10, e um último valor de 255.

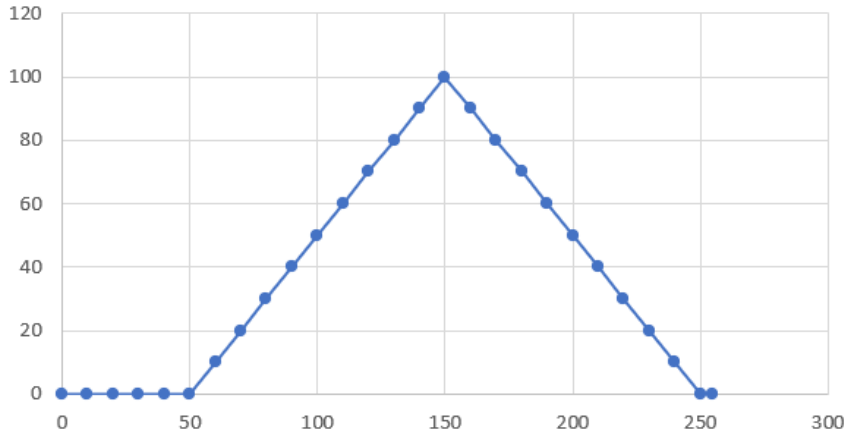
Para a função triangulo os pontos de geometria escolhidos foram 50, 150 e 250. Na Figura 29 é possível verificar os valores de saída simulados pelo Modelsim. para melhor observação do resultado criou-se um gráfico, conforme a Figura 30. Através dessas imagens é possível concluir que o resultado está conforme o esperado. Foram feitos outros testes, que não estão nesse documento, porém não houve em nenhum caso erros.

Figura 30. Resposta da função de pertinência triângulo sobre o universo do discurso, sendo pontos geométricos 50, 100 e 150.



Fonte: Elaborado pelo autor (2023).

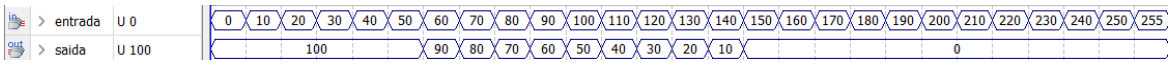
Figura 31. Gráfico representando os valores encontrados na Figura 29



Fonte: Elaborado pelo autor (2023).

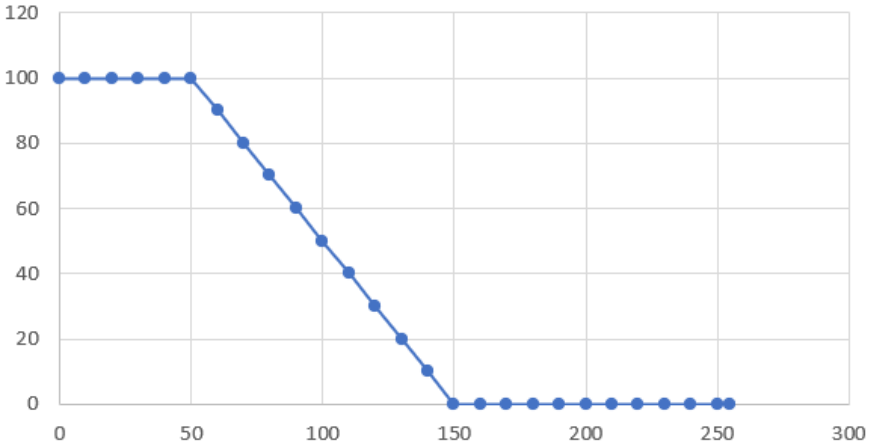
Na Figura 31 está o resultado encontrado para função S (caso especial do trapézio), nesta os pontos geométricos são 50 e 150, para melhor visualização temos o resultado plotado em um gráfico (Figura 32), na qual à conclusão de que os valores encontrados são iguais aos esperados, assim não havendo erros.

Figura 32. Resposta da função de pertinência Z (caso especial do trapézio) sobre o universo *do discurso*, com pontos geométricos 50 e 150.



Fonte: Elaborado pelo autor (2023).

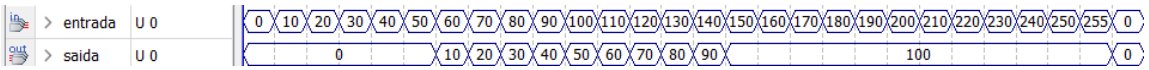
Figura 33. Gráfico representando os valores encontrados na Figura 31



Fonte: Elaborado pelo autor (2023).

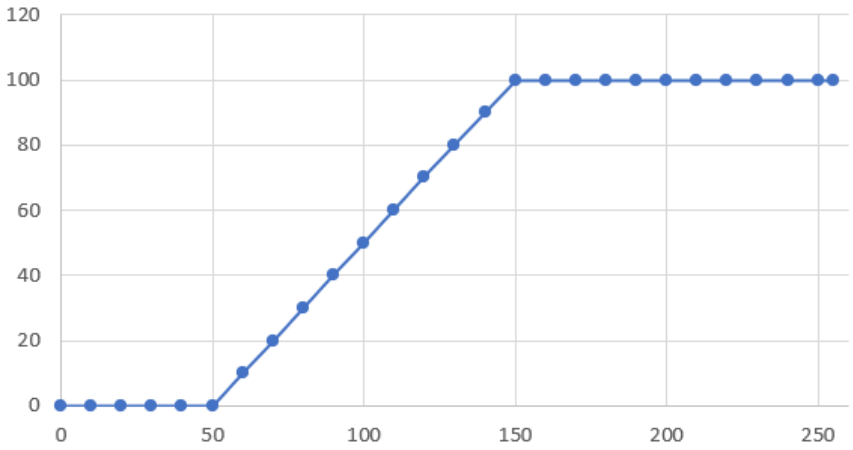
Na função S (caso especial do trapézio) os valores encontrados também estão conformes os esperados, conforme observa-se nas Figuras 33 e 34, para pontos geométricos de 50 e 150.

Figura 34. Resposta da função de pertinência S (caso especial do trapézio) sobre o universo *do discurso*, com pontos geométricos 50 e 150.



Fonte: Elaborado pelo autor (2023).

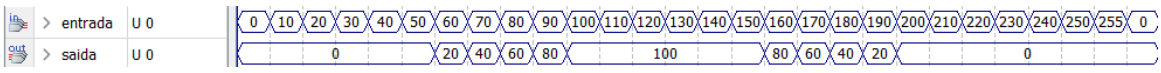
Figura 35. Gráfico representando os valores encontrados na Figura 33



Fonte: Elaborado pelo autor (2023).

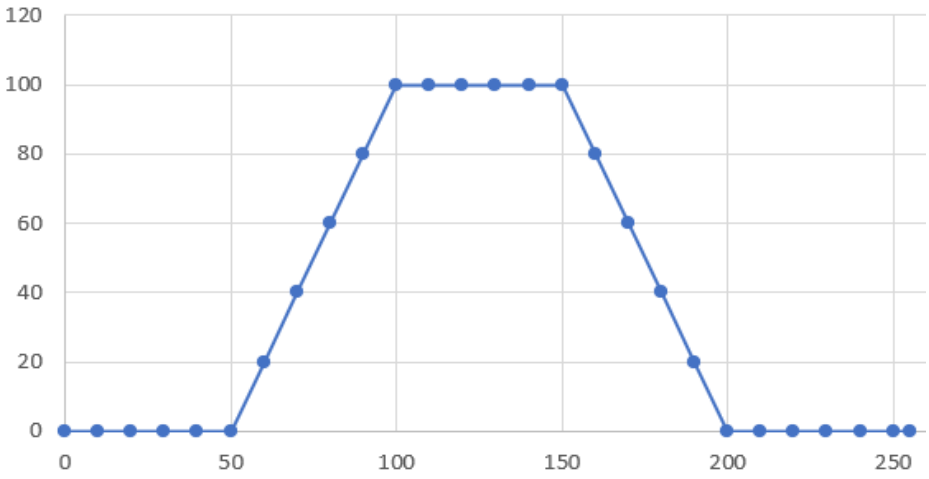
O trapézio com pontos geométricos de 50, 100, 150 e 200, como esperado, também não teve desvio dos valores projetados, conforme as Figuras 35 e 36 mostram.

Figura 36. Resposta da função de pertinência trapézio sobre o universo *do discurso*, com pontos geométricos 50, 100, 150 e 200.



Fonte: Elaborado pelo autor (2023).

Figura 37. Gráfico representando os valores encontrados na Figura 35.



Fonte: Elaborado pelo autor (2023).

Devido a não utilização de ponto flutuante e a utilização de potenciação na equação da função S (equação 15), essa possui um erro atrelado. Após testes empíricos conclui-se que o erro está abaixo de 17%. As Figuras 37 e 38 ilustram esse erro.

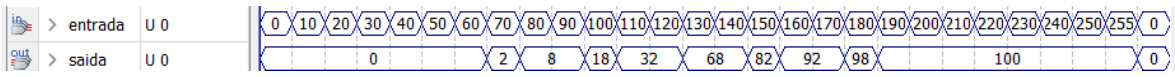
$$f(x) = \begin{cases} 0, & x \leq a \\ 2\left(\frac{x-a}{b-a}\right)^2, & a \leq x \leq \frac{a+b}{2} \\ 1 - 2\left(\frac{x-b}{b-a}\right)^2, & \frac{a+b}{2} \leq x \leq b \\ 1, & x \geq b \end{cases} \quad (15)$$

a: Valor do início da subida;

b: Valor do final da subida.

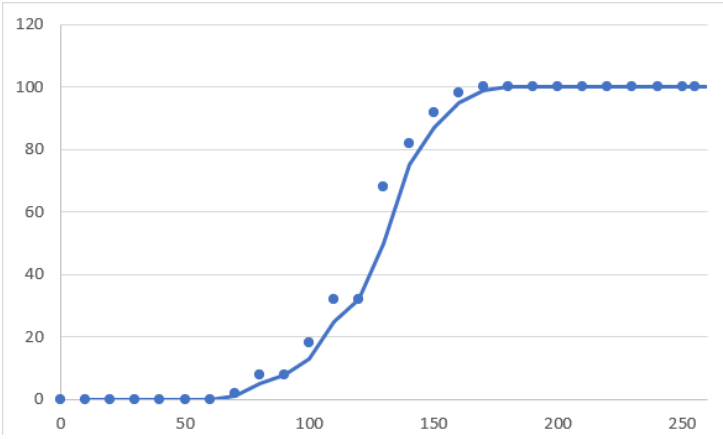
Importante notar que a equação é para valores de 0 até 1, porém no universo da biblioteca é 0 até 100, sendo necessário multiplicar por 100 os resultados.

Figura 38. Resposta da função de pertinência S sobre o universo *do discurso*, com pontos geométricos 50 e 200.



Fonte: Elaborado pelo autor (2023).

Figura 39. Gráfico representando os valores encontrados na Figura 37.



Fonte: Elaborado pelo autor (2023).

O caso da função S é igual à da função Z, pois as Figuras 39 e 40 demostra que existe um erro atrelado e o motivo é o mesmo da anterior, sendo esse erro de 16%. Na equação 16 está descrito o comportamento dessa função.

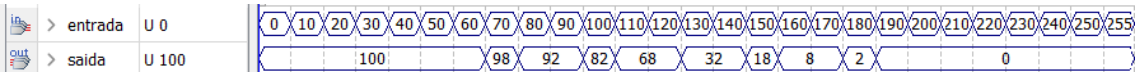
$$f(x) = \begin{cases} 1, & x \leq a \\ 1 - 2\left(\frac{x-a}{b-a}\right)^2, & a \leq x \leq \frac{a+b}{2} \\ 2\left(\frac{x-b}{b-a}\right)^2, & \frac{a+b}{2} \leq x \leq b \\ 0, & x \geq b \end{cases} \quad (16)$$

a: Valor do início da descida;

b: Valor do final da descida.

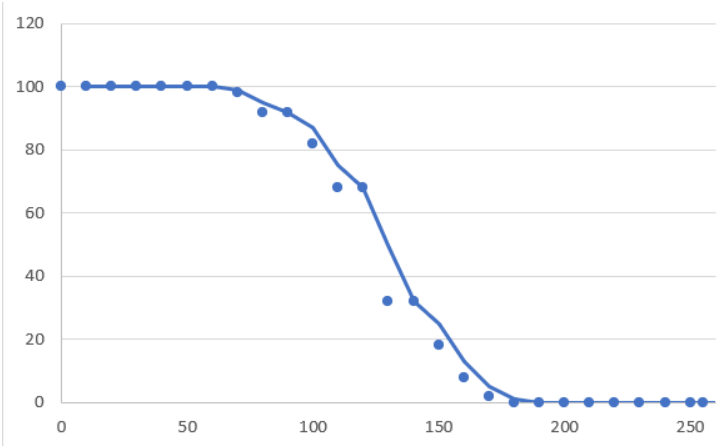
Importante notar que a equação é para valores de 0 até 1, porém o universo da biblioteca é 0 até 100, sendo necessário multiplicar por 100 os resultados.

Figura 40. Resposta da função de pertinência S sobre o universo *do discurso*, sendo pontos geométricos 50 e 200.



Fonte: Elaborado pelo autor (2023).

Figura 41. Gráfico representando os valores encontrados na Figura 39.



Fonte: Elaborado pelo autor (2023).

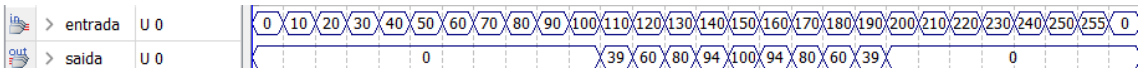
A função semi-gaussiana apresenta erro atrelado devido à grande dificuldade de utilizar o número de Euler na linguagem de descrição, sem o ponto flutuante; por isso definimos que erros menores que 12% não eram interessantes, assim criou-se essa função que se o resultado for de 0% até 30% a resposta é 0%; porém para valores maiores segue a função gausseana. A Figura 41 e 42 mostra o comportamento dessa função de forma gráfica. A equação da gaussiana é mostrada na equação 17.

$$f(x) = e^{\frac{-(x-a)^2}{2.b^2}} \quad (17)$$

a: Valor da média da gaussiana;

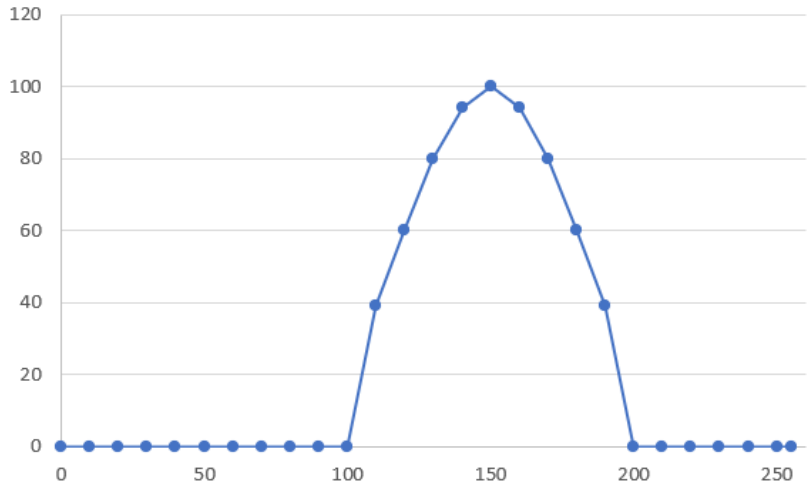
b: Desvio padrão.

Figura 42. Resposta da função de pertinência semi-gaussiana sobre o universo *do discurso*, com pontos geométricos 150 e 35.



Fonte: Elaborado pelo autor (2023).

Figura 43. Gráfico representando os valores encontrados na Figura 41.



Fonte: Elaborado pelo autor (2023).

4.1.2 Módulos de inferência

Estes módulos possuem o mesmo funcionamento, por isso optou-se por realizar o teste no de 2 variáveis de entrada. Utilizou-se como exemplo o caso de 2 variáveis de entrada com 2 funções de pertinência, 2 variáveis de saída, Mamdani do tipo AND e a seguinte regra (0,1,2,1).

Para uma melhor compreensão, foram utilizados 4 cenários de valores de pertinência de entrada, na qual pode observar na Figura 44, junto com os resultados.

Figura 44. Resposta do modulo de inferência com duas variáveis de entrada e cada com duas funções de pertinência.

in	var11	U 50	50	149	240	34
in	var12	U 222	222	30	240	34
in	var21	U 185	185	77	50	180
in	var22	U 153	153	133	50	180
out	saida1	U 153	153	133	50	34
out	saida2	U 185	185	30	50	34

Fonte: Elaborado pelo autor (2023).

Levando em consideração o primeiro caso, calcula-se o valor para comparação. As regras são as seguintes:

Regra 1: $saida_1 = \min\{50,153\} = 50$;

Regra 2: $saida_2 = \min\{222,185\} = 185$;

Regra 3: $saida_1 = \min\{222,153\} = 153$;

Após, calculam-se os valores das saídas:

$saida_1 = \max\{50,153\} = 153$;

$saida_2 = \max\{185\} = 185$;

Os valores encontrados correspondem com as respostas do módulo de inferência. As outras respostas também estão corretas. Logo, confirmamos a eficácia deste módulo.

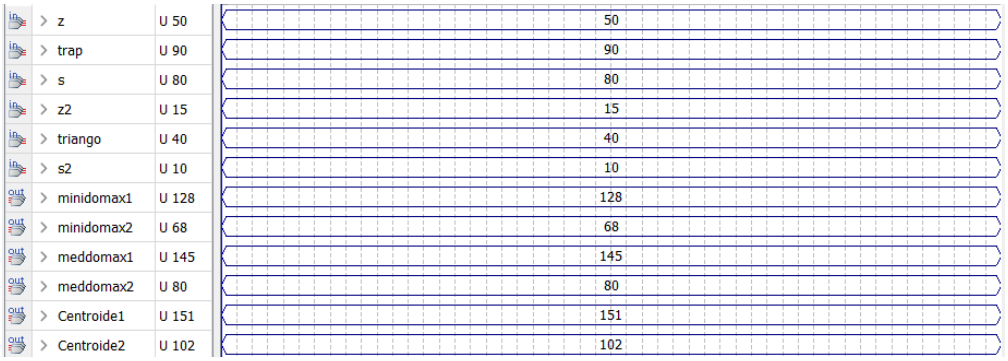
4.1.3 Módulos de funções de pertinência – saída e *Defuzzificação*.

Os módulos das funções de pertinência de saída e da *defuzzificação* serão analisados em conjunto devido à dificuldade e quantidade de valores a serem encontrados pelo modulo das funções, isso porque os resultados destas são 128 inteiros (a resolução é de 2, assim são encontrados valores de dois em dois, 0,2,4,...), exceto no caso do *Singleton.*

Para mostrar os resultados desses módulos foram analisados os seis tipos de função de pertinência com os módulos de *defuzzificação*, exceto o *singleton* que será analisado posteriormente. Foram definidos os valores de pertinência conforme a Figura 44, para o modulo criado é necessário somente inserir os valores de entrada, já para o *Matlab*, como não tem forma de inserir os valores, pelo *Fuzzy Logic Toolbox*, utilizou-se uma variável de entrada com seis funções de pertinência de entrada do tipo triangulo, assim foi adotado como valor de entrada 100, possibilitando que através da equação de reta que encontrássemos os valores de pertinência desejadas. Além disso utilizou-se duas variais de saída, com três funções cada, pois os módulos de *defuzzificação* só permite no máximo cinco funções de pertinência por variável.

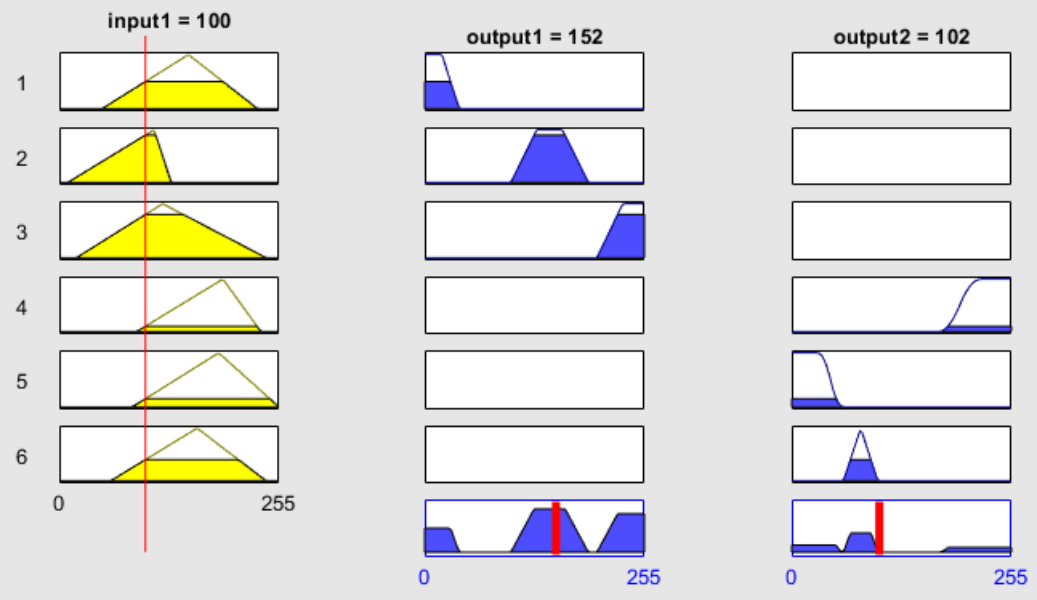
Os valores obtidos pelos três métodos dos módulos criados estão na Figura 44. Nas Figuras 45, 46 e 47 estão as respostas, encontradas no *MatLab*, para os métodos centroide, média dos máximos e mínimo dos máximos, respectivamente.

Figura 45. Resposta da análise das funções de pertinência e *defuzzificação.*



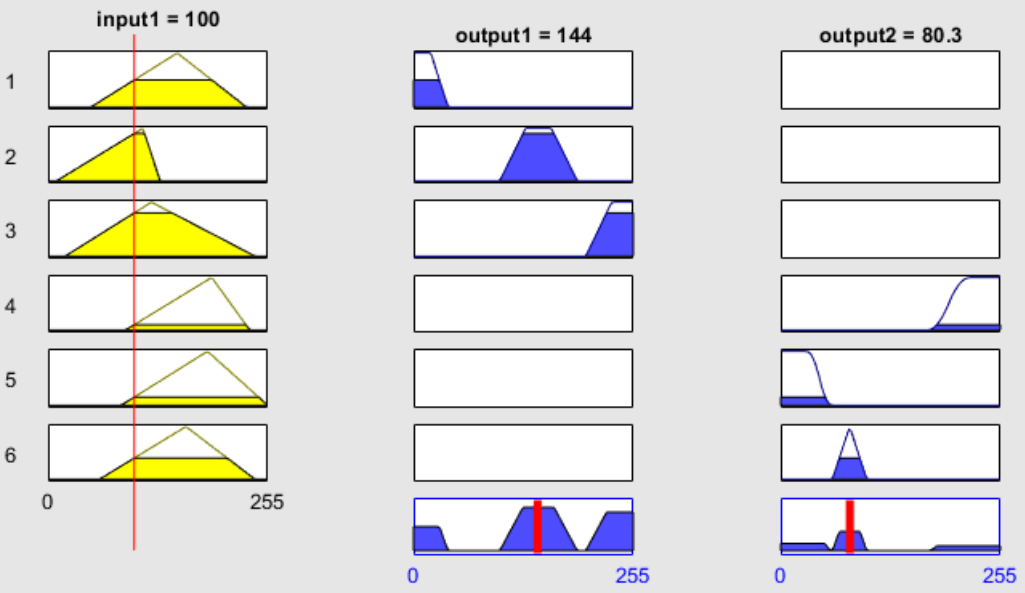
Fonte: Elaborado pelo autor (2023).

Figura 46. Resposta gráfica do método de *defuzzificação* Centroide.



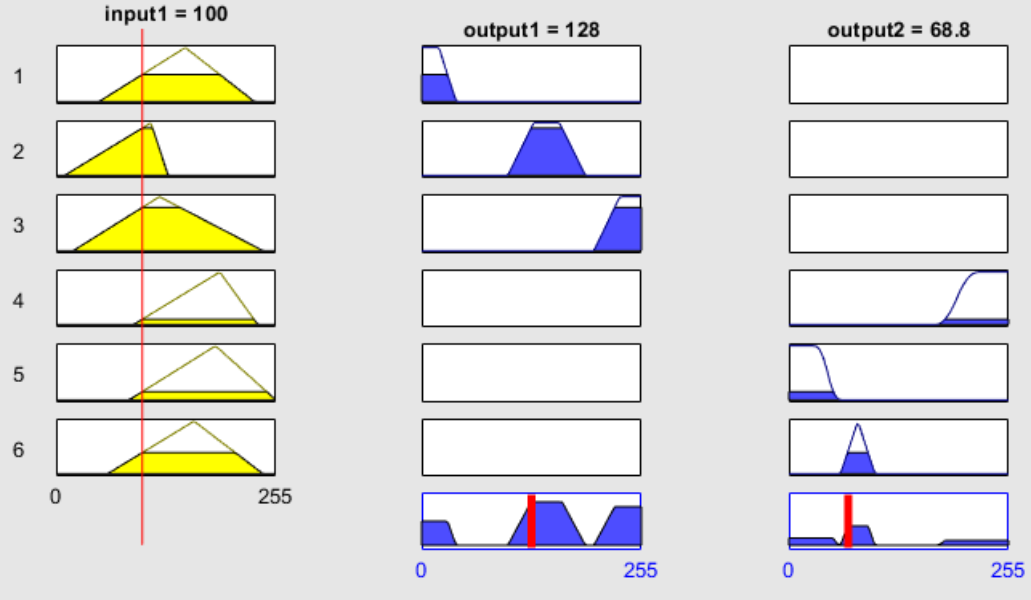
Fonte: Elaborado pelo autor (2022).

Figura 47. Resposta gráfica do método de *defuzzificação* Média dos máximos.



Fonte: Elaborado pelo autor (2022).

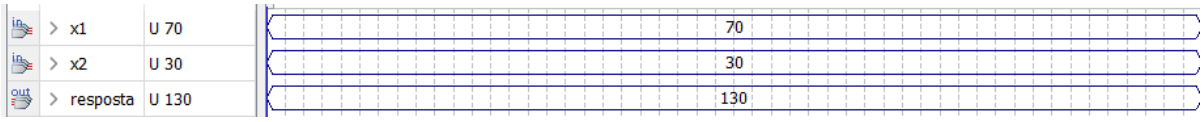
Figura 48. Resposta gráfica do método de *defuzzificação* Menor dos máximos.



Fonte: Elaborado pelo autor (2022).

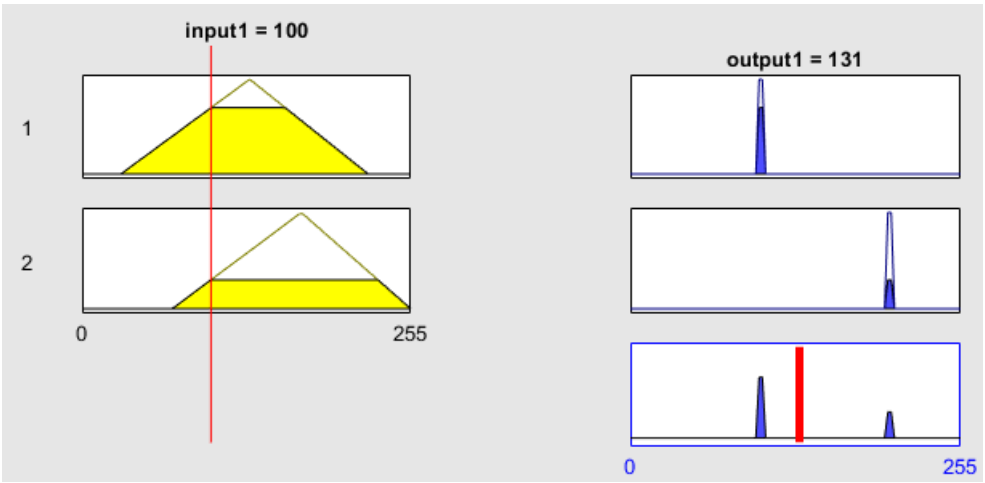
comparação para o *singlinton* foi realizada da mesma maneira que os outros métodos, os resultados do Modulo e do *MatLab* estão nas imagens 48 e 49, respectivamente.

Figura 49. Resposta do modulo *defuzzificação* com *Singleton.*



Fonte: Elaborado pelo autor (2023).

Figura 50. Resposta gráfica do método das alturas.



Fonte: Elaborado pelo autor (2022).

As Figuras comprovam a eficácia dos módulos analisados, pois o erro encontrado foi de 1 ponto, isso deve-se pelo fato de o *MatLab* utilizar ponto flutuante, enquanto não se utilizou nesse no projeto. Não se encontrou erro maior que 1 ponto nos diversos testes.

Os diferentes tipos de *Defuzzificação* utilizam quantidades de elementos lógicos diferentes, para observar essa constatação, criou-se o projeto mais básico (não consideraremos duas várias de entrada com uma função de pertinências cada e uma variável de saída com uma função de pertinências, pois não esse uso não é habitual), sendo na entrada duas variáveis com duas funções de pertinência cada e uma variável de saída com duas funções de pertinência.

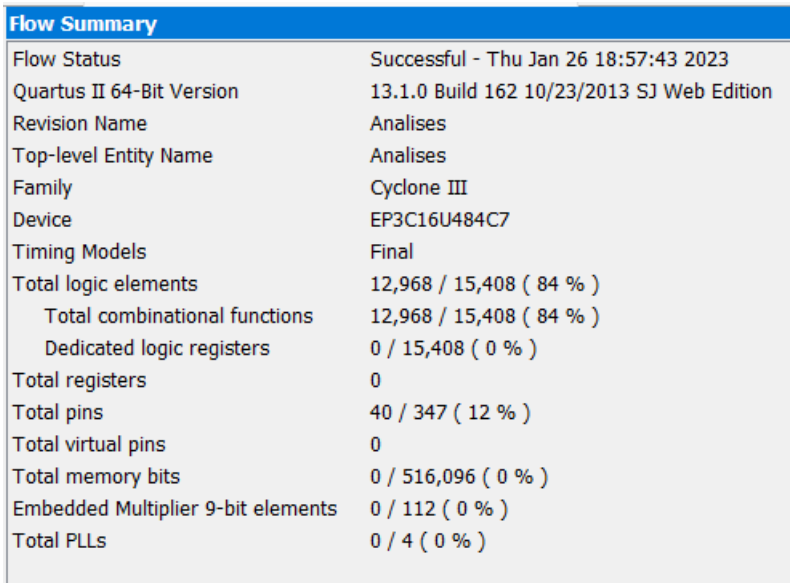
As Figuras 50, 51, 52 e 53 correspondem, respectivamente, a quantidade de unidade lógica do método das alturas com a variável *Singleton*, média dos máximos, centroide e mínimo dos máximos. Observa-se que o *Singleton* utiliza pelo menos três vezes menos unidades que os outros, o que o torna mais econômico e preferível a utilização. Os outros três métodos apresentam quantidades próximas.

Figura 51. Informações do projeto criado, utilizando o método das Alturas.

Flow Summary	
Flow Status	Successful - Thu Jan 26 18:39:31 2023
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	Analises
Top-level Entity Name	Analises
Family	Cyclone III
Device	EP3C16U484C7
Timing Models	Final
Total logic elements	3,875 / 15,408 (25 %)
Total combinational functions	3,875 / 15,408 (25 %)
Dedicated logic registers	0 / 15,408 (0 %)
Total registers	0
Total pins	40 / 347 (12 %)
Total virtual pins	0
Total memory bits	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

Fonte: Elaborado pelo autor (2023).

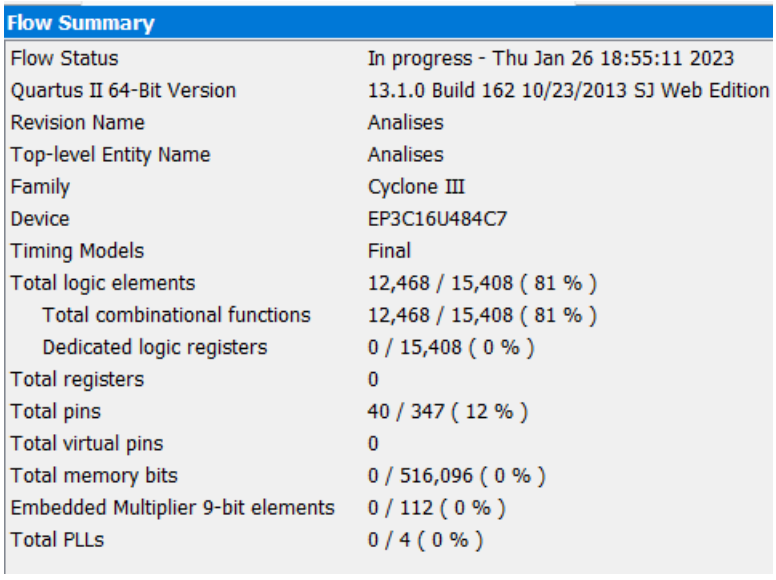
Figura 52. Informações do projeto criado, utilizando o método Média dos máximos.



Flow Summary	
Flow Status	Successful - Thu Jan 26 18:57:43 2023
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	Analises
Top-level Entity Name	Analises
Family	Cyclone III
Device	EP3C16U484C7
Timing Models	Final
Total logic elements	12,968 / 15,408 (84 %)
Total combinational functions	12,968 / 15,408 (84 %)
Dedicated logic registers	0 / 15,408 (0 %)
Total registers	0
Total pins	40 / 347 (12 %)
Total virtual pins	0
Total memory bits	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

Fonte: Elaborado pelo autor (2023).

Figura 53. Informações do projeto criado, utilizando o método Centroide.



Flow Summary	
Flow Status	In progress - Thu Jan 26 18:55:11 2023
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	Analises
Top-level Entity Name	Analises
Family	Cyclone III
Device	EP3C16U484C7
Timing Models	Final
Total logic elements	12,468 / 15,408 (81 %)
Total combinational functions	12,468 / 15,408 (81 %)
Dedicated logic registers	0 / 15,408 (0 %)
Total registers	0
Total pins	40 / 347 (12 %)
Total virtual pins	0
Total memory bits	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

Fonte: Elaborado pelo autor (2023).

Figura 54. Informações do projeto criado, utilizando o método Menor dos máximos.

Flow Summary	
Flow Status	Successful - Thu Jan 26 19:01:51 2023
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	Analises
Top-level Entity Name	Analises
Family	Cyclone III
Device	EP3C16U484C7
Timing Models	Final
Total logic elements	9,319 / 15,408 (60 %)
Total combinational functions	9,319 / 15,408 (60 %)
Dedicated logic registers	0 / 15,408 (0 %)
Total registers	0
Total pins	40 / 347 (12 %)
Total virtual pins	0
Total memory bits	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

Fonte: Elaborado pelo autor (2023).

Através das Figuras também concluímos que o mínimo de unidades logicas para a utilização do sistema *fuzzy* pelo projeto criado é de 3.875 unidades.

A Figura 54 é uma tabela retirada do documento, que mostra uma visão geral dos equipamentos da família *Cyclone III*, encontrada no site da Intel.

Figura 55. Tabela de informações dos dispositivos da família Cyclone III.

Family	Device	Logic Elements	Number of M9K Blocks	Total RAM Bits	18 x 18 Multipliers	PLLs	Global Clock Networks	Maximum User I/Os
Cyclone III	EP3C5	5,136	46	423,936	23	2	10	182
	EP3C10	10,320	46	423,936	23	2	10	182
	EP3C16	15,408	56	516,096	56	4	20	346
	EP3C25	24,624	66	608,256	66	4	20	215
	EP3C40	39,600	126	1,161,216	126	4	20	535
	EP3C55	55,856	260	2,396,160	156	4	20	377
	EP3C80	81,264	305	2,810,880	244	4	20	429
	EP3C120	119,088	432	3,981,312	288	4	20	531
Cyclone III LS	EP3CLS70	70,208	333	3,068,928	200	4	20	413
	EP3CLS100	100,448	483	4,451,328	276	4	20	413
	EP3CLS150	150,848	666	6,137,856	320	4	20	413
	EP3CLS200	198,464	891	8,211,456	396	4	20	413

Fonte: Altera Corporation (2009).

Com as informações da Figura 54, concluímos que os menores equipamentos (serão analisados os equipamentos da família *Cyclone III*) que podem-se utilizar para cada método são:

Singlinton: EP3C5

Média dos máximos: EP3C16

Centroide: EP3C16

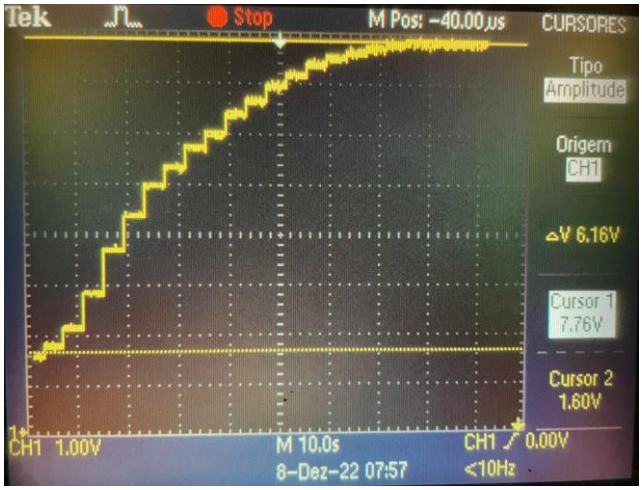
Menor dos máximos: EP3C10.

4.2 COMPARAÇÃO DOS RESULTADOS MATLAB E FPGA - APLICAÇÃO

Os resultados da aplicação do controlador *fuzzy* para um motor DC, descrito no item 6, serão apresentados nesse item, partiremos do repouso para 2.000 rpm (*switches* 111) utilizando o programa no *Simulink* e no FPGA para efeito de comparação e validação das bibliotecas.

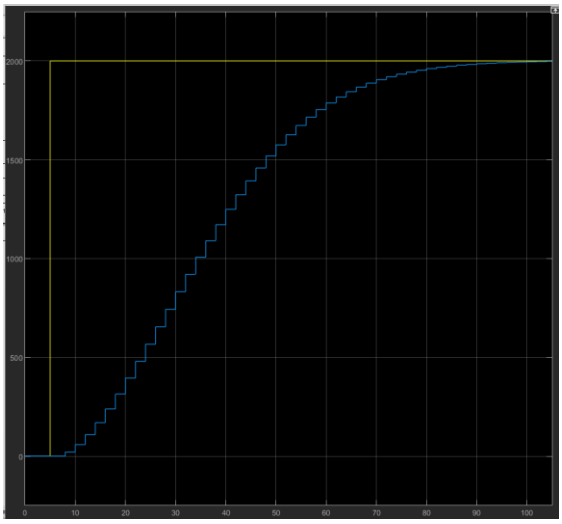
As respostas estão nas Figuras 56 e 57, sendo a primeira do FPGA e consequentemente a outra corresponde ao *MatLab.*

Figura 56. Resultado da tensão do motor controlado de velocidade 0 até 2.000 rpm, extraído de um osciloscópio.



Fonte: Elaborado pelo autor (2023).

Figura 57. Resultado encontrado do modelo de controlador, com a entrada variando de 0 até 2.000 rpm.

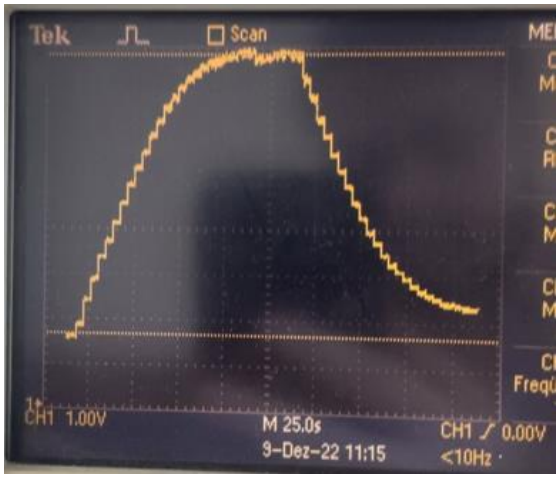


Fonte: Elaborado pelo autor (2022).

Observando as respostas, podemos concluir que os sinais possuem as mesmas características, que são o esperado de um controlador *PID-Fuzzy*. O motor saiu da tensão de 1,6V (0 rpm) e chegou após 900s chegou a 7,76V (2.000 rpm). As diferenças são mínimas a qual demostra que o projeto funciona de forma correta e consequentemente as bibliotecas criadas são confiáveis e podem ser usadas em futuros projetos.

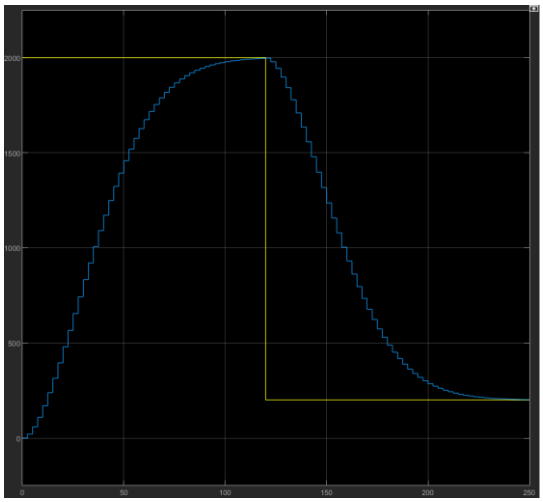
Outro teste realizado foi de partir da inercia do motor e instruir para que chegue aos 2.000 rpm e depois cair para 200 rpm, para observar o comportamento no momento de diminuição da velocidade, os resultados estão nas Figuras 58 e 59. O motor parte da tensão de 1,6V (0 rpm) e chegou após 900s chegou a 7,76V (2.000 rpm), depois decresce até uma tensão em torno de 2,4V (200 rpm). Este resultado confirma o pleno funcionamento do controlador, conforme esperado.

Figura 58. Resultado da tensão do motor controlado de velocidade 0 até 2.000 rpm e depois indo a 200 rpm, extraído de um osciloscópio.



Fonte: Elaborado pelo autor (2023).

Figura 59. Resultado encontrado do modelo de controlador com a entrada variando de 0 até 2.000 rpm e após indo para 200 rpm.



Fonte: Elaborado pelo autor (2022).

5 CONCLUSÕES

Os resultados obtidos neste estudo permitiram concluir que as bibliotecas *PACOTE_LPM_FUZZYI* e *PACOTE_FUNC_FUZZY1*, criadas para utilizar a logica *fuzzy* tipo 1 na linguagem de descrição VHDL e sua utilização no FPGA são funcionais, comprovando suas características e eficácia nos exemplos apresentados e na aplicação do controlado *PID-Fuzzy* para um motor DC. Os resultados quando comparados ao *Matlab* apresentam erros mínimos, principalmente polo motivo do ponto flutuante, a qual podem ser desprezíveis, tendo assim alta confiabilidade.

Foram implementadas nas bibliotecas uma grande variedade de utilização e criação da lógica *fuzzy* pelo usuário, as principais formas para cada etapa com abrangida pelo escopo desse projeto.

Como sugestão para trabalhos futuros, pode-se incrementar a biblioteca a possibilidade de utilizar como base de palavras diferentes de 8 bits, como por exemplo 12 bits, utilizando o *"generic"* e outras ferramentas da linguagem de descrição VHDL. Outra oportunidade é criar um LPM para a logica *fuzzy* tipo 2.

REFERÊNCIAS

ALTERA CORPORATION. **Cyclone III device handbook**. San Jose: Altera, 2009. v.1.

BALDWIN J. F. **Fuzzy logic**, New York: Wiley, 1996.

CHU, Pong P. **FPGA prototyping by VHDL examples:** Xilinx Spartan. New Jersey: John Wiley & sons inc., 2008. v. 3.

IEEE COMPUTER SOCIETY**. IEEE standard for VHDL language reference manual**. New York: IEEE, 2019.

JAMSSHIDI M.; VADIEE N.; ROSS T. J. **Fuzzy logic and control**: software and hardware application. New Jersey: Pearson, 1993.

TANAKA K. **An introduction to fuzzy logic for practical application**. Berlin: Springer-Verlag, 1996.

KLIIR, G. J.; YUAN, B. **Fuzzy sets and fuzzy logic:** theory and applications. New Jersey: Pearson Prentice Hall: Upper Saddle River, 1995.

LOFTI A. Zadeh; J. Kacprzyk. **Fuzzy logic for the management of uncertainty**. Wiley-Interscience, 1992.

MENDEL, Jerry M. **Fuzzy logic systems for engineering:** a tutorial. New York: IEEE. v. 83, n. 3, p. 345-377, 1995.