

GUSTAVO ADOLFO ROHM DE MARCO

**Desenvolvimento de Módulos Parametrizáveis de Processador *Fuzzy* Visando o Projeto
de Aplicações Embarcadas em FPGA**

Guaratinguetá - SP
2016

GUSTAVO ADOLFO ROHM DE MARCO

Desenvolvimento de Módulos Parametrizáveis de Processador *Fuzzy* Visando o Projeto de Aplicações Embarcadas em FPGA

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

Orientador (a): Prof. Dr. Leonardo Mesquita

Guaratinguetá - SP
2016

M321d Marco, Gustavo Adolfo Rohm de
Desenvolvimento de módulos parametrizáveis de processador Fuzzy
visando o projeto de aplicações embarcadas em FPGA / Gustavo Adolfo
Rohm de Marco – Guaratinguetá, 2017.
45 f : il.
Bibliografia: f. 44-45

Trabalho de Graduação em Engenharia Elétrica – Universidade
Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2017.
Orientador: Prof. Dr. Leonardo Mesquita

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

CDU 510.6

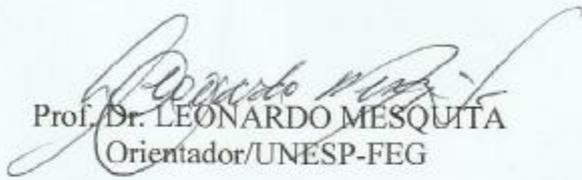
GUSTAVO ADOLFO ROHM DE MARCO

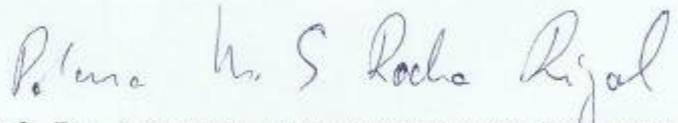
ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO
PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE
"GRADUADO EM ENGENHARIA ELÉTRICA"

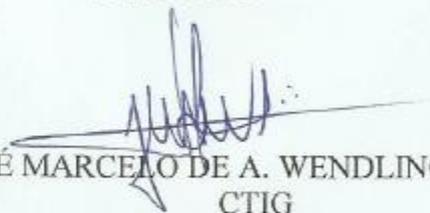
APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE
GRADUAÇÃO EM NOME DO CURSO

Prof. Dr. LEONARDO MESQUITA
Coordenador

BANCA EXAMINADORA:


Prof. Dr. LEONARDO MESQUITA
Orientador/UNESP-FEG


Profa. Dra. PALOMA MARIA SILVA ROCHA RIZOL
UNESP-FEG


Eng. JOSÉ MARCELO DE A. WENDLING JR.
CTIG

Dezembro de 2016

Dedico este trabalho de modo especial, ao meu pai Francisco, à minha mãe Cleide, à minha irmã Giovanna, aos meus companheiros do ABC *Rugby* e a todos os amigos que participaram e contribuíram com minha trajetória até aqui.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, fonte da vida e da graça. Agradeço pela minha vida, minha inteligência, minha família e meus amigos,

ao meu orientador, *Prof. Dr. Leonardo Mesquita* que jamais deixou de me incentivar. Sem a sua orientação, dedicação e auxílio, o estudo aqui apresentado seria praticamente impossível.

aos meus pais *Francisco e Cleide*, que apesar das dificuldades enfrentadas, sempre incentivaram meus estudos.

Aos meus companheiros do *ABC Rugby*, por toda paciência e compreensão nas ausências nos treinos durante todos esses anos.

às funcionárias da Biblioteca do Campus de Guaratinguetá pela dedicação, presteza e principalmente pela vontade de ajudar,

aos funcionários da Faculdade de Engenharia do Campus de Guaratinguetá pela dedicação e alegria no atendimento.

RESUMO

Este trabalho relata as etapas de desenvolvimento de uma biblioteca de módulos virtuais parametrizáveis que poderão ser usados na implementação de processadores baseados na Lógica *Fuzzy* implementado em VHDL para síntese em um componente FPGA. Inicialmente foi feita um breve estudo acerca da Lógica *Fuzzy*, sendo abordados os principais conceitos acerca dessa teoria, incluindo uma breve descrição sobre conjuntos e regras *fuzzy*, bem como a regra de inferência *fuzzy* Mandani, método utilizado no desenvolvimento do trabalho. Além disso, é citado os métodos que serão utilizados para a implementação dos algoritmos e uma breve abordagem sobre a tecnologia FPGA utilizada no desenvolvimento do projeto. Por fim, relata-se sobre os resultados alcançados com a parametrização dos blocos essenciais de um controlador *fuzzy* Mandani e um modelo *template*, que poderá ser utilizado por projetistas que assim desejam utilizar um controlador Mandani em seu processo, assim, tornando mais acessível à utilização do controle inteligente.

PALAVRAS-CHAVE: Lógica *Fuzzy*. Controlador *Fuzzy*. Controlador Mandani. *FLC*. Processador *Fuzzy*. *FPGA*. *VHDL*.

ABSTRACT

This work reports the development stages of a library of virtual modules that can be used in the implementation of Fuzzy Logic-based processors implemented in VHDL for synthesis in an FPGA component. First, a brief study about the Fuzzy Logic was made, addressing the main concepts about this theory, including a brief description of fuzzy sets and rules, as well as the Mandani fuzzy inference rule, method used in the development of the work. In addition, it is mentioned the methods that will be used for the implementation of the algorithms and a brief approach on the FPGA technology used in the development of the project. Finally, we report on the results achieved with the parameterization of the essential blocks of a Mandani fuzzy controller and a template model, which can be used by designers who wish to use a Mandani controller in their process, thus making it more accessible the use of intelligent control.

KEYWORDS: Fuzzy Logic. Fuzzy Logic Controller. Mandani Controller. FLC. Fuzzy Processor. FPGA. VHDL.

LISTA DE ILUSTRAÇÕES

Figura 1 – Funções de pertinência e difusa respectivamente	17
Figura 2 – Função Pertinência do tipo Trapezoidal.....	18
Figura 3 – Função pertinência do tipo triangular	19
Figura 4 – Função pertinência do tipo Z	19
Figura 5 – Função pertinência do tipo S.....	20
Figura 6 – Funções de pertinência do tipo <i>singleton</i>	21
Figura 7 - Blocos de um controlador <i>fuzzy</i>	21
Figura 8 – Fuzificação das variáveis linguísticas de entrada.....	22
Figura 9 – Sistema de inferência Mamdani	24
Figura 10 – Exemplo do método de defuzificação centro de área.....	24
Figura 11 – Método de defuzificação por média ponderada	25
Figura 12 – Janela de trabalho Altera Quartus II 9.1.....	26
Figura 13 – Placa de desenvolvimento Altera DE0.....	27
Figura 14 – Algoritmo do bloco de fuzificação.....	30
Figura 15 – Função de pertinência trapezoidal e pontos característicos	30
Figura 16 – Código VHDL do cálculo das inclinações das retas	31
Figura 17 – Código VHDL, bloco fuzificador	31
Figura 18 – <i>Template</i> utilizado para determinar as funções pertinência	32
Figura 19 – Função de pertinência trapezoidal e áreas das funções pertinência	33
Figura 20 – Algoritmo do bloco de inferência	33
Figura 21 – Código VHDL do bloco de inferência	34
Figura 22 – Algoritmo da base de regras.....	34
Figura 23 – Código VHDL das funções de máximo e mínimo.....	35
Figura 24 – <i>Template</i> utilizado para adicionar e modificar a base de regras	35
Figura 25 – Função de pertinência do defuzificador	36
Figura 26 – <i>Template</i> dos valores das saídas <i>singleton</i>	36
Figura 27 – Código VHDL do bloco defuzificador, cálculo das saídas	37
Figura 28 – Código VHDL do bloco defuzificador, entradas do bloco	37
Figura 29 – <i>Template</i> do bloco defuzificador.....	38
Figura 30 – Resultado da simulação do fuzificador	39
Figura 31 – Resultado de simulação do bloco de inferência	40
Figura 32 – Resultados da simulação do bloco defuzificador.....	41

LISTA DE TABELAS

Tabela 1 – Descrição das entradas e saídas do bloco fuzificador.....	38
Tabela 2 – Valores de Entrada e Resultados esperados para o teste do fuzificador.....	39
Tabela 3 – Descrição das entradas do bloco de inferência.....	39
Tabela 4 – Valores de entrada e resultados para o bloco de inferência.....	40
Tabela 5 – Descrição da entrada do bloco de defuzificação.....	40
Tabela 6 – Valores de entrada do bloco defuzificador.....	41
Tabela 7 – Entradas do processador <i>fuzzy</i>	41
Tabela 8 – Valores de entrada do processador <i>fuzzy</i>	42

SUMÁRIO

1	INTRODUÇÃO.....	14
1.1	MOTIVAÇÃO	14
1.2	OBJETIVO.....	14
1.3	ESTRUTURA DO TRABALHO.....	15
2	FUNDAMENTAÇÃO TEÓRICA	16
2.1	LÓGICA <i>FUZZY</i>	16
2.1.1	Conjuntos <i>Fuzzy</i>.....	17
2.1.2	Funções de Pertinência.....	17
2.1.1.1	Função Trapezoidal	18
2.1.1.2	Função Triangular.....	18
2.1.1.3	Função Pertinência Tipo <i>Z</i>	19
2.1.1.4	Função Pertinência do tipo <i>S</i>	20
2.1.1.5	Função pertinência do tipo <i>Singleton</i>	20
2.2	CONTROLADOR <i>FUZZY</i>	21
2.2.1	Fuzificação.....	22
2.2.2	Máquina de Inferência	22
2.2.2.1	Base de Regras.....	23
2.2.2.2	Inferência	23
2.2.3	Defuzificação	24
2.2.3.1	Método Centro de Área	24
2.2.3.2	Método da Média Ponderada	25
2.3	FERRAMENTAS UTILIZADAS.....	25
2.3.1	Altera Quartus II 9.1	26
2.3.2	Placa de Desenvolvimento	27
3	ESPECIFICAÇÃO E PROJETO	29
3.1	BLOCO DE FUZIFICAÇÃO	29
3.2	BLOCO DA MÁQUINA DE INFERÊNCIA.....	32
3.2.1	Inferência.....	32
3.2.2	Base de Regras	34
3.3	BLOCO DE DEFUZIFICAÇÃO	35
4	SIMULAÇÕES E RESULTADOS	38

4.1	SIMULAÇÃO DO FUZIFICADOR	38
4.2	SIMULAÇÃO DA MÁQUINA DE INFERÊNCIA	39
4.3	SIMULAÇÃO DO BLOCO DE DEFUZIFICAÇÃO	40
4.4	SIMULAÇÃO DO PROCESSADOR <i>FUZZY</i> COMPLETO	41
5	CONCLUSÃO	43
	REFERÊNCIAS	44

1 INTRODUÇÃO

As teorias de controle comumente estudadas hoje são a teoria de controle clássico, teoria de controle moderno e a teoria de controle robusta. A teoria de controle moderna, tratada neste trabalho, mais precisamente o controle *fuzzy*, vem ganhando muito espaço na indústria, onde apresentam plantas complexas de se modelar matematicamente, impossibilitando ou tornando muito complexo a utilização do controle clássico. Também vale destacar o uso da lógica *fuzzy* em combinação com a neuro-computação e algoritmos genéticos, que são os principais constituintes de uma nova área de pesquisa denominada de *soft computing* que, ao contrário da computação tradicional, explora a imprecisão do mundo real. Ela agrega ao sistema de controle características inteligentes, ou seja, capazes de fornecer respostas que solucionam problemas, mesmo em situações novas ou inesperadas.

O uso de sistemas construídos utilizando esta tecnologia é especialmente interessante quando o modelo matemático está sujeito a incertezas (SANDRI; CORREA, 1999).

1.1 MOTIVAÇÃO

A principal motivação para o desenvolvimento deste trabalho se deu devido ao aumento da utilização da lógica *fuzzy*, que vai desde produtos de consumo, como câmeras, máquinas de lavar e micro-ondas, até aplicações industriais mais complexas, como controle de processo específico em reatores nucleares e mísseis balísticos. Também vale destacar que a criação da biblioteca de módulos virtuais, torna possível o projetista utilizar o controlador *fuzzy* para diversas aplicações, apenas modificando os dados parametrizáveis, como quantidade de funções de pertinência, número de entradas e saídas e tipo de função de pertinência, como função S, função Z, *singleton*, triangular e trapezoidal.

1.2 OBJETIVO

Este trabalho tem como objetivo o desenvolvimento em VHDL de uma biblioteca virtual de módulos parametrizáveis, como fuzificador, defuzificador e máquina de inferência utilizando a metodologia Mandani, necessários para a implementação de um processador baseado em lógica *fuzzy*, para posteriormente implementar em um componente programável FPGA.

1.3 ESTRUTURA DO TRABALHO

O conteúdo deste trabalho está dividido em cinco capítulos. O primeiro apresenta uma breve introdução ao tema.

O capítulo 2 apresenta conceitos básicos da lógica *fuzzy*, sobre o *software* utilizado para o desenvolvimento do projeto em VHDL, e também o componente FPGA e a placa de desenvolvimento utilizado para a implementação do controlador lógico.

O capítulo 3 apresenta a definição de Controlador Lógico *Fuzzy* (FLC) e algumas arquiteturas de processador implementadas em digital.

O capítulo 4 discute os módulos desenvolvidos em VHDL, exibindo alguns algoritmos criados baseados nos conceitos de lógica *fuzzy*, suas especificações e os resultados obtidos com o processador.

Por fim, o capítulo 5 encerra o trabalho, apresentando os blocos lógicos desenvolvidos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 LÓGICA FUZZY

As primeiras noções de lógica *fuzzy* foram desenvolvidas por Jan Lukasiewicz, um lógico polonês, em 1920 que introduziu conjuntos com graus de pertinência, que diferentemente da Lógica Booleana que admite apenas valores booleanos, ou seja, verdadeiro ou falso, a lógica difusa ou *fuzzy*, trata de valores que variam entre 0 e 1. Assim, uma pertinência de 0.5 pode representar meio verdade, logo 0.9 e 0.1, representam quase verdade e quase falso, respectivamente (SILVA, 2005).

A primeira publicação sobre lógica *fuzzy* foi feita por Zadeh em 1965, professor em Berkeley, Universidade da Califórnia. Ele combinou os conceitos de lógica clássica com os conjuntos de Lukasiewicz, definindo graus de pertinência. A mesma surgiu da constatação de que quando maior a complexidade de um sistema, nossa habilidade de conclusão e de tomar decisões precisas e significativas, tende a diminuir até um ponto onde a precisão e relevância passa a ser características quase excludentes (ZADEH, 1973).

A estrutura proposta por Mamdani (1975) de universo de discurso, variáveis linguísticas, fuzificação, banco de regras, inferência *fuzzy* e sistema de defuzificação para um controlador *fuzzy*, quando bem assimilada, é uma arma poderosa de simplificação e aumento da velocidade de processamento e robustez do mesmo, possibilitando decisões rápidas e coerentes num ambiente de incertezas. Ainda em 1975, Mamdani publicou um trabalho denominado “Um experimento em Síntese Linguística com um Controlador *Fuzzy*”, descrevendo o controle de uma máquina a vapor com aplicação de lógica difusa, sendo considerada uma das primeiras aplicações, com sucesso, de um controlador *fuzzy*, voltando o mundo científico para essa tecnologia.

Após 1980, o Japão iniciou seu uso com aplicações na indústria. Algumas das primeiras aplicações foram em um tratamento de água feito pela *Fuji Electric* em 1983 e pela *Hitachi* em um sistema de metrô inaugurado em 1987. Por volta de 1990, devido ao desenvolvimento e as inúmeras possibilidades práticas dos sistemas *fuzzy* e o grande sucesso comercial de suas aplicações, despertou um maior interesse em empresas dos Estados Unidos em desenvolver pesquisas e tecnologia baseado em inteligência artificial.

A lógica *fuzzy* possibilita a implementação de implicações lógicas semelhantes às utilizadas pelo indivíduo. Busca modelos capazes de representar sua percepção da realidade,

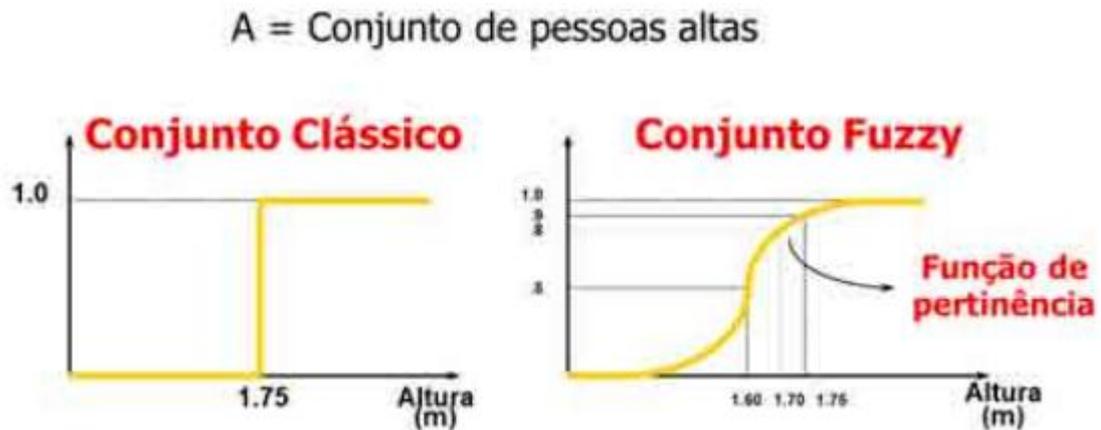
incluindo assim suas características na realização de inferências e tomadas de decisões. Diferentemente, a abordagem convencional propõe a criação de modelos da própria realidade.

2.1.1 Conjuntos *Fuzzy*

Na lógica booleana ou clássica, como também é chamada, os conjuntos é sempre bem definidos, ou seja, o elemento necessariamente pertencerá ou não a um conjunto definido. Portanto a pertinência neste caso é definida com somente dois valores: 1 ou 0.

Quando utilizamos a lógica *fuzzy*, seu conjunto, diferentemente da booleana, possui vários graus de pertinência, com isso quanto um elemento pode pertencer a um conjunto pode variar de zero a um. Tomando como exemplo a estatura de uma pessoa, na lógica difusa, as incertezas que definem uma pessoa alta ficariam representadas por um grau de pertinência, como mostrado na Figura 1.

Figura 1 – Funções de pertinência e difusa respectivamente



Fonte: (WEBER, 2003).

A teoria de conjuntos *fuzzy* tenta traduzir a informação imprecisa em forma de variáveis linguísticas. Tomando como exemplo a altura de uma pessoa os conjuntos de termos linguísticos que poderiam ser utilizado seriam: baixa, média, alta e muito alta, cada um deles definidos por um conjunto *fuzzy*.

2.1.2 Funções de Pertinência

A função de pertinência é, essencialmente, uma curva distribuída com graus de pertinência que melhor representam as propriedades semânticas de uma palavra ou expressão

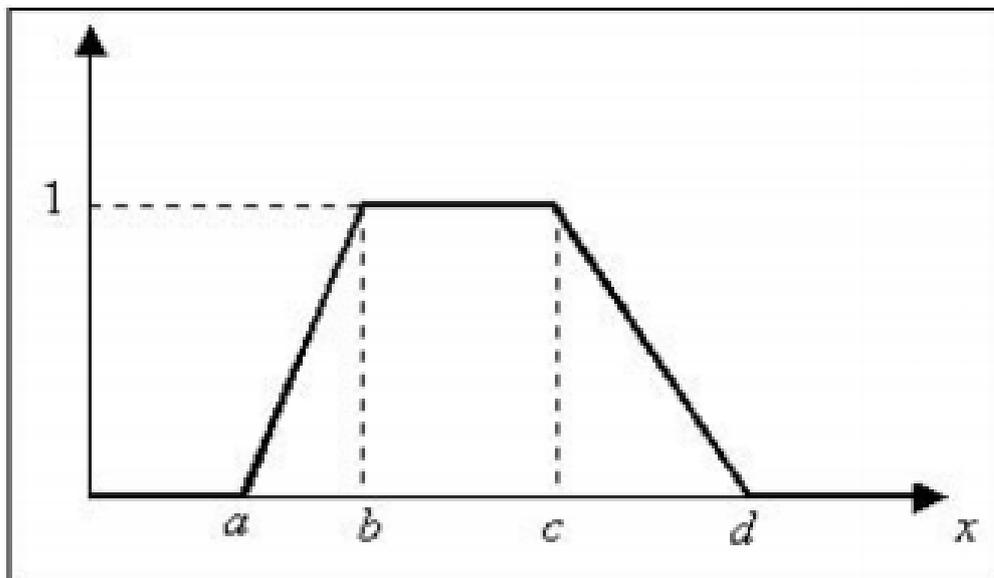
linguística. Tais funções podem assumir qualquer formato, mas não se pode esquecer que tal representação deve traduzir o resultado mais fiel possível da realidade do fenômeno físico modelado e como tal, deve ser sintonizado empiricamente (SILVEIRA, 2004).

Existem vários tipos de funções que podem ser utilizadas, sendo que os tipos a serem aplicados sempre dependem de como o sistema físico se comporta.

2.1.1.1 Função Trapezoidal

A curva trapezoidal é uma função de “ x ”, e depende de quatro parâmetros escalares, os pontos $\{a, b, c, d\}$ como mostrado na Figura 2.

Figura 2 – Função Pertinência do tipo Trapezoidal



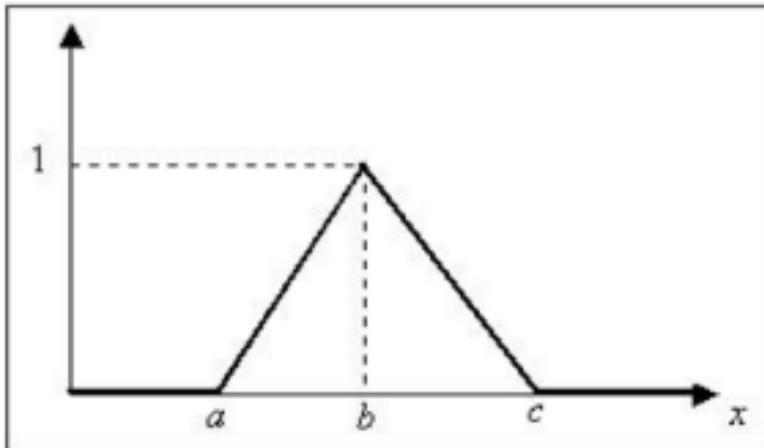
Fonte: (CREMASCO, 2010).

Os pontos $\{a, d\}$ representam os valores cuja pertinência é zero, enquanto os pontos $\{b, c\}$ correspondem aos valores cuja pertinência é igual a 1.

2.1.1.2 Função Triangular

A curva triangular é uma função de “ x ”, e depende de três parâmetros escalares, os pontos $\{a, b, c\}$ como mostrado na Figura 3.

Figura 3 – Função pertinência do tipo triangular



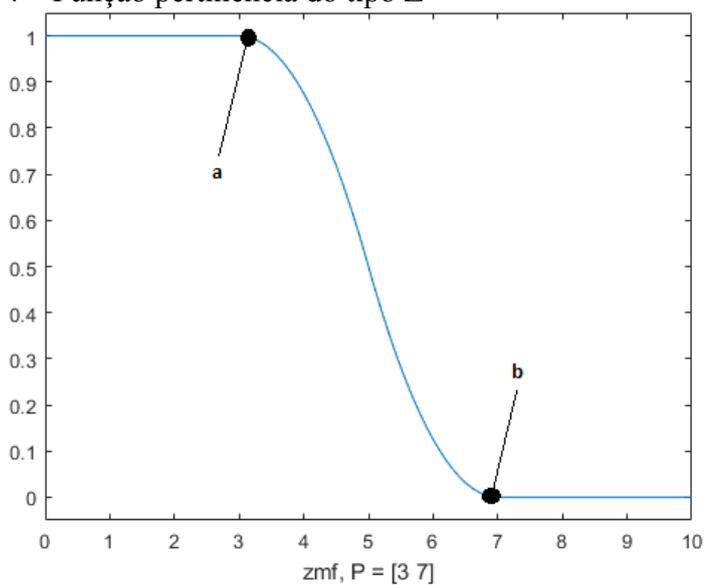
Fonte: (CREMASCO, 2010).

Os pontos $\{a, c\}$ representam os valores cuja pertinência é zero, enquanto os pontos $\{b\}$ corresponde ao valor cuja pertinência é igual a 1.

2.1.1.3 Função Pertinência Tipo Z

Esta função de pertinência de “x”, que possui a forma de Z é definida por dois parâmetros, os pontos $\{a, b\}$ que correspondem aos dois extremos da inclinação da curva, como mostrado na Figura 4.

Figura 4 – Função pertinência do tipo Z



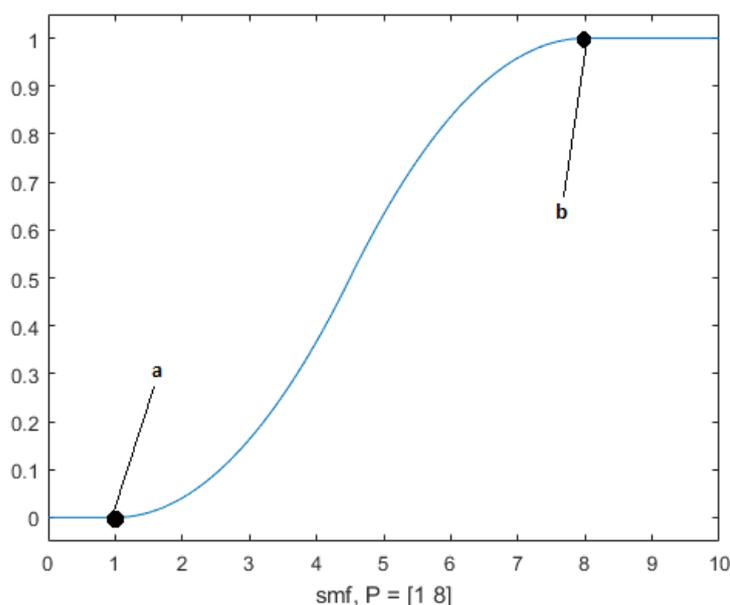
Fonte: Adaptado de MATHWORKS.

Para $x \leq a$, a pertinência será igual a 1, já para $x \geq b$, a pertinência será igual a zero. A inclinação da curva pode ser aproximada para uma reta, o que faria a função Z ser um caso especial da função trapezoidal.

2.1.1.4 Função Pertinência do tipo S

Esta função de pertinência de “x”, que possui a forma de S é definida por dois parâmetros, os pontos {a, b} que correspondem aos dois extremos da inclinação da curva, como mostrado na Figura 5.

Figura 5 – Função pertinência do tipo S



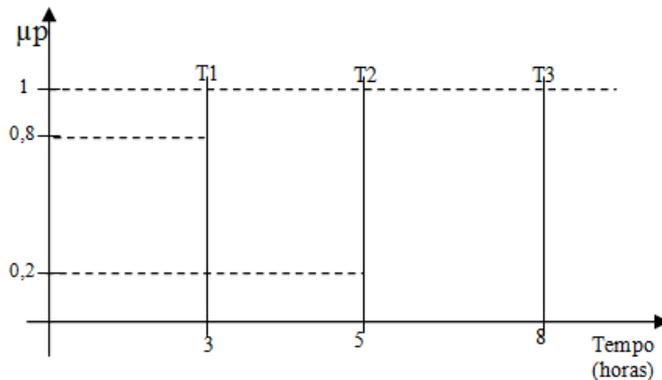
Fonte: Adaptado de MATHWORKS.

Para $x \leq a$, a pertinência será igual a zero, já para $x \geq b$, a pertinência será igual a 1. Assim como na função do tipo Z, a inclinação da curva pode ser aproximada para uma reta, o que faria a função S ser um caso especial da função trapezoidal.

2.1.1.5 Função pertinência do tipo Singleton

Esta função de pertinência, também denominada de função impulso, é definida por somente um valor pontual {a} que possui pertinência 1, como é representado na Figura 6 pelos pontos T1, T2 e T3.

Figura 6 – Funções de pertinência do tipo *singleton*



Fonte: (ASCARI, 2012).

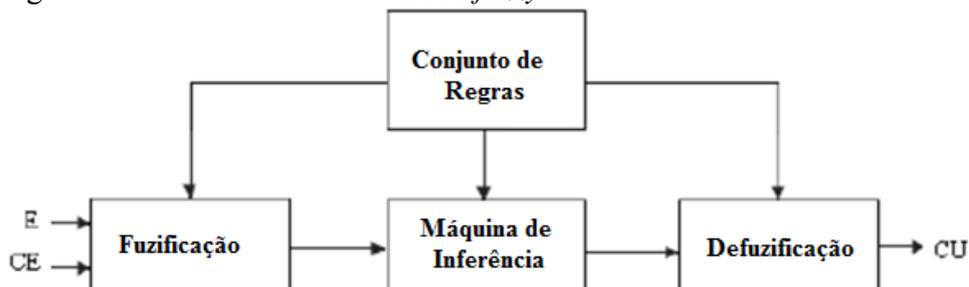
2.2 CONTROLADOR FUZZY

O controle clássico se utiliza de modelos matemáticos, muitas vezes complexos descritos por equações diferenciais, enquanto o controlador nebuloso é uma função não linear, onde as variáveis de entrada e saída se relacionam através da base de conhecimento que os especialistas têm sobre o processo (GRILLO, 2012).

O controlador baseado na lógica *fuzzy* não necessita de um modelo analítico da planta que deseja controlar. Segundo Campos e Saito (2004), o controlador *fuzzy* deve ser utilizado quando o processo não é linear e de uma complexidade alta, sendo assim muito difícil sua modelagem matemática, e além do mais ele deve ser simples, robusto e adquirir respostas em tempo real.

O controlador *fuzzy* pode ser representado por três blocos principais, sendo estes o fuzzificador, máquina de inferência e defuzzificador. Sua estrutura mais simples é apresentada na Figura 7 e, em seguida, é feita uma análise de cada bloco.

Figura 7 - Blocos de um controlador *fuzzy*

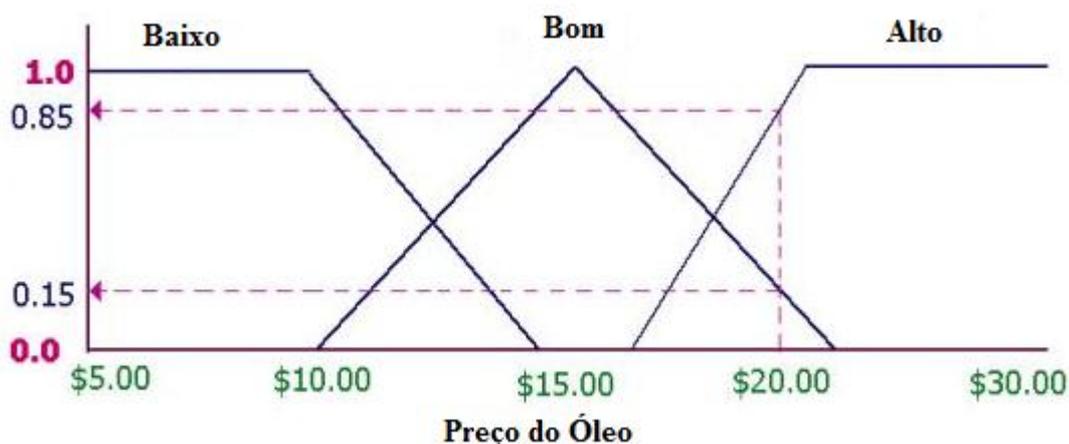


Fonte: Adaptado de BENNASSAR (2013).

2.2.1 Fuzificação

A etapa de fuzificação é responsável em transformar os valores numéricos do domínio real, proveniente de sensores ou de dispositivos computadorizados de entrada, em valores linguísticos no domínio *fuzzy*. Através dos conjuntos *fuzzy*, criados se baseando no conhecimento prévio de um especialista do processo, cada entrada é normalizada, dentro do universo de discurso criado, e associado a um grau de pertinência. A Figura 8 exemplifica um caso de fuzificação.

Figura 8 – Fuzificação das variáveis linguísticas de entrada



Fonte: Adaptado de MOHAGHEGH (2000).

Os conjuntos *fuzzy* neste exemplo são classificados quanto ao preço do petróleo. A variável linguística “Baixo” é representada por uma função de pertinência tipo Z, e possui pertinência 1 no intervalo \$5.00 a \$10.00, e pertinência zero quando maior ou igual a \$15.00. A variável “Bom” é representada por uma função pertinência triangular, onde possui pertinência 1 no ponto \$15.00, e pertinência zero quando menor de \$10.00 e maior que \$20.00. Por último a variável “Alto” é representada por uma função pertinência do tipo S.

Os valores 0.85 e 0.15 do gráfico mostram a pertinência para os conjuntos Alto e Bom respectivamente, quando o valor do petróleo é \$20.00.

2.2.2 Máquina de Inferência

O processo de inferência consiste em verificar o grau de compatibilidade entre a entrada e saída do controlador *fuzzy*. Esta verificação é feita através da avaliação de uma base de regras baseado no conhecimento do processo a ser controlado.

2.2.2.1 Base de Regras

Representam a relação entre a entrada e a saída do controlador *fuzzy*, onde a confiabilidade da saída, quanto a sua representatividade no modelo físico aplicado, depende diretamente da base de regras utilizada. As preposições *fuzzy* são descritas na forma de agregações e composições, definidas respectivamente pelas parcelas SE e ENTÃO da sentença, podendo ser descrita na forma **SE** <condição> **ENTÃO** <conclusão>.

2.2.2.2 Inferência

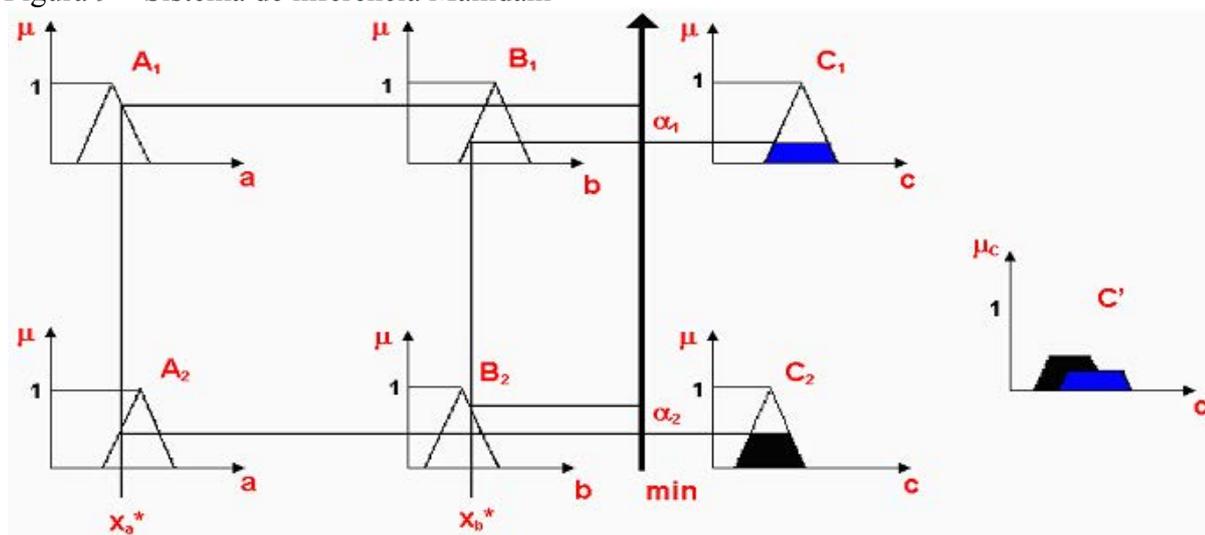
O processo de inferência consiste em verificar o grau de compatibilidade entre a entrada e a saída, através de um conjunto de regras. Na literatura existem alguns diferentes modelos de inferência *fuzzy*, como Mamdani, Takagi-Sugeno, neste trabalho será abordado o método de Mamdani.

O método de inferência em questão é definido por um conjunto de funções de máximo e mínimo, como ela este método também ficou conhecido, aplicados na base de regras. A estrutura da preposição *fuzzy* é dado por **SE** <condição 1> **E** <condição 2> **ENTÃO** <conclusão>, onde “E” é um operador lógico *fuzzy* que representa a função mínimo entre duas condições. Na inferência também apresenta o operador “OU” implícito, aplicado em regras sucessivas que agem sobre uma mesma saída. Assim, em termos gerais, a validação das regras são apresentadas pela Equação 1.

$$\mathbf{IF} (x_1 \text{ is } A_1 \mathbf{AND} y_1 \text{ is } B_1) \mathbf{OR} (x_2 \text{ is } A_2 \mathbf{AND} y_2 \text{ is } B_2) \mathbf{THEN} z \text{ is } C \quad (1)$$

Neste modelo de proposição tem-se que x_i corresponde a um valor de entrada, A_i corresponde a um conjunto *fuzzy* que representa um antecedente da regra, y_i corresponde à saída do controlador e B_i corresponde um conjunto *fuzzy* que representa o consequente da regra. Toda regra será avaliada para uma determinada entrada, assim obtendo um valor que será aplicado ao conjunto *fuzzy* da variável. O método e o raciocínio utilizado por Mamdani pode ser observado na Figura 9.

Figura 9 – Sistema de inferência Mamdani



Fonte: (SANTIAGO, 2008).

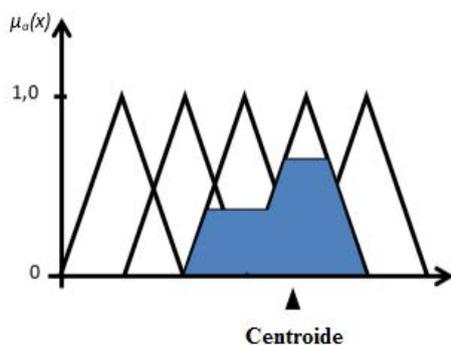
2.2.3 Defuzificação

Este bloco é responsável por transformar os dados *fuzzy*, oriundos do bloco de inferência, em dados no domínio real para serem interpretados e utilizados em sistemas de controle. Existem vários métodos de defuzificação, porém será abordado os mais comuns e também o método utilizado neste trabalho.

2.2.3.1 Método Centro de Área

Método mais utilizado em defuzificação, também conhecido como método do centroide, que obtém o ponto onde se divide o conjunto agregado ao meio. Um exemplo do centroide é dado pela Figura 10.

Figura 10 – Exemplo do método de defuzificação centro de área



Fonte: Adaptado de SANTIAGO (2008).

A equação para o cálculo do centro de massa é dado pela Equação 2.

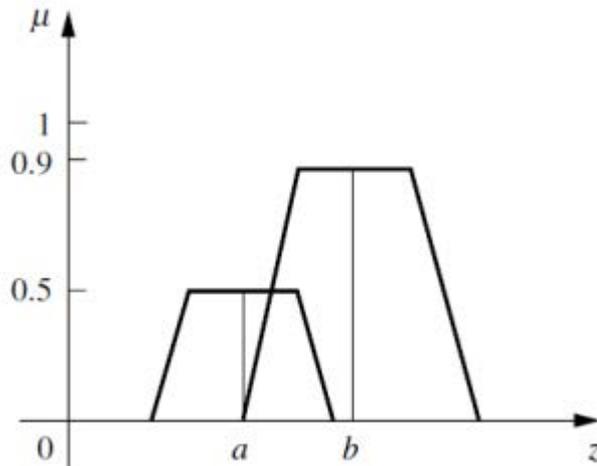
$$COG = \frac{\sum_{x=a}^b \mu(x).x}{\sum_{x=a}^b \mu(x)} \quad (2)$$

Onde b é o número de regras ativas, $\mu(x)$ é a área de uma função pertinência e x é a posição do centroide de cada função.

2.2.3.2 Método da Média Ponderada

De acordo com ROSS (2004) este método, também conhecido como método das alturas, é o mais eficiente para aplicações computacionais. O método é apresentado na Figura 11.

Figura 11 – Método de defuzificação por média ponderada



Fonte: (ROSS, 2004).

Baseia-se no cálculo dos centroides de cada função pertinência e realiza uma média ponderada entre elas. Sendo z^* a saída do defuzificador e z a centroide de cada função pertinência temos a seguinte Equação 3.

$$z^* = \frac{\sum \mu_i(z).z_i}{\sum \mu_i(z)} \quad (3)$$

No caso da saída apresentar somente funções pertinência do tipo *singleton*, os valores de z serão os valores pontuais da função.

2.3 FERRAMENTAS UTILIZADAS

A seguir será apresentado o software de programação e a placa de desenvolvimento utilizada no desenvolvimento deste projeto, denotando suas principais características.

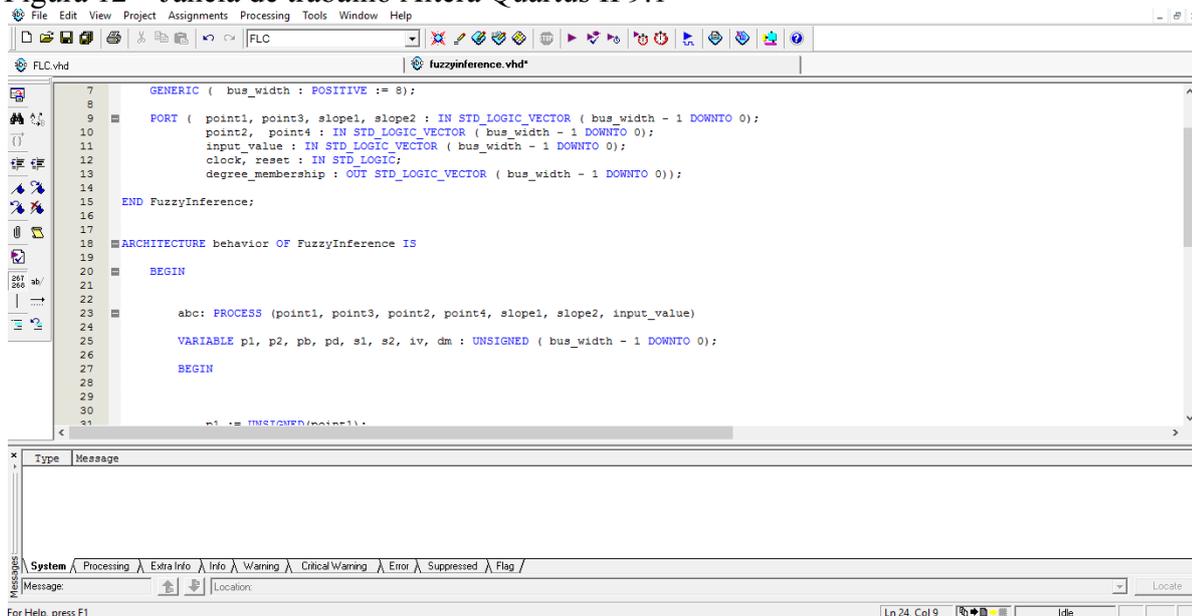
2.3.1 Altera Quartus II 9.1

Este foi o *software* utilizado para desenvolvimento dos blocos de programação e respectivas simulações, devido por ele ser um ambiente de desenvolvimento integrado, que possibilita a realização de todas as etapas de um projeto digital, desde a lógica, simulação e até gravação do projeto nos componentes CPLD ou FPGA.

O Quartus trabalha com um sistema orientado a projeto, possibilitando o projetista a criação de vários arquivos para, posteriormente, serem agregados num único projeto. Com esta característica possibilita a utilização de vários componentes básicos e blocos de programação parametrizáveis, como o deste trabalho, em diversos projetos de forma simplificada. Deste modo a sua programação é facilitada, pois, um sistema grande e complexo pode ser dividido em diversos sistemas menores e mais simples.

O ambiente de programação é apresentado na Figura 12.

Figura 12 – Janela de trabalho Altera Quartus II 9.1



Fonte: Produção do próprio Autor.

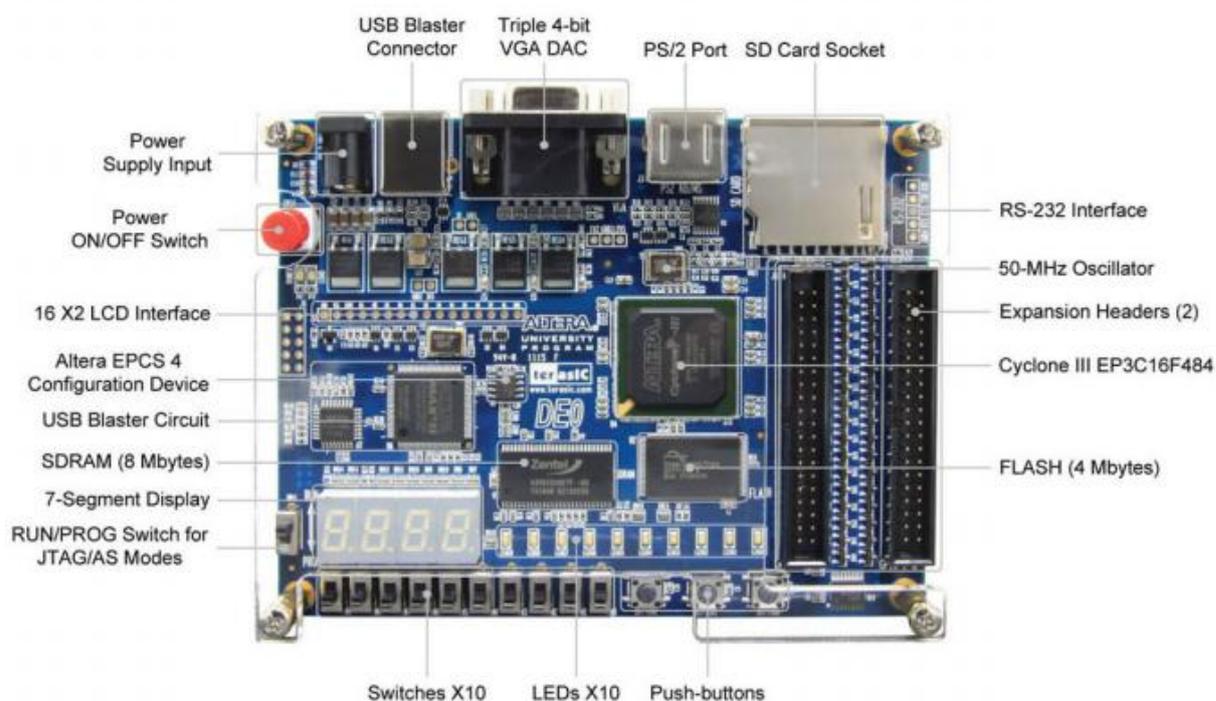
Existe muitas linguagens de descrição de hardware que possam ser utilizadas no ambiente de desenvolvimento citado, como SystemC, VERILOG, Handel-C, SDL, ISP,

ESTEREL e VHDL. A linguagem escolhida foi a VHDL, uma linguagem de descrição de hardware iniciado em 1980 pelo departamento de defesa dos Estados Unidos para documentação dos circuitos vendidos às forças aéreas americanas. Depois de padronizada pela IEEE e de domínio público, ela começou a ser utilizada mundialmente para especificação, simulação, síntese e propriedade intelectual. Suas principais vantagens são a criação de projetos independentes da tecnologia, ferramentas de CAD são compatíveis, flexibilidade na escolha de fornecedores e reutilização do código, permite explorar em alto nível de abstração e analisar de diferentes alternativas de implementação, verificação do comportamento do sistema digital através de simulação além de ser uma linguagem fortemente tipada, diminuído, assim, erros de baixo nível.

2.3.2 Placa de Desenvolvimento

A Figura 13 mostra a placa de desenvolvimento Altera DE0, utilizada no projeto dos blocos de programação.

Figura 13 – Placa de desenvolvimento Altera DE0



Fonte: (Altera Corporation)

A placa de desenvolvimento e educacional DE0 apresenta tamanho compacto, porém com todas as ferramentas essenciais para o usuário, seja iniciante ou avançado, para se obter

conhecimento nas áreas de lógica digital, organização computacional e FPGAs. O FPGA, como alvo de programação, encontrada na placa, é a Altera Cyclone III, que oferece 346 pinos de entrada e saída, portas de conexão serial e paralela, como USB para programação, porta serial EPCS4 para configuração do dispositivo, saídas VGA, porta PS/2 para se conectar com mouses e teclados, porta de comunicação RS-232, ainda apresentam diversos componentes que fazem interface com o usuário, como *push buttons* e *swithes*, LEDs, *display* de sete segmentos, interface LCD 16x2, além de outras características, como placa de expansões de entrada e saída e oscilador 50 MHz.

Por fim essa placa une o baixo consumo de energia, baixo preço de compra e sua alta eficiência, transformando-o, assim, em uma das melhores placas de desenvolvimento no mercado.

3 ESPECIFICAÇÃO E PROJETO

Neste capítulo serão mostradas as etapas de desenvolvimento dos blocos virtuais do processador *fuzzy*. Como o foco é obter a parametrização dos blocos do processador, foi escolhido trabalhar com blocos funcionais de programação no desenvolvimento do projeto. Isto é possível graças à linguagem VHDL de descrição de circuitos, que nos permite trabalhar de forma estrutural, assim, simplificando o desenvolvimento de sistemas digitais mais complexos.

Utilizando-se como base a arquitetura apresentada no capítulo anterior, podem-se identificar os principais blocos e funções que deverão ser desenvolvidos: fuzificador, máquina de inferência e defuzificador. Após a criação dos blocos virtuais parametrizáveis, também é feito um *template* de sua aplicação, facilitando para o projetista a modificação e adaptação do código para casos específicos de aplicação.

A seguir, serão detalhados todos os componentes desenvolvidos para esta aplicação, explicitando-se suas características e funcionalidade.

3.1 BLOCO DE FUZIFICAÇÃO

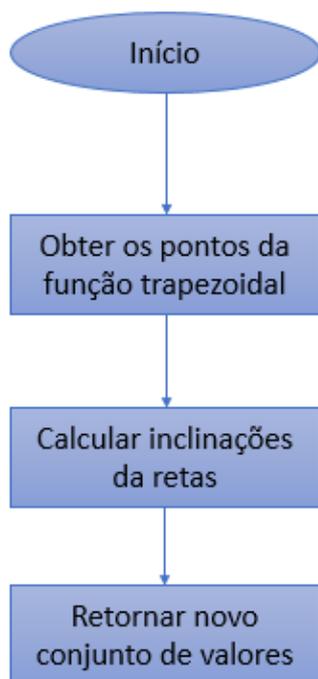
O componente Fuzificador realiza a fuzificação do processo, trazendo os dados do domínio real para o domínio *fuzzy*. Neste bloco são definidas as funções de pertinência da aplicação.

Observando as principais funções utilizadas, já apresentadas no escopo deste trabalho, percebe-se que as funções triangular, S e Z são casos particulares da trapezoidal, portanto para este projeto precisará ser definido somente a função trapezoidal, assim facilitando a implementação das funções de pertinência.

Para cada função, trapezoidais e variantes, é necessária a declaração de quatro pontos $\{a, b, c, d\}$, sendo que no caso de querer obter uma função triangular basta declarar os pontos b e c idênticos. Utilizando este método de declaração, pode-se obter alta flexibilidade, porque fica a critério do projetista as inclinações e intersecções entre as funções a serem utilizadas.

O algoritmo da etapa de fuzificação é representado pelo diagrama na Figura 14.

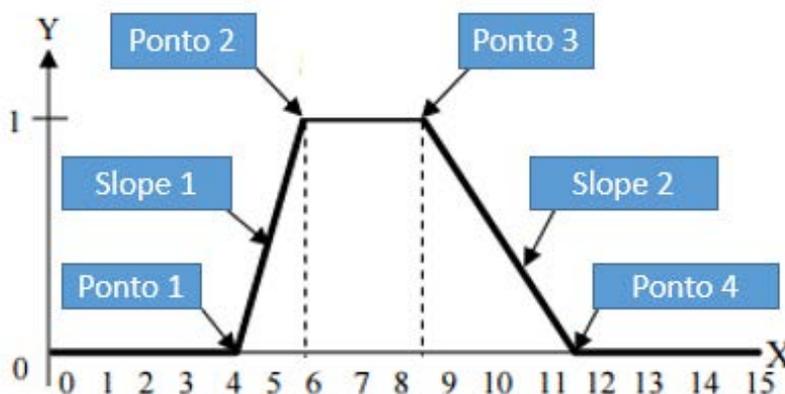
Figura 14 – Algoritmo do bloco de fuzificação



Fonte: Produção do Próprio Autor.

O bloco de fuzificação recebe os quatro pontos que definem a função de pertinência e, a partir disso, são calculados as inclinações das retas. A Figura 15 evidencia os pontos e inclinações.

Figura 15 – Função de pertinência trapezoidal e pontos característicos



Fonte: Adaptado de VUONG (2006).

Sendo assim, como entrada do bloco de fuzificação temos quatro vetores parametrizáveis que definem os pontos de um a quatro. Como saída teremos dois vetores de mesma quantidade de bits que a entrada, que serão as inclinações, *slope 1* e *slope 2*. Ambos, entradas e saídas, são definidos como sinais do tipo *unsigned*. A Figura 16 ilustra o código com o cálculo das inclinações, enquanto a Figura 17 ilustra as variáveis de entrada.

Figura 16 – Código VHDL do cálculo das inclinações das retas

```

--descrição da função desenvolvida para calculo da inclinação de uma reta
function inclinacao (pt1,pt2:unsigned (bus_width - 1 downto 0))
  return unsigned is
  variable slope: unsigned(bus_width - 1 downto 0);
begin
  slope := resize (((2**bus_width - 1)/(pt2 - pt1)), (bus_width));
  return slope;
end inclinacao;

signal slope_pos,slope_neg:unsigned (bus_width - 1 downto 0);

begin
--processo sincrono proposto para calculo da inclinação, positiva e negativa,
--de segmentos de reta para geração das funções de pertinência
funcao: process(clk,st)
begin
  if (st = '1') then
    if(clk'event and clk = '1') then
      slope_pos <= inclinacao(a,b);
      slope_neg <= inclinacao(c,d);
    end if;
  else
    slope_pos <= x"00"; slope_neg <= x"FF";
  end if;
end process;

```

Fonte: Produção do Próprio Autor.

Figura 17 – Código VHDL, bloco fuzificador

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fuzificador is
--parâmetro responsável por definir a quantidade de bits do
--fuzificador
  generic(bus_width:positive := 8);
--sinal de sincronismo e inicialização do módulo gerador de inclinação
  port(clk,st:in std_logic;
--pontos limites para calculo das inclinações, pos(p1,p2) e neg(p3,p4)
--sinal de entrada (in_signal)
    a,b,c,d,in_signal: in unsigned (bus_width - 1 downto 0);
--vetor representado o grau de pertinência da função
    grau_pertinencia:out unsigned (bus_width - 1 downto 0));
end fuzificador;

```

Fonte: Produção do Próprio Autor.

Os pontos a, b, c e d, das funções pertinência, são definidos pelo usuário dentro do *template*, onde para cada função se determina os valores de seus pontos, como está representado na Figura 18.

Figura 18 – *Template* utilizado para determinar as funções pertinência

```

--Variação de Temperatura - Função de Pertinência: Slow
slow.a<=x"00";
slow.b<=x"00";
slow.c<=x"32";
slow.d<=x"7F";

--Temperatura - Função de Pertinência: Moderate
moderate.a<=x"32";
moderate.b<=x"7F";
moderate.c<=x"7F";
moderate.d<=x"CD";

--Temperatura - Função de Pertinência: Fast
fast.a<=x"7F";
fast.b<=x"CD";
fast.c<=x"FF";
fast.d<=x"FF";

```

Fonte: Produção do Próprio Autor.

3.2 BLOCO DA MÁQUINA DE INFERÊNCIA

A etapa de inferência é responsável por obter os graus de pertinência, além de verificar a ativação das regras. Neste caso a parametrização das regras não foi atingida, pois mesmo sabendo o número máximo de regras, que é obtido multiplicando o número de entradas pelo número de saídas, o projetista pode optar por não utilizar em sua totalidade. Sabendo-se disso o bloco de inferência foi utilizado para obtenção dos graus de pertinência de cada função, e a base de regra feito separadamente.

3.2.1 Inferência

As funções pertinência podem ser divididas em cinco diferentes áreas. Essa separação facilita o cálculo para dos graus de pertinência para a entrada. As áreas 1 e 5 representam as áreas onde o grau de pertinência é zero, a área 3 representa a área onde o grau de pertinência é um, e por fim as áreas 2 e 4 representam as áreas de inclinação da função pertinência, onde seu valor é definido pela Equação 4 e Equação 5 respectivamente.

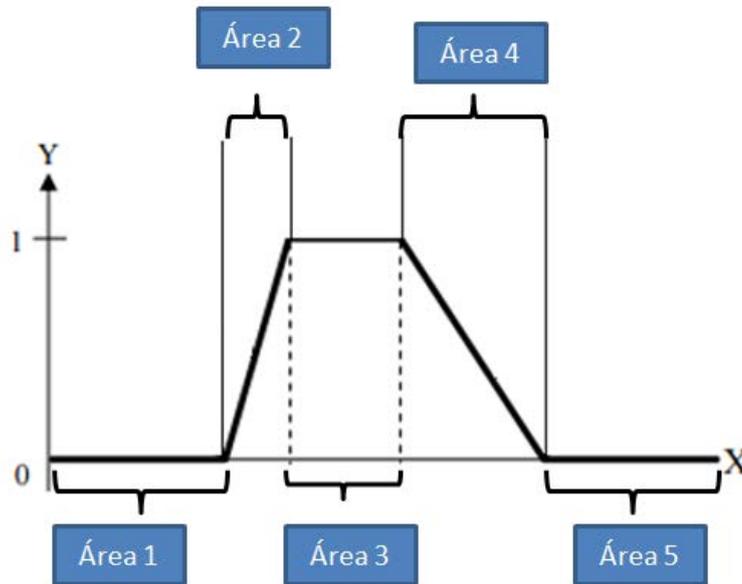
$$\mu = ((x - ponto2).slope1) \quad (4)$$

$$\mu = 1 - ((x - ponto4).slope2) \quad (5)$$

Onde μ representa o grau de pertinência da entrada e x o valor da entrada. A

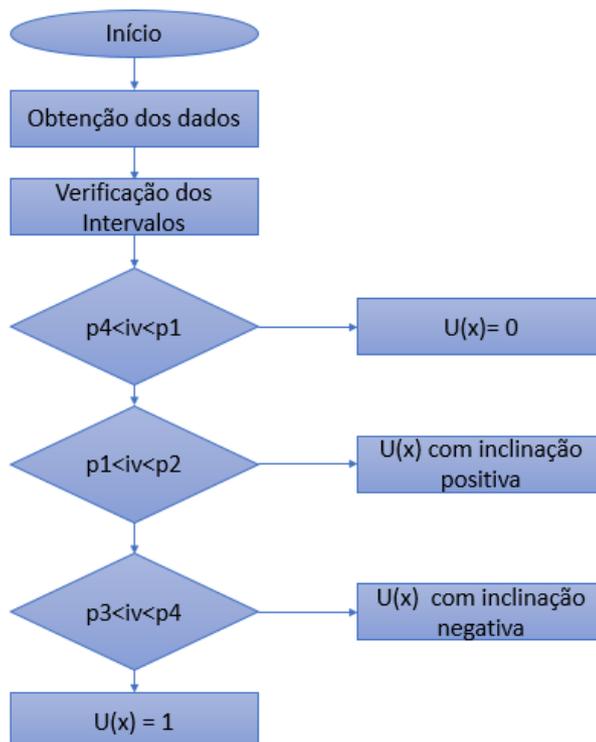
Figura 19 evidencia as possíveis zonas de intersecção das funções, em seguida a Figura 20 evidencia o algoritmo utilizado para a etapa de inferência.

Figura 19 – Função de pertinência trapezoidal e áreas das funções pertinência



Fonte: Adaptado de VUONG (2006).

Figura 20 – Algoritmo do bloco de inferência



Fonte: Produção do Próprio Autor.

No bloco de inferência têm-se sete entradas e uma saída. Nas estradas têm-se os quatro pontos que definem a função de pertinência, as duas inclinações obtidas na etapa de fuzificação e a o valor de entrada referente. Já na saída tem-se apenas o grau de pertinência da função.

A verificação da inclinação serve, neste caso, somente facilitar a intersecção do valor de entrada com a função pertinência. A Figura 21 apresenta o código VHDL referente a este bloco.

Figura 21 – Código VHDL do bloco de inferência

```

pertinencia: process(clk,st,slope_pos,slope_neg)
begin
  if (st = '1') then
    if (clk'event and clk = '0') then
      --condição que define a situação que não há intersecção entre a entrada
      --e as curvas da função de pertinência
      if ((in_signal<a) or (in_signal>d)) then
        grau_pertinencia <= (others => '0');--grau de pertinência zero
      --condição que define a situação que há intersecção entre a entrada
      --e a área de inclinação positiva da função
      elsif ((in_signal>a) and (in_signal<b)) then
        grau_pertinencia <= resize(((in_signal-a)*slope_pos), (bus_width));
      --condição que define a situação que há intersecção entre a entrada
      --e a área de inclinação negatica da função
      elsif ((in_signal>c) and (in_signal<d)) then
        grau_pertinencia <= resize(((2**bus_width - 1) - ((in_signal-c)*slope_neg)), (bus_width));
      else
        grau_pertinencia <= to_unsigned(2**bus_width - 1, bus_width);
      end if;
    end if;
  end if;
end process;

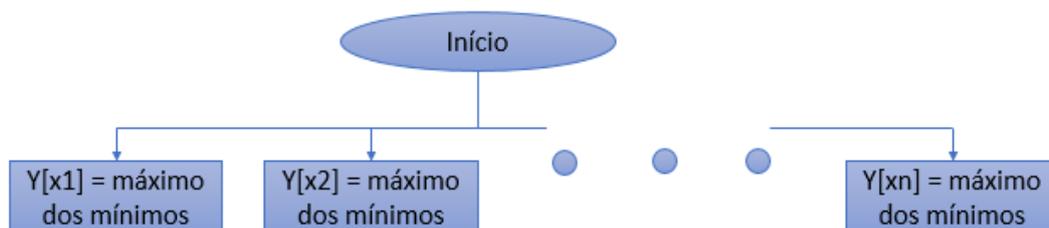
```

Fonte: Produção do Próprio Autor.

3.2.2 Base de Regras

No desenvolvimento dos blocos do FLC, ou, processador *fuzzy*, foi escolhido implementar o método de Mamdani. Sendo assim, para a base de regras serem feita, necessita das funções de máximo e mínimo, que representam os operadores lógicos “OU” e “E” respectivamente. A Figura 22 indica o algoritmo do bloco da base de regras.

Figura 22 – Algoritmo da base de regras



Fonte: Produção do Próprio Autor.

Em cada função de máximo e mínimo, sempre existirá duas entradas e uma saída. A base de regras é obtida a partir da combinação destas duas funções em relação aos graus de pertinência obtidos no bloco de inferência. A combinação de ambas as funções obtêm o modelo de regras evidenciado na Equação 6.

$$C \leq \text{maximum}(\text{minimum}(A_1, B_1), \text{minimum}(A_2, B_2)) \quad (6)$$

A Figura 23 evidencia o código das funções de máximo e mínimo.

Figura 23 – Código VHDL das funções de máximo e mínimo

```
--descrição da função de mínimo desenvolvida para o processo de inferencia
function min(number1,number2:unsigned(bus_width - 1 downto 0)) return unsigned is
begin
    if (number1>number2) then
        return number2;
    else
        return number1;
    end if;
end min;
--descrição da função de máximo desenvolvida para o processo de inferencia
function max(number1,number2:unsigned(bus_width - 1 downto 0)) return unsigned is
begin
    if (number1>number2) then
        return number1;
    else
        return number2;
    end if;
end max;
```

Fonte: Produção do Próprio Autor.

As funções máximas e mínimas são igualmente parametrizáveis em relação ao tamanho do vetor utilizado, cabendo somente ao projetista escrever a base de regras baseada em Mamdani. A Figura 24 mostra o *template* da base regras, onde o projetista pode modificar e adicionar regras ao seu sistema.

Figura 24 – *Template* utilizado para adicionar e modificar a base de regras

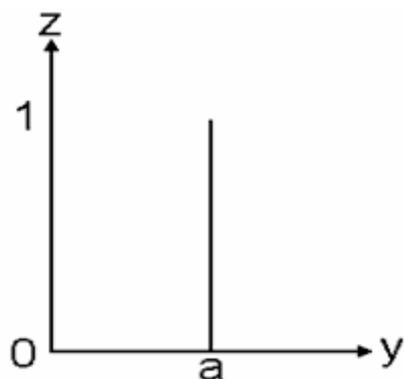
```
out_very_low <= max(min(hot.degree_membership,slow.degree_membership),min(hot.degree_membership,fast.degree_membership));
out_low <= max(min(warm.degree_membership,slow.degree_membership),min(warm.degree_membership,moderate.degree_membership));
out_medium <= max(min(cool.degree_membership,moderate.degree_membership),min(mild.degree_membership,slow.degree_membership));
out_high <= min(cold.degree_membership,slow.degree_membership);
out_very_high <= min(cold.degree_membership, fast.degree_membership);
```

Fonte: Produção do Próprio Autor.

3.3 BLOCO DE DEFUZIFICAÇÃO

A etapa de defuzificação converte o conjunto *fuzzy* obtido na saída da etapa de validação da base de regras em um valor real. A Figura 25 ilustra a função de pertinência de saída do sistema *fuzzy*, utilizado no processador.

Figura 25 – Função de pertinência do defuzificador



Fonte: Adaptado de HAGRAS (2009).

A defuzificação é feita na forma de média ponderada dos valores de ativação da regra de cada função, um método mais rápido do que o usual, utilizando funções trapezoidais e calculando seu centro de massa. Assim aumentamos a velocidade de processamento para obter uma resposta válida. A Figura 26 representa o *template* onde o usuário pode modificar o valor das saídas.

Figura 26 – *Template* dos valores das saídas *singleton*

```
Definição dos valores das saídas (singleton) da defuzificação
-----
s1<=42;
s2<=85;
s3<=127;
s4<=168;
s5<=210;
```

Fonte: Produção do Próprio Autor.

Para o cálculo da saída, primeiramente obtém o valor do numerador, dado pela multiplicação do valor da regra pelo valor da saída, como mostrado na Equação 7.

$$Num = \mu_1 \cdot Y_1 + \mu_2 \cdot Y_2 + \dots + \mu_n \cdot Y_n \quad (7)$$

Depois, obtém o valor do denominador, definido pelo somatório do valor das regras, como mostrado na Equação 8.

$$Den = \mu_1 + \mu_2 + \dots + \mu_n \quad (8)$$

Por último, efetua-se a divisão de ambos os coeficientes para obter o valor de defuzificação, como mostrado na Equação 9.

$$Defuz = \frac{Num}{Den} \quad (9)$$

A Figura 27 representa o código com o cálculo da saída do defuzificador, enquanto a Figura 28 representa as entradas do bloco de defuzificação.

Figura 27 – Código VHDL do bloco defuzificador, cálculo das saídas

```

--processo sincrono proposto para calculo do valor de defuzificação
process(clk,st)
begin
--conversão dos sinais de entrada em numeros inteiros
int_y1 <= to_integer(y1);
int_y2 <= to_integer(y2);
int_y3 <= to_integer(y3);
int_y4 <= to_integer(y4);
int_y5 <= to_integer(y5);
int_y6 <= to_integer(y6);
int_y7 <= to_integer(y7);

if(st = '1') then
if(clk'event and clk = '1') then
num <= int_y1*s_out1 + int_y2*s_out2 + int_y3*s_out3 + int_y4*s_out4 + int_y5*s_out5 + int_y6*s_out6 + int_y7*s_out7; --numerador
den <= int_y1 + int_y2 + int_y3 + int_y4 + int_y5 + int_y6 + int_y7; --denominador
z <= to_unsigned((num/den), (bus_width));
end if;
else
z <= x"00";
end if;
end process;

```

Fonte: Produção do Próprio Autor.

Figura 28 – Código VHDL do bloco defuzificador, entradas do bloco

```

entity defuzificador is
--parâmetro responsável por definir a quantidade de bits do
--defuzificador
generic(bus_width:positive := 8);
--sinal de sincronismo e inicialização do bloco de defuzificação
port( clk,st:in std_logic;
--valores das regras ativas
y1,y2,y3,y4,y5,y6,y7:in unsigned (bus_width -1 downto 0);
--valores da função de saída singleton do defuzificador
s_out1,s_out2,s_out3,s_out4,s_out5,s_out6,s_out7:in integer range 0 to (2**(bus_width - 1));
--vetor representando a saída do defuzificador
z: out unsigned (bus_width - 1 downto 0));
end defuzificador;

```

Fonte: Produção do Próprio Autor.

O bloco defuzificador é parametrizável, quanto ao número de saídas e quantidade de conjuntos *fuzzy* de saída, sendo que para modificar ou adicionar as funções, basta o usuário modificar no *template* do controlador, mostrado na Figura 29.

Figura 29 – *Template* do bloco defuzificador

```
Defuzificacao : defuzificador port map (clk=>clk, st=>st, y1=>out_very_low, y2=>out_low,
                                         y3=>out_medium, y4=>out_high, y5=>out_very_high,
                                         y6=>x"00", y7=>x"00", s_out1=>s1, s_out2=>s2, s_out3=>s3,
                                         s_out4=>s4, s_out5=>s5, s_out6=>0, s_out7=>0, z=>z);
```

Fonte: Produção do Próprio Autor.

4 SIMULAÇÕES E RESULTADOS

A implementação na parte física não foi possível, portanto os resultados obtidos serão apresentados na forma de simulações individuais dos blocos desenvolvidos. No fim, será avaliada, entretanto, a simulação do controlador *fuzzy* por inteiro. Espera-se que as simulações apresente um erro no cálculo, devido ao arredondamento, visto que utilizou-se números inteiros, uma vez que a utilização de número de ponto flutuante seja muito complexa em VHDL.

4.1 SIMULAÇÃO DO FUZIFICADOR

O fuzificador transforma as entradas do domínio real para o domínio *fuzzy*. A Tabela 2 apresenta a descrição das entradas e saídas utilizadas.

Tabela 1 – Descrição das entradas e saídas do bloco fuzificador

Legenda	Descrição
Entradas	
Pontos {a, b, c, d}	Pontos que definem a função pertinência
Clk	Sinal de relógio do sistema
St, control	Sinal de reset e controle do sistema
Saídas	
Slope_neg	Valor da inclinação negativa da função
Slope_pos	Valor da inclinação positiva da função

Fonte: Produção do Próprio Autor.

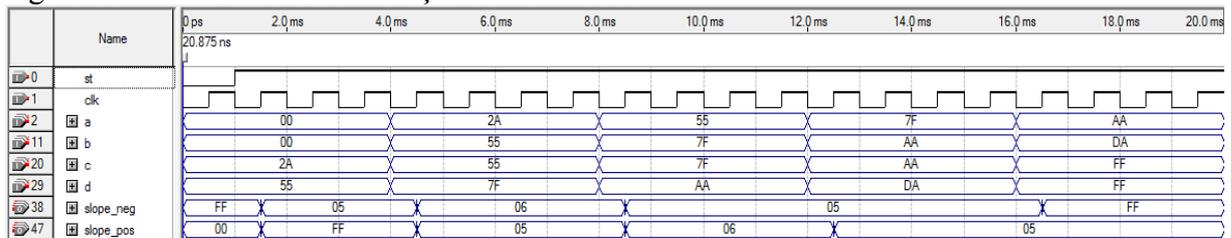
Os valores de entrada que serão utilizados na simulação deste bloco, são apresentadas na Tabela 2, onde também mostra as saídas esperadas no sistema. Na Figura 30 apresentam-se os resultados da simulação executada no Quartus.

Tabela 2 – Valores de Entrada e Resultados esperados para o teste do fuzzificador

ENTRADAS					
a	0	2 A	55	7F	AA
b	0	55	7F	AA	DA
c	2 A	55	7F	AA	FF
d	55	7F	AA	DA	FF
Slope_neg	05	06	05	05	FF
Neg_esperado	06	06	06	06	FF
Slope_pos	FF	05	06	05	05
Pos_esperado	FF	06	06	06	06

Fonte: Produção do Próprio Autor.

Figura 30 – Resultado da simulação do fuzzificador



Fonte: Produção do Próprio Autor.

4.2 SIMULAÇÃO DA MÁQUINA DE INFERÊNCIA

A máquina de inferência é responsável por verificar o grau de compatibilidade entre a entrada e saída do controlador *fuzzy*. Os parâmetros de entrada e sua explicação são apresentados na Tabela 3.

Tabela 3 – Descrição das entradas do bloco de inferência

Legenda	Descrição
Entradas	
Pontos {a, b, c, d}	Pontos que definem a função pertinência
In_signal	Valor de entrada
clk	Sinal de relógio do sistema
St, control	Sinal de reset e controle do sistema
Saídas	
grau_pertinencia	Grau de pertinência

Fonte: Produção do Próprio Autor.

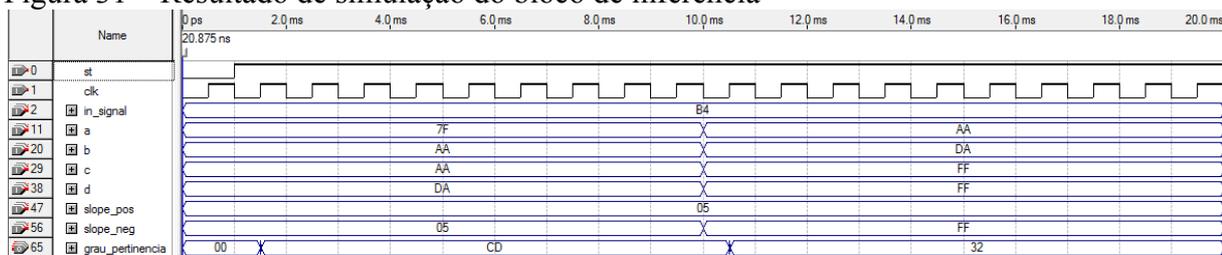
Os valores de entrada que serão utilizados na simulação deste bloco, são apresentadas na Tabela 4, onde também mostra as saídas esperadas no sistema. Na Figura 30 apresentam-se os resultados da simulação executada no Quartus.

Tabela 4 – Valores de entrada e resultados para o bloco de inferência

ENTRADAS		
In_signal	B9	
a	7F	AA
b	AA	DA
c	AA	FF
d	DA	FF
Saída	CD	32
Saída Esperada	C3	3C

Fonte: Produção do Próprio Autor.

Figura 31 – Resultado de simulação do bloco de inferência



Fonte: Produção do Próprio Autor.

4.3 SIMULAÇÃO DO BLOCO DE DEFUZIFICAÇÃO

A etapa de defuzificação converte o conjunto *fuzzy* obtido na saída da etapa de validação da base de regras em um valor real. A Tabela 5 descreve as entradas e saídas do bloco.

Tabela 5 – Descrição da entrada do bloco de defuzificação

Legenda	Descrição
Entradas	
Y1, y2,...	Grau de pertinência de todas as regras avaliadas
Clk	Sinal de relógio do sistema
St, control	Sinais de reset e controle do sistema
S_out1, s_out2,...	Função de pertinência da saída, função <i>singleton</i>
Saídas	
z	Saída do defuzificador

Fonte: Produção do Próprio Autor.

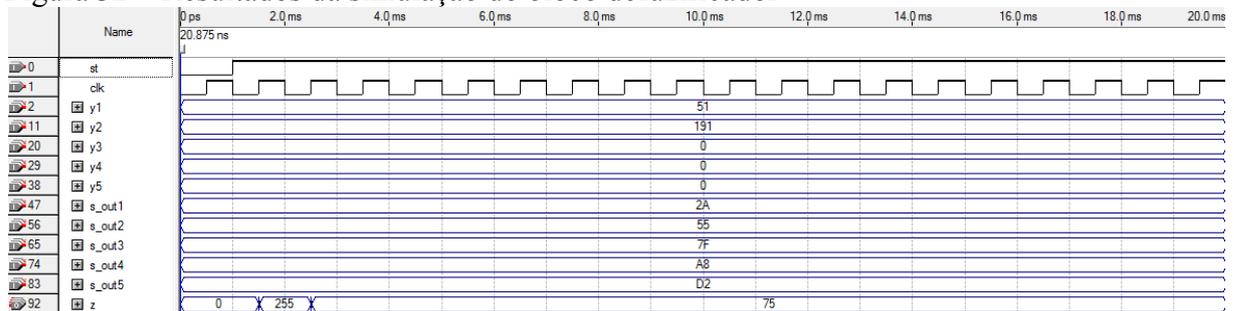
Os valores de entrada que serão utilizados na simulação deste bloco são apresentados na Tabela 6, onde também mostra as saídas esperadas no sistema. Na Figura 32 apresentam-se os resultados da simulação executada no Quartus.

Tabela 6 – Valores de entrada do bloco defuzificador

ENTRADAS	
Y1	51
Y2	191
Y3	0
Y4	0
Y5	0
S_out1	2 A
S_out2	55
S_out3	7F
S_out4	A8
S_out5	D2
z	75
Saída Esperada (z)	75

Fonte: Produção do Próprio Autor.

Figura 32 – Resultados da simulação do bloco defuzificador



Fonte: Produção do Próprio Autor.

4.4 SIMULAÇÃO DO PROCESSADOR *FUZZY* COMPLETO

A simulação do processado *fuzzy* completo considera o controle de um ar condicionado, proposto por VUONG (2006). As descrições da entrada saída do processador são mostradas na Tabela 7.

Tabela 7 – Entradas do processador *fuzzy*

Legenda	Descrição
Signal_temperatura	Temperatura do ambiente
Signal_var_temperatura	Varição de temperatura
z	Saída do processador

Fonte: Autoria do Próprio Autor.

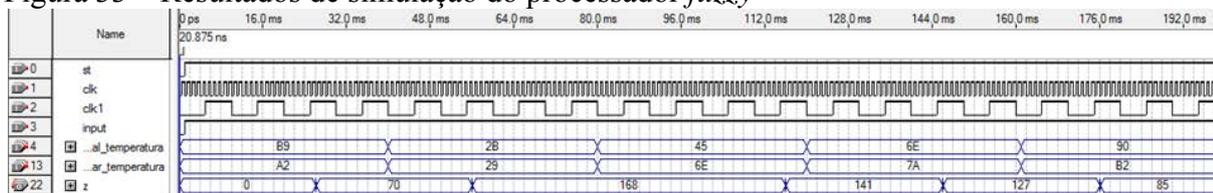
Os valores de entrada que serão utilizados na simulação deste bloco são apresentados na Tabela 8, onde também mostra as saídas esperadas no sistema. Na Figura 33 apresentam-se os resultados da simulação executada no Quartus.

Tabela 8 – Valores de entrada do processador *fuzzy*

ENTRADAS					
Signal_temperatura	B9	2B	45	6E	90
Signal_var_temperatura	A2	29	6E	7 A	B2
Z	70	168	141	127	85
Saída Esperada (z)	70	168	141	127	85

Fonte: Aatoria do Próprio Autor.

Figura 33 – Resultados de simulação do processador *fuzzy*



Fonte: Aatoria do Próprio Autor.

5 CONCLUSÃO

Neste trabalho foi realizado o desenvolvimento de blocos parametrizáveis de um processador *fuzzy* digital utilizando a linguagem de descrição de hardware VHDL, mirando a utilização em um componente FPGA.

Durante a fase de programação, a maior dificuldade foi a parametrização dos blocos virtuais. A dificuldade foi devido a algumas limitações da linguagem VHDL quanto a estrutura de dados a ser utilizada, que muitas vezes não poderia ser sintetizada.

Outro problema encontrado na parte de programação foi a modelagem da base de regras. Ela não pode ser totalmente parametrizável devido a grande versatilidade que ela apresenta, pois o projetista pode optar por não utilizar todas as regras permitidas. Isso foi resolvido com a criação das funções de máximo e mínimo, além de um *template*, que pode ser seguido pelo projetista, para criação das regras.

Durante as simulações dos blocos percebe-se que há um erro devido a utilização de números inteiros. Isto ocorreu devido a complexidade de utilizar números de ponto flutuante em VHDL, o que não foi viável neste projeto.

Como trabalho futuro deve-se aperfeiçoar os blocos individuais a ponto de obter uma melhor resposta e eficiência, realizar um estudo aprofundado das bibliotecas de número flutuantes em VHDL, a fim de sanar os erros provenientes de arredondamentos.

REFERÊNCIAS

- ASCARI, R. E. O. S.; BORSOI, B. T.; LINARES, K. S. C.; TOSCAN, L. F. Aplicação de Lógica *Fuzzy* na Estimativa de Prazo de Projetos de *Software*. **Revista Eletrônica de Sistemas de Informação**, 2012, 25p.
- BENNASSAR, A.; BARARA, M.; ABBOU, A. Fuzzy Logic Speed Control for Sensors Indirect Field Oriented of Induction Motor Using an Extended Kalman Filter. **International Review of Automatic Control**, 2013, 9p.
- CAMPOS, M. M.; SAITO, K. **Sistemas Inteligentes em Controle e Automação de Processos**. 1 ed. Rio de Janeiro: Ciência Moderna, 2004. 235p.
- CREMASCO, P. C.; FILHO, L. R. A. G.; CAETANO, A. Metodologia de Determinação de Funções de Pertinência de Controladores *Fuzzy* para a Avaliação Energética de Empresas de Avicultura de Postura. **Revista Energia na Agricultura**, 2010, 39p.
- GRILLO, S. F.; MARINHO, R. S. **Projeto de um Controlador Fuzzy para o Experimento Pêndulo Invertido Partindo do Estado de Repouso no Kit Didático Pêndulo Digital**. 2012. 59 f. Trabalho de Graduação (Graduação Engenharia Controle e Automação) – Instituto Federal de Educação, Ciência e Tecnologia Fluminense, Campos dos Goytacazes, 2012.
- MOHAGHEGH, S. Virtual Intelligence and Its Applications in Petroleum Engineering. **Journal of Petroleum Technology**, 2000. Disponível em:
< <http://www.intelligentsolutionsinc.com/Technology/AITheory/AI3-Fuzzy.shtml> > Acessado em dezembro 2016.
- ROSS, T. J. **Fuzzy Logic with Engineering Applications**. Second Edition. West Sussex: John Wiley & Sons Ltd, 2004. 628p.
- SANDRI, S.; CORREA, C. Lógica Nebulosa. V **Escola de Redes Neurais**, 1999, pp c073-c090.
- MATHWORKS Disponível em:
< <https://www.mathworks.com/help/fuzzy/index.html> > Acessado em dezembro 2016.
- SANTIAGO, R. Y. **Modelagem e controle de um pêndulo invertido utilizando a lógica fuzzy no matlab**. 2008. Trabalho de graduação (Graduação em Engenharia Elétrica) – Faculdade de Engenharia de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2008.
- SILVA, R. A. C. Inteligência Artificial aplicada a ambientes de Engenharia de Software: Uma visão geral. **INFOCOMP Journal of Computer Science**, Viçosa, v. 4, n. 4, p. 27-37, nov. 2005.
- SILVEIRA, P. R.; SANTOS, W. E. **Automação e Controle Discreto**. 6 ed. São Paulo: Érica, 2004. 227p.

VUONG, P. T.; MADNI, A. M.; VUONG, J. B. VHDL Implementation for a Fuzzy Logic Controller. **World Automation Congress**, Budapeste, p. 1-8, jul. 2006.

WEBER, L.; KLEIN, P. A. T. **Aplicação da Lógica fuzzy em software e hardware**. 2 ed. Porto Alegre: Editora da Ulbra, 2003. 110p.

ZADEH, L. A. Outline of a New Approach to the Analysis of Complex Systems and Decision Process. **IEEE Transactions on Systems, Man and Cybernetics**, v. 3, n. 1, p. 28-44, 1973.