Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Paulo Renato de Souza e Silva Sandres

Hardware reconfigurável para controladores nebulosos

Rio de Janeiro
2013

Paulo Renato de Souza e Silva Sandres

Hardware reconfigurável para controladores nebulosos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof.ª Dr.ª Nadia Nedjah
Coorientadora: Prof.ª Dr.ª Luiza de Macedo Mourelle

Rio de Janeiro
2013

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

S219 Sandres, Paulo Renato de Souza e Silva
Hardware reconfigurável para controladores nebulosos/Paulo Renato de Souza e Silva Sandres. – 2013.
104 f.

Orientadora: Nadia Nedjah.
Coorientadora: Luiza de Macedo Mourelle.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Eletrônica. 2. Lógica Nebulosa – Dissertações. 3. Hardware – Dissertações. I. Nedjah, Nadia. II. Mourelle, Luiza de Macedo. III. Universidade do Estado do Rio de Janeiro. IV. Título.

CDU 004.89

Assinatura _____ Data _____

Paulo Renato de Souza e Silva Sandres

Hardware reconfigurável para controladores nebulosos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas inteligentes e automação.

Aprovado em: 22 de fevereiro de 2013.

Banca Examinadora:

Prof.ª Dr.ª Nadia Nedjah (Orientadora)
Faculdade de Engenharia - UERJ

Prof.ª Dr.ª Luiza de Macedo Mourelle (Coorientadora)
Faculdade de Engenharia - UERJ

Prof. Dr. Fernando Antônio Campos Gomide
Universidade Estadual de Campinas - UNICAMP

Prof. Dr. Alexandre Gonçalves Evsukoff
Universidade Federal do Rio de Janeiro - UFRJ

Rio de Janeiro
2013

AGRADECIMENTOS

Agradeço a todos os professores do PEL-UERJ pelas incríveis lições aprendidas neste dois anos de curso. Em especial, agradeço às professoras Nadia Nedjah e Luiza de Macedo Mourelle, não apenas pela orientação neste trabalho e pela oportunidade de desenvolvê-lo, mas pela compreensão em lidar com um aluno com eterna falta de tempo. Obrigado por tudo, professoras!

Agradeço aos colegas de mestrado, Alexandre, Fábio, Luneque, Nicolás, Rafael, Rodrigo e Rogério. Obrigado pelas conversas intermináveis, pelas discussões político-filosóficas, ou simplesmente pela companhia no almoço. Agradeço ainda a todos que pude conhecer durante este período, muito obrigado.

Como não poderia deixar de ser, agradeço a minha inteligente e linda esposa Maria Eduarda, a quem eu amo muito e que me estimulou, apoiou e suportou durante as longas noites e finais de semana, pesquisando e desenvolvendo este trabalho. Agradeço ainda, por ter me agraciado com o melhor presente de todos, meu filho Paulo Eduardo, obrigado por existir. Apesar de você ter nascido durante o período de realização deste mestrado, você e sua mãe foram essenciais para suportar a pressão psicológica de algo como um mestrado. Se para tudo há um significado, então o destes anos está sendo tornar-me mentalmente forte.

Agradeço à minha tia, Marcia, que sempre está presente, me apoiando e incentivando. Ao meu pai, Paulo Fernando (em memória), que apesar não ter podido acompanhar a minha vida, onde quer que esteja, acredito que estará feliz por esta conquista. À minha mãe Neyma, padrasto Annibal e irmã Juliana, que mesmo estando longe durante este período, me apoiaram e incentivaram à realização do mestrado.

E por último, mas não menos importante, agradeço ao Sistema FIRJAN e ao SENAI / RJ pelo apoio e permissão da liberação de horas de trabalho para dedicação ao mestrado, sem esse tempo disponível, não seria possível a conclusão deste trabalho. Em especial ao Bruno, chefe e amigo, que sempre batalhou para mudar os paradigmas da instituição, pensando fora da caixa.

Educação é que nem moeda de ouro, é válida no mundo todo.

Provérbio Chinês

RESUMO

SANDRES, Paulo Renato de Souza e Silva. *Hardware reconfigurável para controladores nebulosos*. 2013. 104f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

Controle de processos é uma das muitas aplicações que aproveitam as vantagens do uso da teoria de conjuntos nebulosos. Nesse tipo de aplicação, o controlador é, geralmente, embutido no dispositivo controlado. Esta dissertação propõe uma arquitetura reconfigurável eficiente para controladores nebulosos embutidos. A arquitetura é parametrizável, de tal forma, que permite a configuração do controlador para que este possa ser usado na implementação de qualquer aplicação ou modelo nebuloso. Os parâmetros de configuração são: o número de variáveis de entrada (N); o número de variáveis de saída (M); o número de termos linguísticos (Q); e o número total de regras (P). A arquitetura proposta proporciona também a configuração das características que definem as regras e as funções de pertinência de cada variável de entrada e saída, permitindo a escalabilidade do projeto. A composição das premissas e consequentes das regras são configuráveis, de acordo com o controlador nebuloso objetivado. A arquitetura suporta funções de pertinência triangulares, mas pode ser estendida para aceitar outras formas, do tipo trapezoidal, sem grandes modificações. As características das funções de pertinência de cada termo linguístico, podem ser ajustadas de acordo com a definição do controlador nebuloso, permitindo o uso de triângulos. Virtualmente, não há limites máximos do número de regras ou de termos linguísticos empregados no modelo, bem como no número de variáveis de entrada e de saída. A macro-arquitetura do controlador proposto é composta por N blocos de fuzzificação, 1 bloco de inferência, M blocos de defuzzificação e N blocos referentes às características das funções de pertinência. Este último opera apenas durante a configuração do controlador. A função dos blocos de fuzzificação das variáveis de entrada é executada em paralelo, assim como, os cálculos realizados pelos blocos de defuzzificação das variáveis de saída. A paralelização das unidades de fuzzificação e defuzzificação permite acelerar o processo de obtenção da resposta final do controlador. Foram realizadas várias simulações para verificar o correto funcionamento do controlador, especificado em VHDL. Em um segundo momento, para avaliar o desempenho da arquitetura, o controlador foi sintetizado em FPGA e testado em seis aplicações para verificar sua reconfigurabilidade e escalabilidade. Os resultados obtidos foram comparados com os do MATLAB em cada aplicação implementada, para comprovar precisão do controlador.

Palavras-chave: Controlador Nebuloso, Controle Nebuloso, FPGA.

ABSTRACT

Process control is one of the many applications that benefits from fuzzy control. In this kind of application, the controller is usually embedded in the controlled device. This dissertation proposes a reconfigurable architecture for efficient embedded fuzzy controllers. The architecture is customizable, as it allows the controller configuration to be used to implement any fuzzy model. The configuration parameters are: the number of input variables (N); the number of output variables (M); the number of linguistic terms (Q); and the total number of rules (P). The proposed architecture also enables the configuration of the characteristics that define the rules and membership functions of each input and output variable, allowing for an optimal scalability of the project. The composition of the antecedent and consequent of the rules are configurable, according to the fuzzy model that is being implemented. A priori, the architecture supports triangular membership functions, but it can be extended to accommodate other forms, such as trapezium, without major modifications. The characteristics of the lines, forming the membership functions of the linguistic terms, can be adjusted according to the definition of the fuzzy model, allowing the use of non-isosceles and isosceles triangles. Virtually, there are no limits on the number of rules or linguistic terms used in the model, as well as the number of input and output variables. The macro-architecture of the proposed controller is composed of N fuzzification blocks, 1 inference block, M defuzzification blocks and N blocks to handle the characteristics of the membership functions. This block operates only during the controller setup. The work done by the fuzzification blocks of the input variables is executed in parallel, as well as the computation performed by the defuzzification blocks of the output variables. The duplication of the fuzzification and defuzzification blocks accelerates the process of yielding the final response of the controller. Several simulations were performed to verify the correct operation of the controller, which is specified in VHDL. In a second stage, to evaluate the controller performance, the architecture was synthesized into a FPGA and tested with six applications to verify the reconfigurability and scalability of the design. The results obtained were compared with the ones obtained from MATLAB for each of the implemented applications, to demonstrate the accuracy of the controller.

Keywords: Fuzzy Control. Fuzzy controller. FPGA.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

M	Número de variáveis de saída
N	Número de variáveis de entrada
P	Número de regras
Q	Número de termos linguísticos
CNP	Controlador Nebuloso Proposto
FIS	Fuzzy Inference System
FPGA	Field Programmable Gate Arrays
FPU	Floating Point Unit
IP	Intellectual Property
MATLAB	MATrix LABoratory
MEF	Máquina de Estados Finitos
PID	Proporcional Integral Derivativo
VHDL	Very High Speed Integrated Circuits Hardware Description Language

SUMÁRIO

INTRODUÇÃO

A TEORIA de conjuntos nebulosos é um assunto de elevado interesse no meio científico, mas ainda não é comumente utilizado na indústria Brasileira, como deveria. Eventualmente, encontramos uma publicação, sobre uma aplicação de um controlador nebuloso, que esteja sendo usado em uma aplicação real na indústria Brasileira (JUNHOR et al., 2005; RACHEL, 2006; MACHADO et al., 2011). Entretanto, na indústria, em geral, existem mais casos de aplicações reais, como: controle de conversor DC-DC multifásico (ALVAREZ et al., 2006), controle para sistemas de alimentação fotovoltaicos (CHEKIRED et al., 2011), controle para sistemas eletro-hidráulicos (SINTHIPSOMBOON et al., 2011), controle para processos industriais (RUBAAI; JERRY; SMITH, 2011), controle para processos de misturas químicas (SRINIVASAN; LAKSHMI, 2002) e controle de velocidade de sistemas com servo motores (WANG, 2011).

A intensão do desenvolvimento deste controlador em *hardware* é a possibilidade de criação de um produto, que possa ser utilizado de forma mais ampla e talvez contribuir na disseminação do conceito de controladores nebulosos na indústria Brasileira.

A modelagem de sistemas computacionais é repleta de situações ambíguas, onde o desenvolvedor não consegue decidir, com precisão, o que deveria ser a saída do sistema. L. Zadeh, em (ZADEH, 1965), introduziu pela primeira vez o conceito de conjuntos nebulosos. Quando ele sugeriu os conjuntos nebulosos e a teoria relacionada, a principal ideia de Zadeh foi de reduzir a complexidade dos sistemas e prover ao desenvolvedor um novo paradigma computacional que permita alcançar resultados aproximados, quando a precisão não é necessária. Podemos aplicar a teoria de conjuntos nebulosos e raciocínio aproximado sempre que há imprecisão. A teoria de conjuntos nebulosos e o raciocínio aproximado (ZADEH, 1968, 1984) podem ser usados em modelagem e controle de sistemas, da mesma forma, que agrupamento de informações e predições (RADECKI, 1982). Adicionalmente, também podem ser usados em finanças (DIAO; HELLERSTEIN; PAREKH, 2002), processamento de imagens (WACHS; STERN; EDAN, 2003; FRANKE; KÖPPEN; NICKOLAY,

2000), controle de temperatura e pressão (ZHANG; KNOLL, 1999; MAGDALENA; VELASCO, 1996), controle de robôs para as seguintes finalidades: geração de mapas (GHIDARY et al., 2001), navegação aérea (SHIM et al., 1998), simulação de habilidades humanas de direção (LI; CHANG; CHEN, 2003), navegação terrestre (LIN; HUANG; CHUANG, 2005), entre outras (NEDJAH; MOURELLE, 2005; VARGENS; TANSCHEIT; VELLASCO, 2003).

O objetivo deste trabalho é o desenvolvimento de um controlador nebuloso em *hardware* reconfigurável do tipo FPGA (*Field Programmable Gate Arrays*) , para o controle de processos industriais. O projeto precisa ser reconfigurável em relação à quantidade de variáveis de entrada (N), de variáveis de saída (M), termos linguísticos (Q) e regras (P). Além destes parâmetros, deve ser possível, também, alterar na configuração do controlador, cada função de pertinência, ou seja, a forma de cada termo linguístico de cada variável de entrada e saída. Ainda, deve ser possível alterar as regras, de tal forma que a premissa possa ser usada com todas ou somente algumas das variáveis de entrada e o consequente também possa ser usado com todas ou somente algumas das variáveis de saída. Todos esses parâmetros de configuração do projeto permitem seu uso com, praticamente, qualquer aplicação. Estes parâmetros podem receber, virtualmente, qualquer valor, sendo limitado apenas pelo tamanho de área de *hardware* disponível na FPGA.

Como parte desta pesquisa, o controlador nebuloso proposto (CNP) descrito em VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) tem sua arquitetura detalhada no nível macro e micro, sendo validada em simulação utilizando um pêndulo invertido como processo.

A macro-arquitetura do CNP é composta de N blocos de função de pertinência, N blocos de fuzzificação, 1 bloco inferência e M blocos de defuzzificação. O bloco referente à configuração da função de pertinência, responsável pela composição das funções de pertinência, opera apenas na etapa de configuração do CNP, não participando da etapa de controle. Os blocos de fuzzificação são responsáveis pela conversão dos valores escalares de entrada do controlador para as variáveis nebulosas. O bloco de inferência utiliza as regras nebulosas, verificando suas premissas por meio dos valores das variáveis nebulosas de entrada, e inferindo dos consequentes das regras disparadas para gerar os valores das variáveis nebulosas de saída do controlador. Já os blocos de defuzzificação são responsáveis pela conversão das variáveis nebulosas de saída para valores escalares, que serão os resultados finais do CNP.

A arquitetura do CNP explora o paralelismo durante o processo de fuzzificação das entradas e, também, durante o processo de defuzzificação das saídas. Os blocos de fuzzificação são executados em paralelo, da mesma forma que os blocos de defuzzificação. Isto não onera o CNP, em termos de tempo de execução, quando a quantidade de entradas e saídas é grande. Entretanto, qualquer alteração deste tipo repercute no consumo de área de *hardware*.

A arquitetura do CNP é sintetizada em uma FPGA da Xilinx da família Virtex 5, modelo XC5VLX110T, como um coprocessador nebuloso para o processador Microblaze. Após o CNP receber os valores de entrada e o comando habilitando sua operação, os resultados são entregues ao final de toda a execução. Esta síntese é realizada com várias aplicações diferentes, onde são apresentados os resultados, juntamente com suas respectivas avaliações. A *toolbox* FIS (*Fuzzy Inference System*) da ferramenta MATLAB é utilizada para avaliar o CNP.

Existem muitos trabalhos relacionados que implementam um controlador *fuzzy* em FPGA. No entanto, a maioria destes apresenta o projeto de controladores que servem apenas para uma aplicação específica, ou seja, não são genéricos, nem reconfiguráveis. Mesmo considerando aqueles que são ditos genéricos, estes possuem o número de entradas e saída fixos e/ou um número de termos linguísticos máximo. Em geral, os controladores desenvolvidos não usam variáveis de 32-bits em ponto flutuante. Alguns trabalhos nestas condições são: (POORANI et al., 2005) ou (SULAIMAN et al., 2009), além dos descritos no Capítulo 1. A representação das variáveis em ponto flutuante é crucial para uma boa precisão e, portanto, uma maior sensibilidade do controlador. Todo cálculo feito pelo controlador proposto é realizado por uma unidade de ponto flutuante de propriedade intelectual (IP) especificada em VHDL (AL-ERYANI, 2006).

O Capítulo 1 apresenta a teoria de conjuntos nebulosos, mostrando as principais características, comparando-os com a teoria de conjuntos clássica. Isso permite dar o embasamento teórico para a sua utilização em controladores nebulosos. Além da utilização destes conceitos para a construção de controladores nebulosos, mostrando suas características e utilização, através do exemplo do pêndulo invertido.

O Capítulo 2 apresenta diversos trabalhos publicados sobre controladores nebulosos implementados em *hardware*, utilizando FPGA.

O Capítulo 3 detalha o projeto do CNP, descrevendo-o, no nível de macro e micro-

arquiteturas, assim como o funcionamento de todos os componentes que o compõe. Além disso, resultados de simulação dos componentes são apresentados para validar o projeto.

No Capítulo 4 é feito um levantamento dos resultados experimentais com o CNP configurado para 6 diferentes aplicações que utilizam controladores nebulosos, implementados em FPGA. Para cada aplicação, é feita uma breve explicação do controle realizado, além de apresentar as características do modelo do controle nebuloso para a configuração do CNP. Em seguida, são avaliados os resultados obtidos, tais como a precisão dos valores de saída do CNP, a superfície de controle, o erro introduzido pelo CNP quando comparado com o MATLAB, o consumo de área de *hardware* e os tempos de execução necessários. Este Capítulo reforça a capacidade de reconfiguração do CNP, como uma forte característica do trabalho proposto.

O Capítulo 5 completa esta dissertação, apresentando as principais conclusões decorrentes do desenvolvimento do presente trabalho. São apresentadas também possíveis melhorias para o CNP e sugestões de utilização do mesmo.

Capítulo 1

TEORIA DOS CONJUNTOS NEBULOSOS

NESTE capítulo é feita uma introdução à teoria de conjuntos nebulosos, juntamente com a lógica nebulosa, com uma comparação com a teoria dos conjuntos clássicos e lógica Booleana, respectivamente. Esta introdução proporciona um embasamento conceitual básico para entender o desenvolvimento e a implementação de controladores nebulosos.

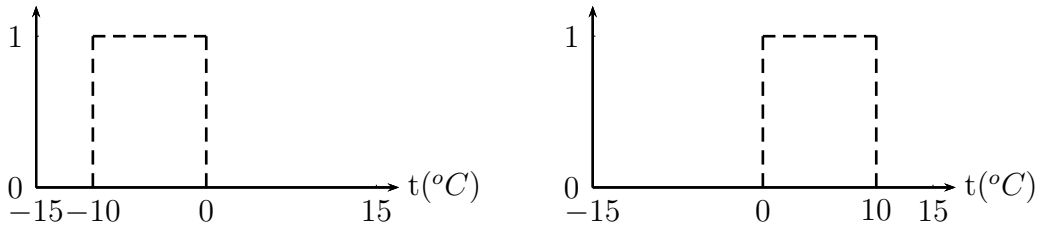
1.1 Conjuntos Nebulosos

Convencionalmente, um conjunto S é dito clássico, se e somente se um dado elemento $x \in \mathcal{U}$, onde $\mathcal{U}$ é chamado de *universo de discurso*, tem-se que ou $x \in S$ ou $x \notin S$. Para conjuntos nebulosos, isto não é totalmente verdade. Todo elemento $x \in \mathcal{U}$ pertence à um conjunto nebuloso, com um certo grau de pertinência. Um conjunto nebuloso S é definido pela função $\mu_S : \mathcal{U} \hookrightarrow [0, 1]$, chamada de *função de pertinência*, o qual dado um elemento $x \in \mathcal{U}$, define seu grau de pertinência para S. O par $(x_0, \mu_S(x_0))$ é um conjunto nebuloso. Portanto, cada conjunto nebuloso pode ser considerado como um conjunto clássico, definido como a união de todos os pares $(x, \mu_S(x))$, para todos $x \in \mathcal{U}$. Um conjunto S é equivalente à sua função característica μ_S, conforme a Equação 1.

$$\mu_S \hookrightarrow \{0, 1\}$$

$$\mu_S(x) = \begin{cases} 1, & \text{se } x \in S \\ 0, & \text{se } x \notin S \end{cases} \tag{1}$$

Há muitos exemplos de conjuntos e conjuntos nebulosos na literatura. Um exemplo para ilustrar conjuntos e conjuntos nebulosos é no caso para se definir o conceito de água



(a) μ_{gelada} (b) μ_{fria}

Figura 1: Representação dos conceitos de água *gelada* e *fria* com conjuntos

gelada e *fria*. Usando conjuntos, é possível descrever estes conceitos conforme a Equação 2.

$$\mu_{gelada}(t) = \begin{cases} 1, & \text{se } -10°\text{C} \leq t \leq 0°\text{C} \\ 0, & \text{senão} \end{cases}$$
$$\mu_{fria}(t) = \begin{cases} 1, & \text{se } 0°\text{C} < t \leq 10°\text{C} \\ 0, & \text{senão} \end{cases} \tag{2}$$

Como é possível ver na representação das funções características μ_{gelada} e μ_{fria} na Equação 2. Na Figura 1, o conceito de água gelada e fria são, neste caso, mutuamente exclusivas.

Certamente, alguém poderia esperar que água à temperatura $-10°$C é mais gelada que à 1°C. Água à temperatura $-10°$C está mais para congelada do que gelada. Analogamente, água à temperatura $-10°$C é muito mais fria do que em 0°C. Água à temperatura 10°C está mais para quente do que fria. Por outro lado, não é preciso definir a transição de gelada para fria pela aplicação de um único grau Celsius. No mundo real, seria de se esperar uma transição gradual e suave entre os estados gelada e fria. Gradualidade não é levada em conta na definição de conjuntos, conforme a Equação 2.

É comum a utilização de gráficos para descrever a função de pertinência de conjuntos nebulosos. Por razões de simplicidade, conjuntos nebulosos com funções de pertinência *triangular*, *trapezoidal* e *Gaussiana* são os mais comuns na prática. A representação com conjuntos nebulosos triangular dos conceitos de água gelada e água fria, por exemplo, pode ser definida, conforme a Equação 3, que está representada graficamente na Figura 2.

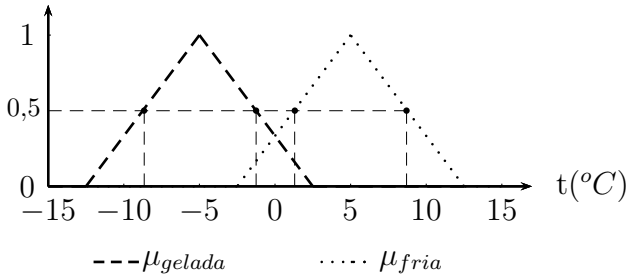


Figura 2: Representação do conceito de água *gelada* e *fria* com conjuntos nebulosos

$$\mu_{gelada}(t) = \begin{cases} \frac{2}{15}t + \frac{5}{3}, & \text{se } -12,5°\text{C} < t \leq -5°\text{C} \\ -\frac{2}{15}t + \frac{1}{3}, & \text{se } -5°\text{C} < t \leq 2,5°\text{C} \\ 0, & \text{senão} \end{cases}$$

$$\mu_{fria}(t) = \begin{cases} \frac{2}{15}t + \frac{1}{3}, & \text{se } -2,5°\text{C} < t \leq 5°\text{C} \\ -\frac{2}{15}t + \frac{5}{3}, & \text{se } 5°\text{C} < t \leq 12,5°\text{C} \\ 0, & \text{senão} \end{cases} \tag{3}$$

Considerando o gráfico da Figura 2, pode-se ver que a água com temperatura 0°C tem um grau de pertinência de $\frac{1}{3}$ com o conjunto nebuloso gelada, e um grau de pertinência de $\frac{1}{3}$ com o conjunto nebuloso fria. No exemplo da Figura 2, existe uma sobreposição de, aproximadamente, 33%. Pontos em que a função de pertinência de um conjunto nebuloso tem valor $0,5$ são chamados de pontos de *crossover* do conjunto (NEDJAH; MOURELLE, 2005), conforme os pontos marcados na Figura 2 dos conjuntos nebulosos *gelada* e *fria*, respectivamente.

1.1.1 Variáveis linguísticas

Dois aspectos, essenciais para desenvolver os sistemas nebulosos são (ZADEH, 1965, 1984): os conceitos a serem considerados, como *temperatura*, *idade* e *altura* e as características dos conceitos identificados, como *gelada* e *fria* para temperatura, *jovem* e *velho* para idade e *baixo* e *alto* para altura. Os conceitos relevantes de um sistema nebuloso são representados por *variáveis linguísticas*, enquanto os valores das características correspondentes são chamadas de *termos linguísticos* (ESRAGH; MAMDANI, 1981).

Por exemplo, considere a variável linguística *temperatura* da água em um tanque, sendo *gelada*, *fria*, *fresca* e *quente* as características da água que nos interessam. Exemplos

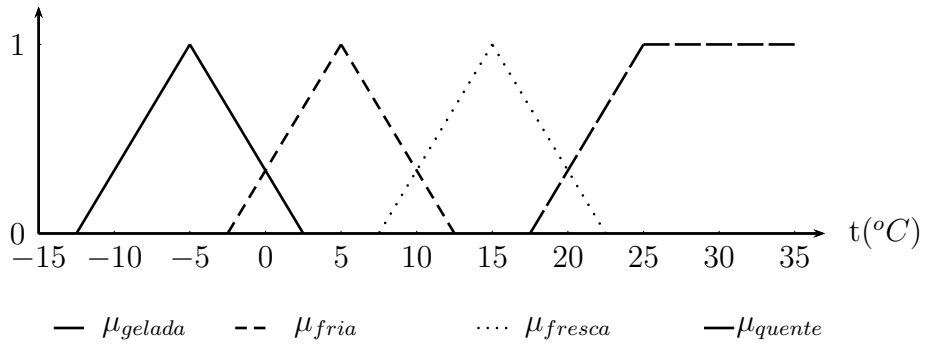


Figura 3: Variável linguística *temperatura* e seus termos linguísticos *gelada*, *fria*, *fresca* e *quente*

de conjuntos nebulosos que representam estas características são mostradas na Figura 3. O universo da variável linguística é $\mathcal{U} = [-15, 35]$.

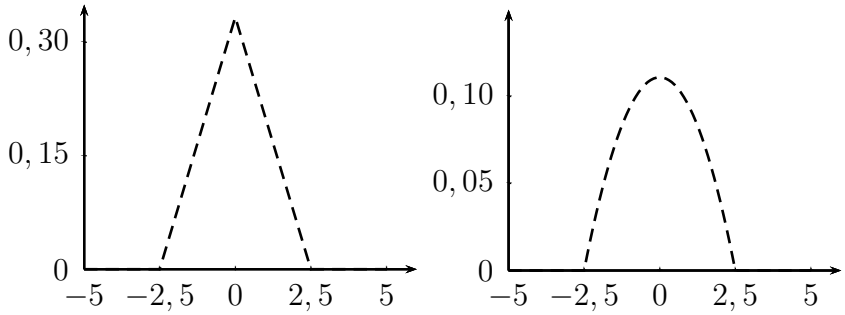
1.1.2 Operadores

A manipulação de variáveis linguísticas requer operações com conjuntos nebulosos. Estes operadores nebulosos são, normalmente, uma extensão dos conjuntos clássicos ou convencionais. A seguir, é definida a versão nebulosa das operações sobre os conjuntos nebulosos.

1.1.2.1 Interseção de conjuntos nebulosos

Existem muitas maneiras de definir a interseção de dois conjuntos nebulosos. Operadores de interseção que satisfazem os axiomas razoáveis para a real definição funcional de interseção são chamados de *normas triangulares* (ou *T-norma*). Sejam A e B dois conjuntos nebulosos definidos por suas respectivas funções de pertinência μ_A e μ_B. O operador de interseção $\sqcap$ aplicado a A e B, $\mu_{A\cap B}(x) = \mu_A(x) \sqcap \mu_B(x)$ é uma *T-norma*, se e somente se, para todo elemento em [0,1], o mesmo satisfaça os seguintes axiomas:

- $\sqcap$: [0,1]×[0,1] → [0,1];
- Comutatividade: $\forall x \in \mathcal{U},\ \mu_A(x) \sqcap \mu_B(x) = \mu_B(x) \sqcap \mu_A(x)$;
- Associatividade: $\forall x \in \mathcal{U},\ (\mu_A(x) \sqcap \mu_B(x)) \sqcap \mu_C(x) = \mu_A(x) \sqcap (\mu_B(x) \sqcap \mu_C(x))$;
- Monotonicidade: $\forall \alpha, \beta \in [0,1], \forall x \in \mathcal{U},\ \mu_A(x) \geq \alpha,\ \mu_B(x) \geq \beta \iff$ $\mu_A(x) \sqcup \mu_B(x) \geq \alpha \sqcup \mu_B(x),\ \mu_A(x) \sqcup \mu_B(x) \geq \mu_A(x) \sqcup \beta$;
- Condições de contorno: $\forall x \in \mathcal{U},\ \mu_A(x) \sqcap 1 = \mu_A(x),\ \mu_A(x) \sqcap 0 = 0$.



(a) Padrão (b) Algébrica

Figura 4: Interseção das funções de pertinência μ_{gelada} e μ_{fria} definidas na Figura 2

Intuitivamente, espera-se que os elementos do conjunto nebuloso obtidos pela interseção de outros dois conjuntos, sejam aqueles elementos que apresentam características comuns, com certo grau. Os operadores de interseção usuais são: *interseção de Zadeh* ou *padrão* (ZADEH, 1965), *produto algébrico* (ZADEH, 1965), *interseção de Lukasiewicz* ou *restrito* (KIKUCHI; S., 1998) e *robusto* (CORDÓN; HERRERA; PEREGRÍN, 2000). Estes operadores são definidos nas Equações 4, 5, 6 e 7. Como exemplo, foi traçada (vide Figura 4) a interseção das funções de pertinência μ_{gelada} e μ_{fria} da Figura 2, usando os operadores padrão e algébrica. Note que os operadores de interseção restrito e robusto produzem uma função de pertinência $\mu_{gelada \cap fria}(x) = 0$.

$$(\mu_A \sqcap \mu_B)(x) = \min(\mu_A(x), \mu_B(x)) \tag{4}$$

$$(\mu_A \sqcap \mu_B)(x) = \mu_A(x) \times \mu_B(x) \tag{5}$$

$$(\mu_A \sqcap \mu_B)(x) = \max(\mu_A(x) + \mu_B(x) - 1, 0) \tag{6}$$

$$(\mu_A \sqcap \mu_B)(x) = \begin{cases} \mu_A(x), & \text{se } \mu_B(x) = 1 \\ \mu_B(x), & \text{se } \mu_A(x) = 1 \\ 0, & \text{senão} \end{cases} \tag{7}$$

1.1.2.2 União de conjuntos nebulosos

Assim como para a interseção, existem muitas maneiras de definir a união de dois conjuntos nebulosos. Os operadores de união que satisfazem os axiomas da união são chamados *conormas triangulares* (ou *T-conorma*). Sejam A e B dois conjuntos nebulosos definidos

por suas respectivas funções de pertinência μ_A e μ_B. O operador de união $\sqcup$, aplicado a A e B, $\mu_{A\cup B}(x) = \mu_A(x) \sqcup \mu_B(x)$ é uma *T-conorma*, se e somente se, para todos os elementos em [0,1], o mesmo satisfaça os seguintes axiomas. Observe que os primeiros quatro axiomas são os mesmos para a *T-norm*. As condições de fronteira, entretanto, são distintas.

- $\sqcup$: [0,1]×[0,1] → [0,1];
- Comutatividade: $\forall x \in \mathcal{U}$, $\mu_A(x) \sqcup \mu_B(x) = \mu_B(x) \sqcup \mu_A(x)$;
- Associatividade: $\forall x \in \mathcal{U}$, $(\mu_A(x) \sqcup \mu_B(x)) \sqcup \mu_C(x) = \mu_A(x) \sqcup (\mu_B(x) \sqcup \mu_C(x))$;
- Monotonicidade: $\forall \alpha, \beta \in [0,1], \forall x \in \mathcal{U}$, $\mu_A(x) \geq \alpha$, $\mu_B(x) \geq \beta \iff$ $\mu_A(x) \sqcup \mu_B(x) \geq \alpha' \sqcup \mu_B(x)$, $\mu_A(x) \sqcup \mu_B(x) \geq \mu_A(x) \sqcup \beta'$;
- Condições de limite: $\forall x \in \mathcal{U}$, $\mu_A(x) \sqcup 1 = 1$, $\mu_A(x) \sqcup 0 = \mu_A(x)$.

Intuitivamente, espera-se que os elementos do conjunto nebuloso obtido pela união de dois outros conjuntos nebulosos, os quais representam duas características distintas, sejam aqueles elementos que apresentem qualquer uma das duas característica, com certo grau. Assim como, aqueles que apresentem ambas, com certo grau. O operador de união mais usuais para conjuntos nebulosos são: *união de Zadeh* ou *padrão*, *soma algébrica*, *união de Lukasiewicz* ou *restrito* e *robusto*. Estes operadores são definidos nas Equações 8, 9, 10 e 11. Como exemplo, foi traçada (vide Figura 5) a união das funções de pertinência μ_{gelada} e μ_{fria} da Figura 2 usando os operadores de união introduzidos.

$$\mu_A(x) \sqcap \mu_B(x) = \max(\mu_A(x), \mu_B(x)) \tag{8}$$

$$\mu_A(x) \sqcap \mu_B(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \times \mu_B(x) \tag{9}$$

$$\mu_A(x) \sqcap \mu_B(x) = \min(1, \mu_A(x) + \mu_B(x) + 1) \tag{10}$$

$$\mu_A(x) \sqcap \mu_B(x) = \begin{cases} \mu_A(x), & \text{se } \mu_B(x) = 0 \\ \mu_B(x), & \text{se } \mu_A(x) = 0 \\ 1, & \text{senão} \end{cases} \tag{11}$$

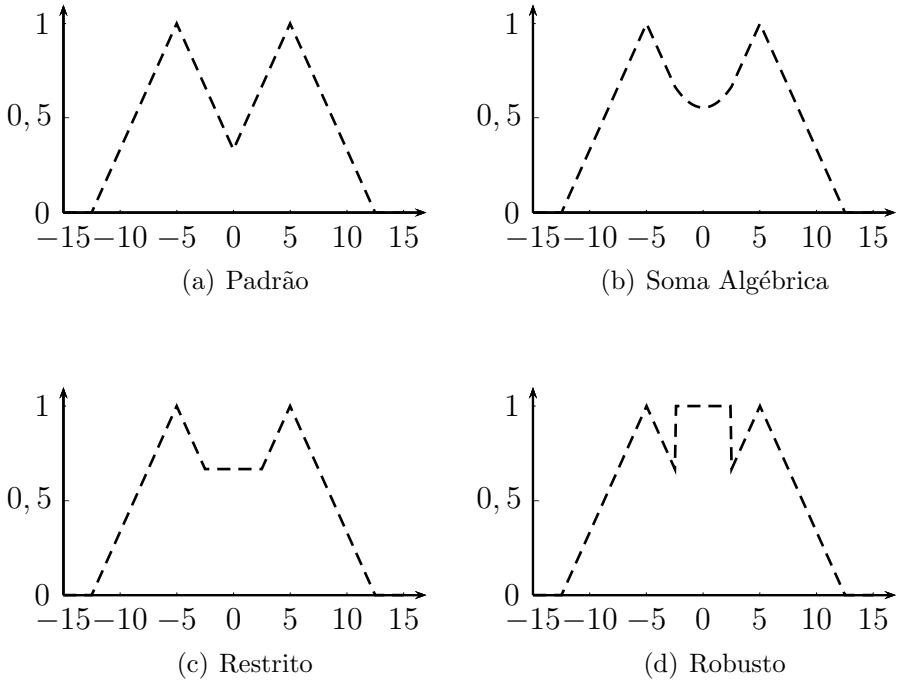


Figura 5: União das funções de pertinência μ_{gelada} e μ_{fria} definidas na Figura 2

1.1.2.3 Complemento de conjuntos nebulosos

Como os operadores de interseção e união, existem muitas maneiras de definir o complemento de um conjunto nebuloso. Operadores de complemento precisam satisfazer alguns axiomas para se justificarem para tal. Seja A um conjunto nebuloso definido por sua função de pertinência μ_A em $\mathcal{U}$. O operador unário $\eta : [0,1] \to [0,1]$, é um operador de complemento, se e somente se, para todos os elementos em $[0,1]$, o mesmo satisfaça os seguintes axiomas:

- Limites: $\eta(0) = 1$ and $\eta(1) = 0$;
- Monotonicidade: $\forall x, y \in \mathcal{U}$, $\mu_A(x) < \mu_A(y) \iff \eta(\mu_A(x)) > \eta(\mu_A(y))$
- Involução: $\forall x \in \mathcal{U}$, $\eta(\eta(\mu_A(x)) = \mu_A(x)$

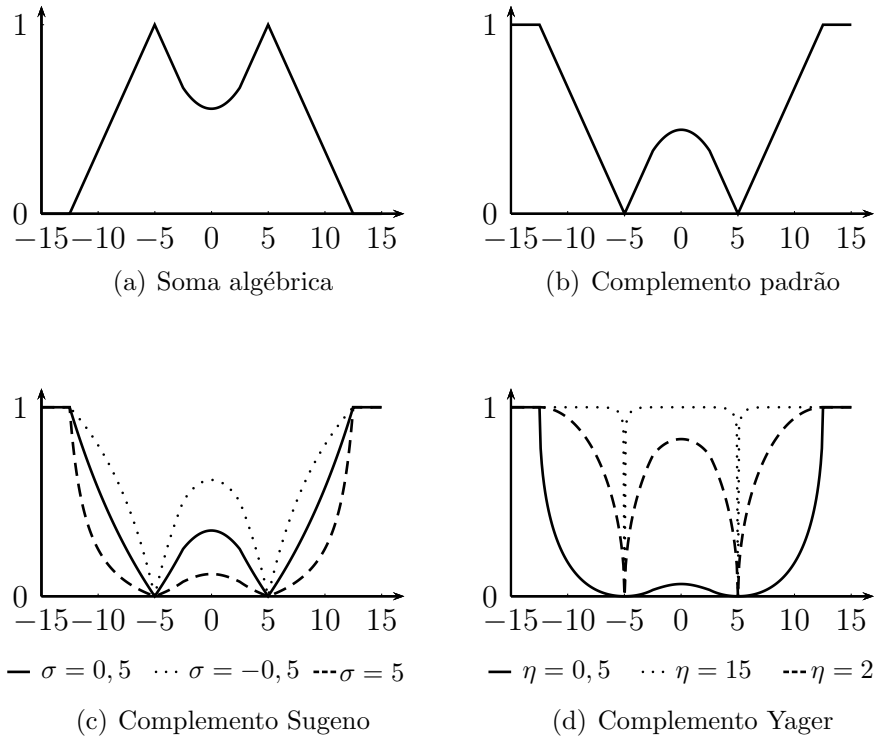
Intuitivamente, espera-se que os elementos do conjunto nebuloso obtido como o complemento de outro conjunto nebuloso, sejam aqueles elementos, no conjunto referencial, que não apresentem esta característica, com certo grau. Os operadores de complemento mais comuns para conjuntos nebulosos são *complemento padrão*, *complemento de Sugeno* e *complemento de Yager*. Estes operadores são definidos nas Equações 12, 13 e 14, respectivamente. Como exemplo, a Figura 6 mostra o complemento da união dos

conjuntos nebulosos com funções de pertinência μ_{gelada} e μ_{fria} da Figura 2, usando os operadores de complemento introduzidos.

$$\eta(\mu_A(x)) = 1 - \mu_A(x) \qquad \forall x \in \mathcal{U} \tag{12}$$

$$\eta_\sigma(\mu_A(x)) = \frac{1-\mu_A(x)}{1+\sigma\times\mu_A(x)} \qquad \forall x \in \mathcal{U} \tag{13}$$

$$\eta_\upsilon(\mu_A(x)) = (1 - \mu_A(x)^\upsilon)^{\frac{1}{\upsilon}} \qquad \forall x \in \mathcal{U} \tag{14}$$



(a) Soma algébrica (b) Complemento padrão

(c) Complemento Sugeno (d) Complemento Yager

Figura 6: Complemento da união via soma algébrica das funções de pertinência μ_{gelada} e μ_{fria} definidas na Figura 2

1.1.2.4 Produto cartesiano

O produto Cartesiano $A\times B$ de dois conjuntos nebulosos A em X e B em Y é um conjunto nebuloso $A \times B$ cuja função de pertinência é $\mu_{A\times B}(x, y) = \min(\mu_A(x), \mu_B(y))$ com $x \in X$ e $y \in Y$.

1.2 Relações Nebulosas

Uma *relação* $R^c : A \times B \rightarrow [0,1]$ estabelece um vínculo entre um elemento $a \in A$ e um elemento $b \in B$ de acordo com a relação R^c, se $(a,b) \in R^c$, caso contrário $(a,b) \notin R^c$. Então, uma relação pode ser vista como um subconjunto de pares (a,b) do produto Cartesiano $A \times B$. Como pode-se supor, uma *relação nebulosa* R^f estabelece o grau com o qual um elemento $a \in A$ está relacionado ao elemento $b \in B$ com respeito à relação R^r. Então, uma relação nebulosa pode ser vista como um conjunto de pares $((a,b), \mu_{R^r}(a,b))$, onde $(a,b) \in A \times B$. A relação nebulosa $R^r : A \times B \rightarrow [0,1]$ pode ser representada por uma matriz $|A| \times |B|$ com o mesmo nome, onde $R^r_{ij} = \mu_{R^r}(i,j)$ (NEDJAH; MOURELLE, 2005).

A composição de relações é definida usando operações do tipo *max-min* (ZADEH, 1988; ZIMMERMANN; ZYSNO, 1980). Seja $\bullet$ o operador de composição da relação nebulosa. Sejam $R : A \times B \rightarrow [0,1]$ e $S : B \times C \rightarrow [0,1]$ duas relações nebulosas. A função de pertinência μ_T da relação nebulosa $T : A \times C \rightarrow [0,1]$, tal que $T = R \bullet S$, é definida conforme a Equação 15.

$$\mu_T(a,c) = \max_{b \in B}(\min(\mu_R(a,b), \mu_S(b,c))), \forall a \in A, \forall b \in B, \forall c \in C \tag{15}$$

1.3 Regras Nebulosas e Inferência

Uma regra nebulosa é simplesmente uma relação entre o antecedente e consequente da regra. Regras são, normalmente, declarações condicionais do tipo *se-então*. Portanto, a proposição $P(x) \Rightarrow Q(y)$ é, normalmente, escrita como, com certo abuso da notação, se $P(x)$ então $Q(y)$.

Tradicionalmente, a *inferência* em termos convencionais é um processo para obter conclusões a partir de um conjunto de fatos e regras. Uma regra dispara quando sua premissa é satisfeita. Em contraste com o processo de inferência tradicional, a inferência nebulosa usa as regras nebulosas e os princípios do raciocínio.

1.4 Controladores Nebulosos

Controle nebuloso, o qual diretamente usa as regras nebulosas é a aplicação mais comum da teoria de conjuntos nebulosos (UMBERS; KING, 1980). Este capítulo trata do

uso de conjuntos nebulosos para desenvolver controladores nebulosos e o seu funcionamento. Como exemplo de aplicação considera-se o pêndulo invertido. Trabalhos sobre controladores nebulosos implementados em FPGA são também enfatizados.

1.4.1 Estrutura

Usando o procedimento originado por E. Mamdani (MAMDANI; ASSILIAN, 1975), três passos são usados para projetar um controlador nebuloso:

1. *fuzzificação* ou codificação: Este passo é responsável por codificar os valores escalares, ditos *crisp*, medidos das variáveis de entrada do sistema;

2. *inferência*: Este passo consiste em identificar o subconjunto das regras nebulosas que estão ativas, ou seja, aquelas com proposições de antecedentes não zero, e inferir as conclusões nebulosas adequadas;

3. *defuzzificação* ou decodificação: Este é o processo reverso da fuzzificação. Ele é responsável pela decodificação das variáveis linguísticas e o cálculo dos respectivos valores escalares. Estes valores representam as saídas do sistema.

A arquitetura de um controlador nebuloso é mostrada na Figura 7. Seus componentes principais consistem de uma *base de conhecimento*, o *codificador* ou *fuzzificador*, o *decodificador* ou *defuzzificador* e o *motor de inferência*. A base de conhecimento armazena as seguintes informações: as regras nebulosas, as quais são usadas pelo motor de inferência para chegar a valores esperados, dados relativos aos termos linguísticos, com suas respectivas funções de pertinência, o universo de discurso de cada variável linguística, etc. O codificador implementa a transformação de escalar para nebuloso e o decodificador a transformação de nebuloso para escalar. O motor de inferência é o componente principal da arquitetura do controlador, uma vez que implementa o mecanismo de raciocínio aproximado.

1.4.2 Operação

A fim de explicar, minuciosamente, como um controlador nebuloso opera, é usado como exemplo o controle de um pêndulo invertido de (BERNSTEIN, 2004). A ideia do pêndulo invertido é mostrado na Figura 8. O problema consiste em controlar o movimento do

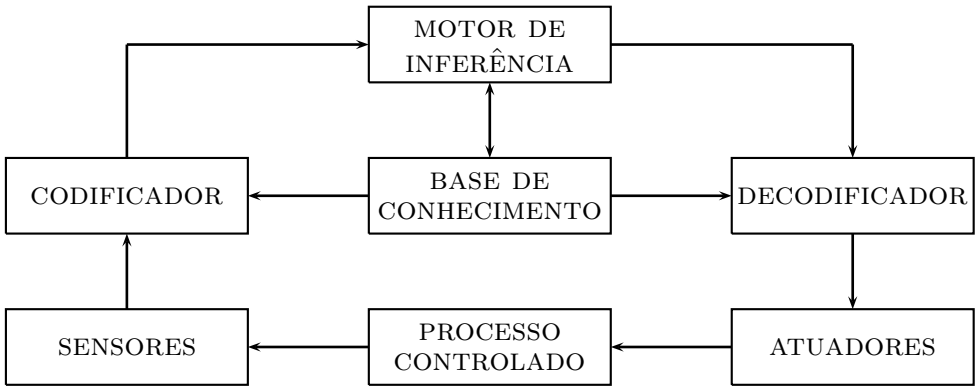


Figura 7: Arquitetura de um controlador nebuloso

pêndulo em uma plataforma móvel para que sua posição angular seja zero. O movimento do carro se dá apenas para a esquerda ou direita.

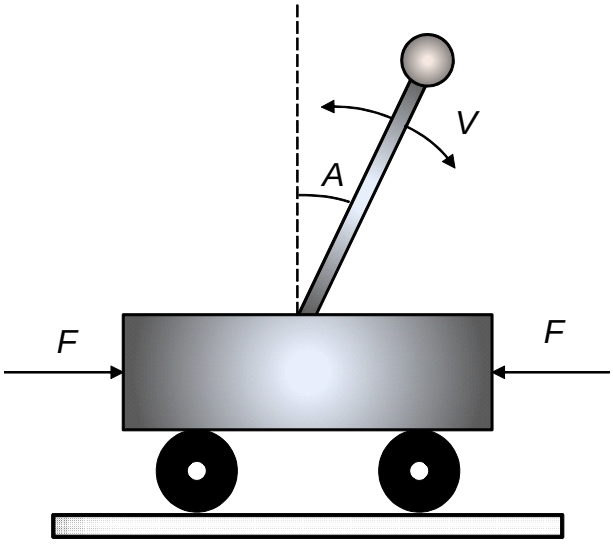


Figura 8: Pêndulo Invertido

As variáveis linguísticas são a posição angular chamada *ângulo* denotada por $A \in [-30°, 30°]$, a *velocidade* angular denotada por $V \in [-15, 15]$ em $°/s$ e a *força* $F \in [-3, 3]$ em N. As primeiras duas variáveis A e V são as entradas do controlador, enquanto a terceira F é a saída esperada. Os termos linguísticos para cada uma das variáveis identificadas são dados na Tabela 1. As funções de pertinência associadas aos termos das variáveis linguísticas *ângulo*, *velocidade* e *força* são definidas nas Figuras 9, 10 e 11, respectivamente.

A coleção de regras nebulosas, as quais são usadas para controlar o pêndulo, são dadas, em formato tabular, na Tabela 2. As regras do controlador são, geralmente, projetadas por um especialista no processo para alcançar o objetivo desejado. Por exemplo,

Tabela 1: Variáveis linguísticas e termos linguísticos correspondentes

ângulo (A)	*velocidade* (V)	*força* (F)
n-grande	*n-alta*	*n-grande*
n-pequeno	*n-baixa*	*n-pequena*
insignificante	*zero*	*zero*
p-pequeno	*p-baixa*	*p-pequena*
p-grande	*p-alta*	*p-grande*

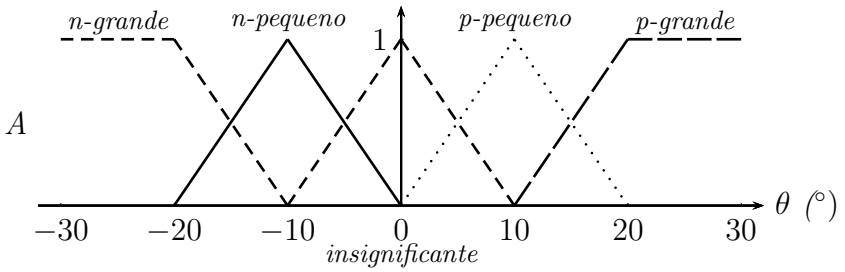


Figura 9: Termos linguísticos e funções de pertinência da variável *ângulo*

o termo na segunda linha e quarta coluna é lido como a regra r_2 apresentada em (16), enquanto o termo na terceira linha e terceira coluna é lido como a regra r_5 apresentada em (17).

r_2: Se *ângulo* é *p-pequeno*e*velocidade* é *n-baixa* então *força* é *zero* (16)

r_5: Se *ângulo* é *insignificante*e*velocidade* é *zero* então *força* é *zero* (17)

A seguir, é mostrado como aplicar as regras da Tabela 2 para valores específicos de medidas de posição e velocidade angular para as variáveis linguísticas *ângulo* e *velocidade*, respectivamente. Assumindo que o valor lido para a variável *ângulo* seja 6° e que para a variável *velocidade* seja $-1°/s$. Os valores das funções de pertinência para a variável *ângulo* são $\mu_{insignificante}(6) = 0,4$ e $\mu_{p\text{-}pequeno}(6) = 0,6$, enquanto para a variável *velocidade*

Tabela 2: Regras nebulosas do controlador

regras		Ângulo				
		n-grande	*n-pequeno*	*insignificante*	*p-pequeno*	*p-grande*
Velocidade	*n-alta*	–	–	r_0: *n-grande*	–	–
	n-baixa	–	–	r_1: *n-pequena*	r_2: *zero*	–
	zero	r_3: *n-grande*	r_4: *n-pequena*	r_5: *zero*	r_6: *p-pequena*	r_7: *p-grande*
	p-baixa	–	r_8: *zero*	r_9: *p-pequena*	–	–
	p-alta	–	r_{10}: *p-grande*	–	–	–

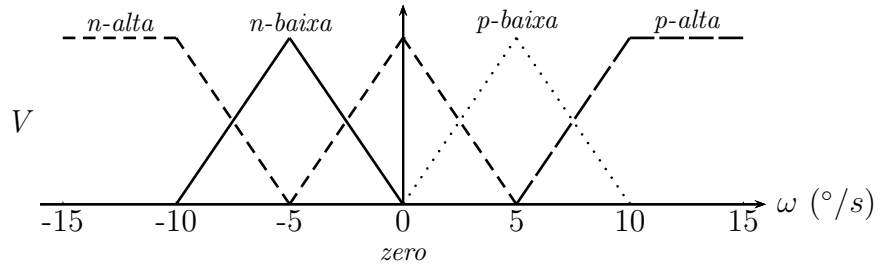


Figura 10: Termos linguísticos e funções de pertinência da variável *velocidade*

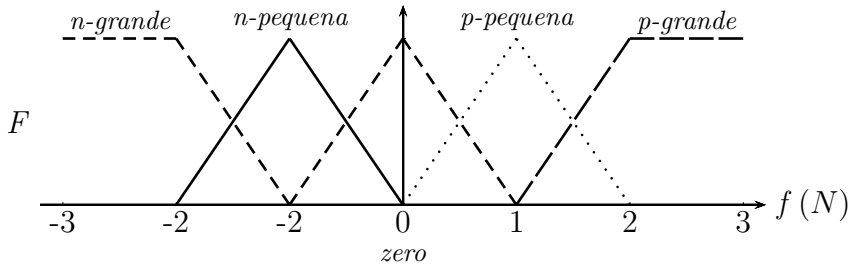


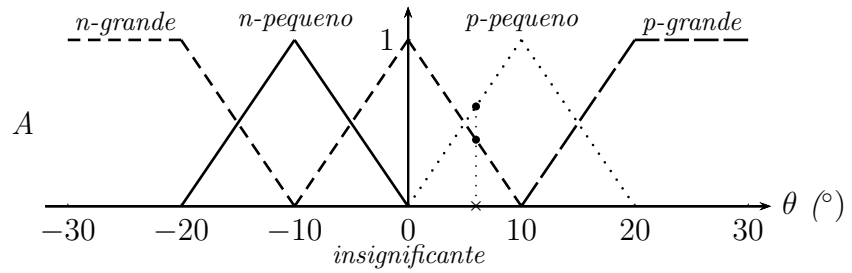
Figura 11: Termos linguísticos e funções de pertinência da variável *força*

$\mu_{zero}(-1) = 0,2$ e $\mu_{p\text{-}baixa}(-1) = 0,8$. Estes pontos estão marcados nos gráficos da Figura 12.

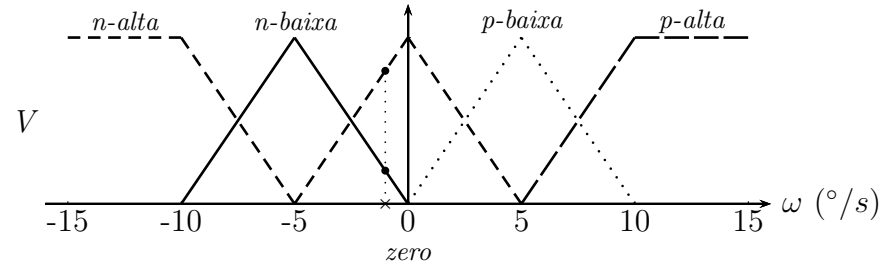
As regras que se aplicam são aquelas que têm seus graus de pertinência diferentes de zero. Então, concluí-se que todas as regras com antecedentes envolvendo os termos linguísticos *insignificante* e/ou *p-pequeno* e *zero* e/ou *n-baixa* deveriam ser disparadas. Da Tabela 2, pode-se identificar que as regras que deveriam ser usadas são aquelas cujo consequente está marcado com um retângulo. As quatro regras também estão listadas nas Equações (18).

$$\begin{aligned}
&r_1\text{: Se } \textit{ângulo} \text{ é } \textit{insignificante} \text{ e } \textit{velocidade} \text{ é } \textit{n-baixa} \text{ então } \textit{força} \text{ é } \textit{n-pequena}\\
&r_2\text{: Se } \textit{ângulo} \text{ é } \textit{p-pequeno} \text{ e } \textit{velocidade} \text{ é } \textit{n-baixa} \text{ então } \textit{força} \text{ é } \textit{zero}\\
&r_5\text{: Se } \textit{ângulo} \text{ é } \textit{insignificante} \text{ e } \textit{velocidade} \text{ é } \textit{zero} \text{ então } \textit{força} \text{ é } \textit{zero}\\
&r_6\text{: Se } \textit{ângulo} \text{ é } \textit{p-pequeno} \text{ e } \textit{velocidade} \text{ é } \textit{zero} \text{ então } \textit{força} \text{ é } \textit{p-pequena}
\end{aligned} \tag{18}$$

Usando a definição de Mandani para uma regra nebulosa, é possível aplicar as regras em (18). A aplicação das regras selecionadas produz os conjuntos nebulosos descritos



(a) ângulo real



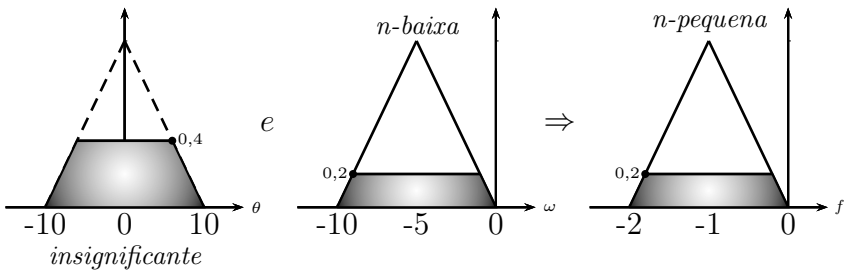
(b) velocidade real

Figura 12: Funções de pertinência para os valores escalares das variáveis *ângulo* (A) e *velocidade* (V)

nas Figuras 13, 14, 15 e 16, relativos as regras r_1, r_2, r_5 e r_6, respectivamente. Neste exemplo, sempre o operador mínimo é usado.

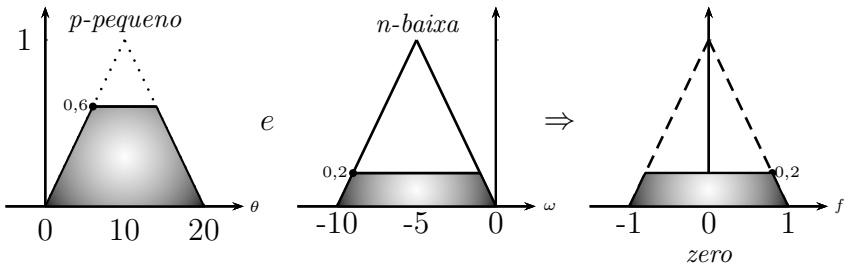
Como as regras disparadas r_2 e r_5 produzem o mesmo termo linguístico como consequente, pode-se combiná-los usando um operador OR ou pela teoria de conjuntos nebulosos, uma união como operador máximo. A Figura 17 mostra este processo.

A defuzzificação a ser selecionada para a obtenção do resultado de saída do controlador depende da natureza do processo que está sendo controlado. Existem várias técnicas



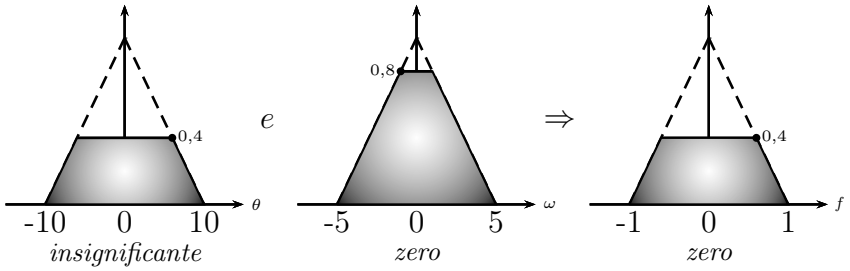
(a) $\mu_{insignificante(6)}$ (b) $\mu_{n\text{-}baixa(-1)}$ (c) $\mu_{n\text{-}lenta}$

Figura 13: Representação do resultado da aplicação da regra r_1



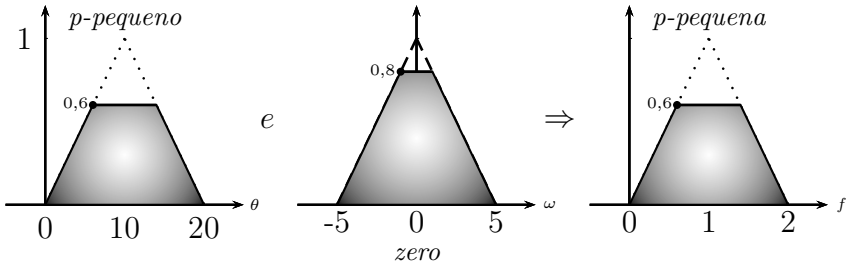
(a) $\mu_{p\text{-}pequeno(6)}$ (b) $\mu_{n\text{-}baixa(-1)}$ (c) μ_{zero}

Figura 14: Representação do resultado da aplicação da regra r_2



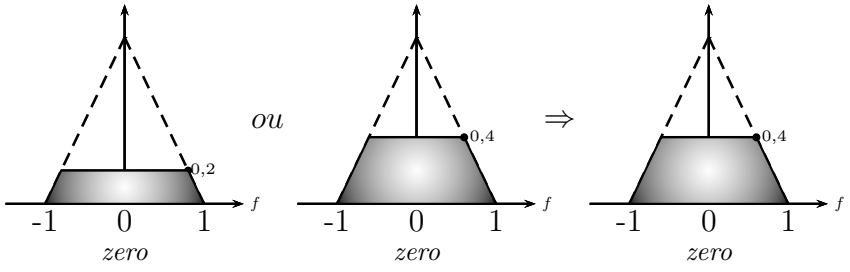
(a) $\mu_{insignificante(6)}$ (b) $\mu_{zero(-1)}$ (c) μ_{zero}

Figura 15: Representação do resultado da aplicação da regra r_5



(a) $\mu_{p\text{-}pequeno(6)}$ (b) $\mu_{zero(-1)}$ (c) $\mu_{p\text{-}pequena}$

Figura 16: Representação do resultado da aplicação da regra r_6



(a) Resultado de r_2 (b) Resultado de r_5 (c) Resultado final

Figura 17: Combinando os resultados da aplicação de r_2 e r_5

para obter um valor numérico representativo do conjunto nebuloso da saída. Para todos os tipos formulados abaixo, x_i é o valor no eixo horizontal de um determinado ponto i e $\mu(x_i)$ é o seu grau de pertinência. Geralmente, as técnicas utilizadas são de dois tipos, são elas:

- *centro de gravidade*: calcula o valor u correspondente a abscissa do centro de gravidade do conjunto nebuloso obtido, onde ℓ é o número de pontos ao longo do eixo horizontal (supondo funções de pertinência discretizadas). No caso de triângulos isósceles é possível usar apenas os pontos máximos de cada termo linguístico. Este cálculo para defuzzificação é descrito na Equação 19;

$$u = \frac{\sum_{i=1}^{\ell} \mu(x_i)x_i}{\sum_{i=1}^{\ell} \mu(x_i)} \tag{19}$$

- *composição dos máximos*, o qual é baseado no valor do conjunto nebuloso com o maior grau de pertinência. A técnica de composição dos máximos pode usar:

  - média dos máximos: é a média de todos os valores que possuem o maior grau de pertinência no conjunto nebuloso obtido. O cálculo para a defuzzificação é descrito na Equação 20, onde ℓ representa no número de pontos com maior grau de pertinência no conjunto nebuloso;

$$u = \frac{\sum_{i=1}^{\ell} \{x|\ \mu(x) = \max(x_i)\}}{\ell} \tag{20}$$

  - primeiro máximo: é o menor valor entre os maiores graus de pertinência no conjunto nebuloso obtido. O cálculo para a defuzzificação é descrito na Equação 21;

$$u = \{x|\ \mu(x) = \max(x_i)\ e\ x < x_i, \forall x_i, i = 1 \ldots \ell\} \tag{21}$$

  - último máximo: é o maior valor entre os maiores graus de pertinência no conjunto nebuloso produzido. O cálculo para a defuzzificação é descrito na Equação 22.

$$u = \{x|\ \mu(x) = \max(x_i)\ e\ x > x_i, \forall x_i, i = 1 \ldots \ell\} \tag{22}$$

Cada uma das técnicas detalhadas nas Equações 19, 20, 21 e 22 é ilustrada na Figura 18 por diferentes pontos, conforme a legenda.

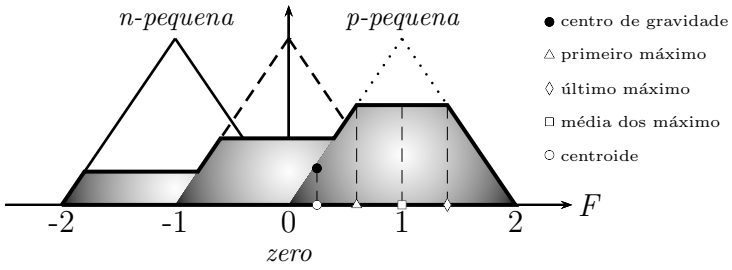


Figura 18: Compondo os resultados de todas as regras disparadas

1.5 Considerações Finais do Capítulo

Neste capítulo foi apresentado a teoria de conjuntos nebulosos e seus principais operadores para utilização da mesma como uma ferramenta para controladores baseados no raciocínio aproximado. Além da apresentado do funcionamento de um controlador nebuloso baseado no conhecido exemplo do pêndulo invertido, perfazendo os passos de fuzzificação, em seguida de inferência e finalmente de defuzzificação, detalhando o funcionamento e os cálculos realizados em cada passo. Este conhecimento servirá de base para a descrição do controlador proposto neste trabalho. O pêndulo invertido foi utilizado para demonstrar o funcionamento da arquitetura proposta via simulação. O capítulo seguinte apresenta, resumidamente, trabalhos sobre a implementação de controladores nebulosos em FPGA.

Capítulo 2

TRABALHOS RELACIONADOS

ESTE capítulo apresenta os trabalhos sobre controladores nebulosos desenvolvidos para FPGA. Todos os artigos encontrados com este foco, não possuem todas as informações detalhadas de suas aplicações, impossibilitando a implementação com o CNP e posterior comparação do consumo de área, tempo de execução e precisão nos resultados. Os controladores descritos nestes trabalhos foram desenvolvidos para aplicações específicas ou possuem alguma limitação para sua reconfiguração, ao contrário do CNP que possui uma arquitetura genérica e escalável.

2.1 Controladores em *Firmware*

O trabalho proposto em (MACHADO et al., 2011) propõe a automação de uma centrífuga no processo de fabricação de álcool de uma empresa no setor sucroalcooleiro, onde existe uma tendência de perdas no rendimento do processo. Esta automação é realizada por meio de uma FPGA programada com o processador NIOS II (NIOS..., 2013). O controlador nebuloso é implementado como um *firmware* para ser executado pelo processador. A síntese e programação foram feitos pela ferramenta SOPC *Builder* no *software* QUARTUS II da Altera. O controlador proposto possui 2 entradas, de 8 bits de resolução cada uma, e 1 saída de 1 único bit de resolução, pois o sinal é gerado em PWM para o controle de abertura de uma válvula. Todas as variáveis possuem funções de pertinência com mesma forma. Estas são modeladas utilizando 7 termos linguísticos com funções de pertinência triangulares.

Esta arquitetura proposta utilizou uma área relativamente reduzida, pois o cerne do controlador nebuloso não foi projetado em *hardware*, mas sim em *firmware*. A concepção deste controlador é simplificada, pois possui apenas duas etapas: a fuzzificação e a

defuzzificação, onde a inferência é realizada dentro da etapa de fuzzificação. Este trabalho não apresenta as regras utilizadas, bem como a superfície de controle do controlador nebuloso, impossibilitando a reprodução no CNP. O controlador foi simulado no MATLAB para análise e validação antes da implementação no processador. Este controlador, não é plenamente reconfigurável, pois para alterar a quantidade de regras, termos linguísticos e número de entradas e saídas é necessário reprogramar o processador NIOS II. É possível apenas alterar as características das funções de pertinência e as regras, sendo que o número total de regras é fixo e não pode ser alterado.

2.2 Controladores em *Hardware*

O trabalho proposto em (ALVAREZ et al., 2006) desenvolve o controle de um conversor DC-DC multifásico para utilização em carros, onde a quantidade de equipamentos elétricos e eletrônicos vem aumentando consideravelmente e a utilização deste conversor permitiria o uso de baterias - alternadores de 42V com os atuais equipamentos de 14V. Nesta aplicação, o controlador proposto possui 2 variáveis de entrada e 1 variável de saída, todas com resolução de 16 bits. As variáveis possuem entre 5 e 7 termos linguísticos. O controlador é composto por 21 regras das 35 possíveis.

Para a implementação do controlador, foi utilizada a FPGA da Xilinx XC3S2000 e a utilização de área de *hardware* para esta aplicação aparenta ser mais compacta em relação ao CNP. Isso é, provavelmente, devido a não utilização de cálculos em ponto flutuante, o que permite uma maior precisão nos cálculos. Utilizando a *toolbox System Generator* da Xilinx para o simulink dentro do MATLAB, o controlador desenvolvido em VHDL foi verificado com um modelo não-linear do conversor multifásico desenvolvido pelo autor. O controlador proposto não permite a reconfiguração da sua estrutura, já que o foco da implementação é uma aplicação específica.

O trabalho proposto em (HASSAN; SHARIF, 2007) apresenta um controlador nebuloso do tipo PID (*Proporcional Integral Derivativo*) para aplicações industriais de forma genérica em FPGA. O controlador proposto tem a possibilidade de se comportar como um controlador PI, PD e PID, sendo a escolha feita por meio de um seletor de 2 bits.

Na arquitetura, existem 2 controladores nebulosos: o primeiro trabalha como um controlador PD e o segundo como um controlador PI. No caso da seleção do controlador para PID, é realizada uma soma aritmética das saídas dos 2 controladores. Cada contro-

lador possui as mesmas 2 entradas e 1 saída para cada, todos de 8 bits de resolução. Cada controlador é formado por 4 componentes: um bloco de ganho para as entradas e a saída; um bloco de fuzzificação; um de inferência; e, finalmente, um de defuzzificação. Nesta arquitetura os dados são representados em excesso de 128. O bloco de ganho das entradas soma o excesso de 128 e o bloco de saída subtrai este excesso. Este controlador possui algumas limitações. As funções de pertinência podem ter no máximo 8 termos linguísticos com sobreposição fixa de 50%, ou seja, no máximo 2 termos linguísticos estarão ativos e o grau de pertinência do segundo termo é dado por $\mu_1 = 1 - \mu_0$, simplificando os cálculos a serem feitos. Consequentemente, esta simplificação levou a uma redução na utilização de área de *hardware* e tempo necessário para a execução.

O trabalho proposto em (CHEKIRED et al., 2011) visa estabelecer o ponto de máxima potência, ou *maximum power point tracking* para sistemas de alimentação fotovoltaicos. Para isso, é utilizado um conversor DC-DC como *drive* entre a célula fotovoltaica e a carga a ser alimentada. O controle atua no ciclo de trabalho do conversor, através de um sinal PWM, de acordo com a relação de potência e tensão geradas pela célula de carga. Esta, por sua vez, gera variações na saída, de acordo com a incidência de luz e temperatura. O controlador nebuloso possui 2 entradas e 1 saída, de 8 bits de resolução. Todas as variáveis possuem 5 termos linguísticos, com um total de 25 regras utilizadas. Nesta implementação a execução de todas as etapas de controle é realizada em 1/100 Hz. O processo e o controlador nebuloso atingem a estabilidade em, no máximo, 12 ciclos de controle. Não foram apresentadas maiores informações sobre a arquitetura utilizada, nem da possibilidade de reconfiguração.

O trabalho proposto em (MCKENNA; WILAMOWSKI, 2001) desenvolve um controlador de forma pouco comum e simplificado. Basicamente, quando um controlador nebuloso é configurado, através dos 4 principais parâmetros, i.e. número de entradas, base de regras, dados das funções de pertinência e número de saídas, é possível obter, através do MATLAB, a superfície de controle imposta. Neste trabalho, a superfície de controle é traduzida para uma tabela armazenada em uma memória ROM da arquitetura. Desta forma, as variáveis de entrada, cada uma de 7 bits, são tratadas diferentemente de um controlador nebuloso padrão, onde os 4 bits mais significativos são tratados como endereço da memória ROM e os 3 bits menos significativos são usados para calcular uma média ponderada. Os valores lidos da memória, que correspondem à superfície de con-

trole são usados como pesos no cálculo mencionado, cujo resultado representa a saída do controlador. Esta possui 16 bits de resolução. Foi mostrado que, o mecanismo funciona corretamente, apesar de o dispositivo desenvolvido não possuir nenhuma operação de conjuntos nebulosos. Um ponto negativo para esta arquitetura é o aumento de tamanho e complexidade da memória de forma exponencial, conforme a quantidade de entradas e saídas aumenta.

Em (KIM, 2000), propõe-se uma ferramenta de implementação, que permite gerar o código VHDL do controlador nebuloso desejado, tendo em vista as características informadas através de um programa escrito em C. No qual, o controlador nebuloso é divido em vários módulos funcionais independentes temporalmente. Cada módulo é implementado individualmente no sistema automático de implementação e projeto, o qual é um ambiente de desenvolvimento integrado para a execução de várias sub-tarefas, como a criação automática do VHDL, síntese em FPGAs, otimização, alocação, roteamento e *download*. Cada módulo implementado forma um *hardware* que pode se transferido individualmente para a FPGA. Neste caso, a fuzzificação é realizada através de uma memória, onde as funções de pertinência são discretizadas e armazenadas em forma de tabela. Os valores das variáveis de entrada são usados como endereços nessa memória, retornando o termo linguístico ativo junto com o grau de pertinência associado. O controlador foi configurado para controlar a trajetória de um carro. O controlador nebuloso possui 2 entradas e 1 saída, de 8 bits de resolução cada. Tendo entre 5 e 7 termos linguísticos. O trabalho não apresenta com clareza a utilização de área de *hardware* deste controlador, nem mesmo, o tempo necessário para execução.

2.3 Considerações Finais do Capítulo

Neste capítulo foram apresentados, resumidamente, diversos trabalhos sobre implementação de controladores nebulosos em FPGA. Vale lembrar que nenhum dos trabalhos apresentados utiliza um controlador em *hardware* com cálculos em ponto flutuante. O capítulo seguinte apresenta as arquiteturas macro e micro do controlador proposto nesta dissertação.

Capítulo 3

A ARQUITETURA PROPOSTA

NESTE capítulo é apresentado o controlador nebuloso proposto (CNP), foco de pesquisa deste trabalho. Primeiramente, é descrita a macro-arquitetura, incluindo sua máquina de estados, com os componentes principais. Em seguida, cada componente é detalhado em termos de arquitetura, máquina de estados, funcionamento e suas características relativas à reconfiguração do CNP.

3.1 Macro-arquitetura

A arquitetura do CNP consiste de três unidades principais: *(i)* a unidade de fuzzificação (`UF`), a qual é responsável por traduzir os valores escalares das entradas do sistema para as variáveis linguísticas adequadas, usando suas respectivas funções de pertinência. Esta unidade possui tantos blocos `Fuzzy` quantos necessários para o controle do processo, ou seja, um para cada variável de entrada; *(ii)* a unidade de inferência (`Inferência`), a qual testa todas as regras nebulosas, verificando quais termos linguísticos estão ativos, gerando seus respectivos graus de pertinência e identificando o termo linguístico a ser usado na sequência; *(iii)* a unidade de defuzzificação (`UD`), a qual é responsável por traduzir os termos nebulosos em um valor escalar representativo para fornecer a saída do controlador nebuloso. A unidade de defuzzificação possui tantos blocos `Defuzzy` quantos necessários para o controle do processo em questão, ou seja, um para cada variável de saída. O diagrama de blocos da macro-arquitetura do CNP é mostrado na Figura 19, onde N e M representam o número de variáveis de entradas e saídas, respectivamente. (SANDRES; NEDJAH; MOURELLE, 2011) (SANDRES; NEDJAH; MOURELLE, 2012)

Além das unidades principais, a arquitetura também inclui um componente que permite determinar as características das funções de pertinência, as quais são usadas por

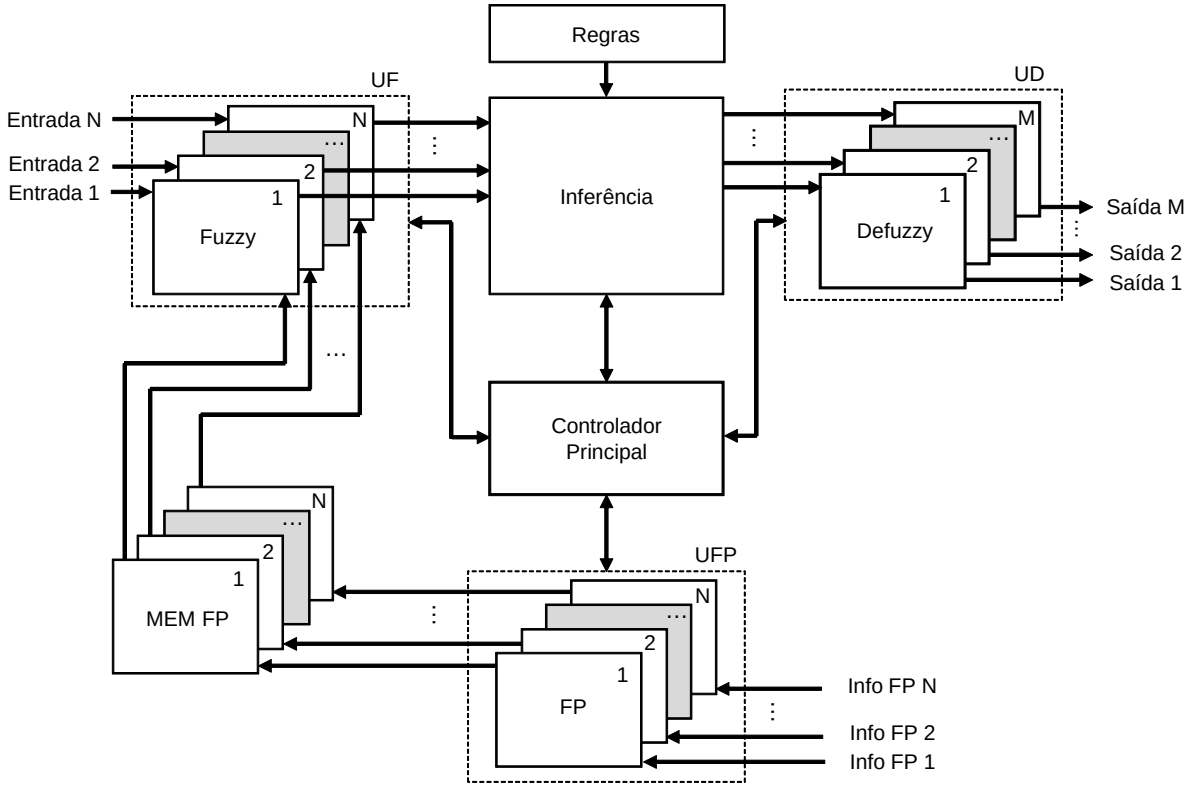


Figura 19: Arquitetura do controlador nebuloso desenvolvido

ambas as unidades de fuzzificação e defuzzificação. Na Figura 19 este componente é identificado de unidade de função de pertinência (`UFP`). Esta unidade possui tantos blocos `FP` quantos necessários, ou seja, um para cada variável de entrada. Todas as informações relativas às funções de pertinência são armazenadas na memória de função de pertinência, identificada por `MEM FP`. Esta memória é formada por tantos segmentos de memória quantos necessários pelas variáveis linguísticas correspondentes às entradas do CNP. As regras usadas pela unidade de inferência são armazenadas em um bloco de memória somente leitura, identificado por `Regras`. O componente `Controlador Principal` executa a sequência necessária e/ou simultaneidade dos passos necessários do controlador nebuloso através de uma máquina de estados finita concorrente. Mais detalhes sobre este componente são fornecidos mais a frente, nesta seção.

O CNP é projetado para ser genérico e parametrizável, e por tanto, escalável, de modo que permita configurar a quantidade de variáveis de entrada e saída, a quantidade de termos linguísticos usados, as respectivas funções de pertinência e a quantidade de regras nebulosas, assim como o tipo de controle nebuloso que está sendo implementado. A possibilidade da configuração destes parâmetros torna possível, bem como fácil, ajustar o controlador para qualquer aplicação desejada. Resumindo, os principais parâmetros do

controlador são:

- N: Número de variáveis de entrada e, portanto, de blocos `Fuzzy`;
- M: Número de variáveis de saída e, portanto, de blocos `Defuzzy`;
- P: Número de regras e, portanto, do número de palavras na base de regras `Regras`;
- Q: Número de termos linguísticos usados para granularizar os universos das variáveis de entrada e saída do controlador nebuloso. Note que para a implementação atual, este parâmetro é o mesmo para todas as entradas e saídas. Entretanto, isto pode ser facilmente alterado, de tal forma que permita diferentes modelos para funções de pertinências distintas. Portanto, este parâmetro rege o número de palavras da memória `MEM FP`.

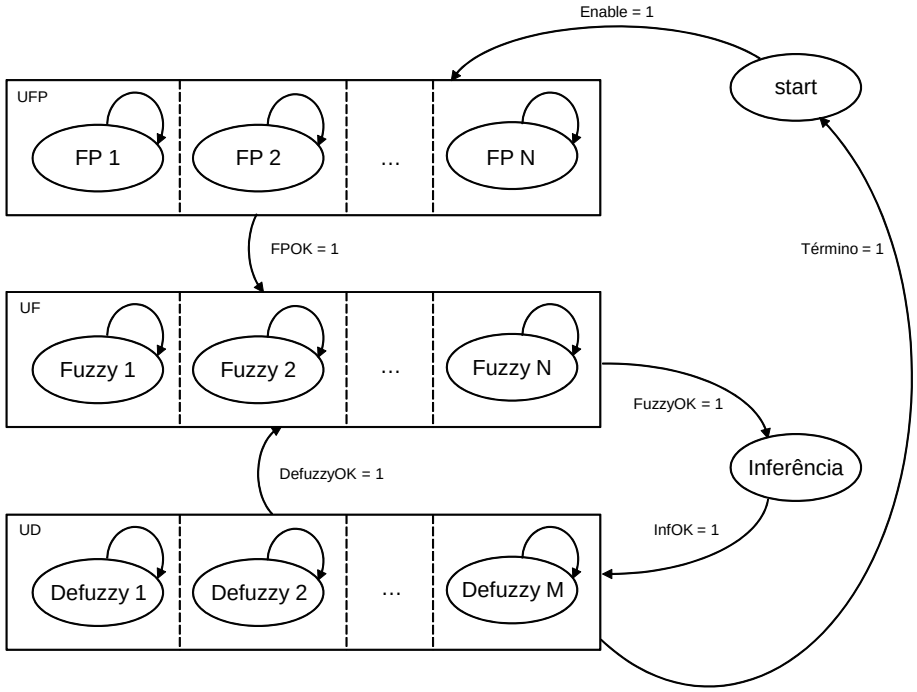


Figura 20: Hierarquia e concorrência da máquina de estados do controlador principal

No momento da configuração, todas as funções de pertinência utilizadas pelo controlador são definidas e armazenadas nos respectivos segmentos da memória `MEM FP` de função de pertinência. Em sequência, todos os dados calculados, que serão apresentados a seguir, estarão prontamente disponíveis para serem usados pelos blocos `Fuzzy` e/ou `Defuzzy` na unidade de fuzzificação e defuzzificação, respectivamente. Em consequência

disto, este passo denominado de *configuração*, é executado apenas uma vez. Após a execução do passo de configuração, na etapa seguinte, denominada de *operação*, o controlador nebuloso executa os passos acionando os blocos `Fuzzy`, em seguida, a unidade `Inferência` e, em seguida, os blocos `Defuzzy`, em sequência. Depois disso, o CNP aguarda por um novo conjunto de dados de entrada a serem lidos pelos sensores do sistema e, assim, chegar às portas de entrada dos blocos `Fuzzy`. As máquinas de estados finitos que controlam os blocos `Fuzzy` são todas executadas em paralelo, assim como as que controlam os blocos `Defuzzy`. Portanto, foi usada a linguagem de descrição de *Statechart* (HAREL, 1987) para expressar os aspectos hierárquicos e concorrentes da máquina de estados da operação global do controlador principal, mostrada na Figura 20. As funções dos sinais usados nessa figura serão introduzidas na descrição das micro-arquiteturas de cada componente.

3.2 Micro-arquitetura das Unidades Funcionais

Esta seção, descreve a micro-arquitetura dos componentes principais, mostrados na macro-arquitetura da Figura 19. Estes são: o componente básico responsável pelo cálculo da função de pertinência (`FP`), incluindo o componente de memória (`MEM FP`); o componente básico responsável pelo processo de fuzzificação (`Fuzzy`); o componente que implementa o processo de inferência (`Inferência`), usando a base de regras (`Regras`); e o componente básico que executa o processo de defuzzificação (`Defuzzy`).

Em geral, todos os blocos que executam cálculos em ponto flutuante possuem uma unidade `FPU` (*Floating Point Unit*), são eles: `FP`, `Fuzzy` e `Defuzzy`. Esta FPU executa as principais operações matemáticas com precisão simples de 32 bits (AL-ERYANI, 2006). As operações necessárias são adição, subtração, multiplicação e divisão.

3.2.1 Unidade de função de pertinência

Uma função de pertinência é um conjunto nebuloso, e está relacionada a um termo linguístico, onde cada um é definido por duas retas. O controlador proposto utiliza funções de pertinência triangular para representar os termos linguísticos. Todavia, é possível modificar o projeto para considerar outras formas como trapézios e sigmoides. A Figura 21 mostra um exemplo de variável linguística com Q termos linguísticos, onde o eixo horizontal representa o universo da entrada do controlador e o eixo vertical representa o grau

de pertinência y associado à uma entrada x. Este é um valor real, no intervalo [0, 1], representado por um número em ponto flutuante precisão simples (32 bits).

Os termos linguísticos com função de pertinência triangular são definidos por 3 parâmetros: PM, IE e ID, representando ponto máximo, intervalo esquerdo e intervalo direito, respectivamente, conforme ilustrado na Figura 21. Para cada reta, min e max representam os limites mínimo e máximo, na Figura 21 é possível ver esses limites para a reta esquerda do termo linguístico 2, em negrito. Neste caso, min será equivalente à $PM_2 - IE_2$ e max será igual ao valor de PM_2.

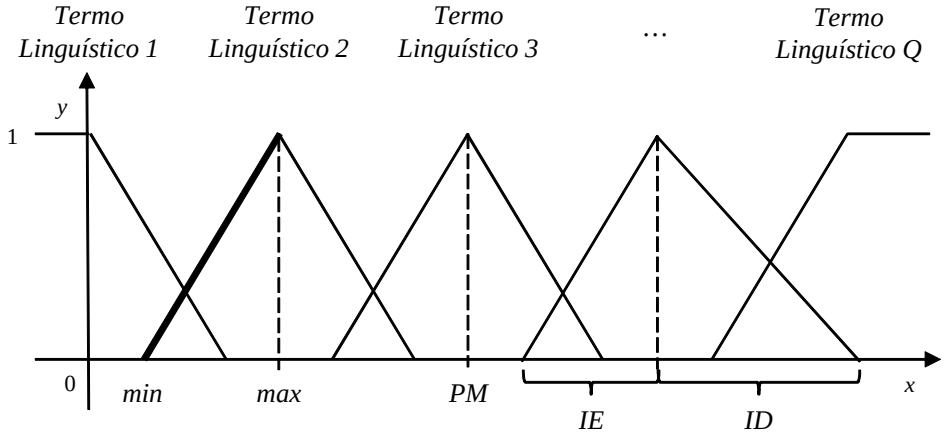


Figura 21: Variável linguística de Q termos linguísticos

O bloco `FP` é projetado para calcular os valores que definem as retas para qualquer valor da variável x, de acordo com a Equação 23, onde duas retas são usadas para representar a função de pertinência de um termo linguístico. As informações básicas necessárias para definir completamente essas formas precisam ser identificadas.

$$y = ax + b \tag{23}$$

Os valores de entrada do bloco `FP` são PM, IE e ID para cada reta usada para definir a função de pertinência de cada termo linguístico. O bloco os utiliza e calcula os coeficientes a e b e seus limites max e min, adequadamente, e os armazena no segmento de memória da função de pertinência. Três casos são possíveis: termo linguístico mais a esquerda (e.g. termo linguístico 1 na Figura 21); termo linguístico intermediário (e.g. termo linguístico 2 na Figura 21); e finalmente, termo linguístico mais a direita (e.g. termo linguístico Q na Figura 21). O cálculo de a, b, min e max das retas mais a esquerda, do meio e mais a direita são definidos como nas Equações 24, 25 e 26, respectivamente.

$$\mu_e(x) = \begin{cases} 1, & \text{se } x \leq PM \\ -\frac{1}{ID} \times x + \frac{PM}{ID} + 1, & \text{se } PM > x \geq PM + ID \\ 0, & \text{senão} \end{cases} \quad (24)$$

$$\mu_m(x) = \begin{cases} \frac{1}{IE} \times x - \frac{PM-IE}{IE}, & \text{se } PM - IE < x \leq PM \\ -\frac{1}{ID} \times x + \frac{PM}{ID} + 1, & \text{se } PM > x \geq PM + ID \\ 0, & \text{senão} \end{cases} \quad (25)$$

$$\mu_d(x) = \begin{cases} \frac{1}{IE} \times x - \frac{PM-IE}{IE}, & \text{se } PM - IE < x \leq PM \\ 1, & \text{se } x > PM \\ 0, & \text{senão} \end{cases} \quad (26)$$

3.2.1.1 Arquitetura

A micro-arquitetura dos blocos de função de pertinência `FP` é mostrada na Figura 22[1]. É usada uma unidade de cálculo em ponto flutuante (`FPU`) para executar as operações matemáticas necessárias. Os resultados obtidos são armazenados na memória `MEM FP`.

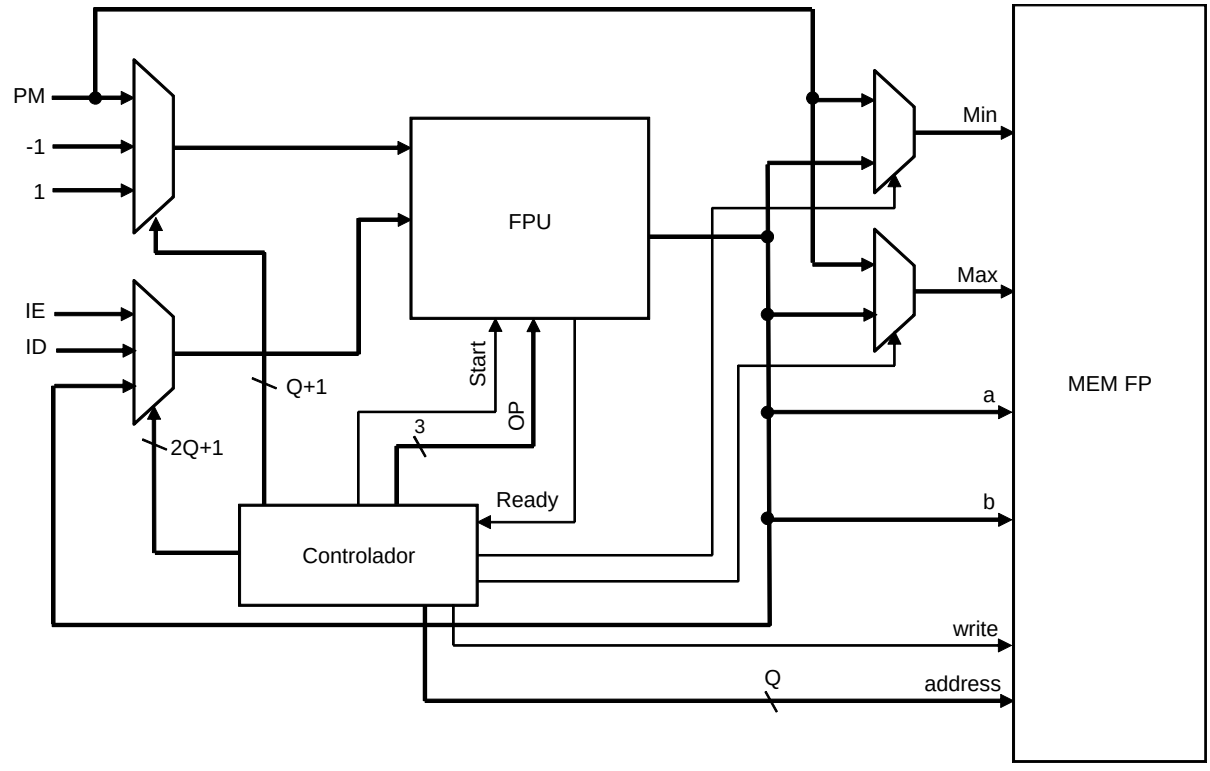


Figura 22: Micro-arquitetura do bloco de função de pertinência (FP)

[1]As linhas finas representam sinais de 1 bit, enquanto as mais grossas representam sinais de 32 bits

Conforme mostrado na Figura 22, a unidade FPU possui os seguintes sinais de entrada: *input1* é o primeiro operando; *input2* é o segundo operando; *OP* representa o código da operação matemática a ser calculada; *Start* é o comando para iniciar o cálculo desejado. Já os sinais de saída: *Ready* indica o status de término do cálculo; e finalmente, *output* que é o resultado do calculo realizado.

O bloco FP possui um controlador que é implementado por uma máquina de estados finitos, cujo diagrama de transição é mostrado na Figura 23. Esta máquina é otimizada para ter o menor número de estados possíveis. Isso permite sincronizar a configuração de todos os termos linguísticos, necessários para completar a especificação das funções de pertinência para cada variável de entrada, em um menor tempo possível.

3.2.1.2 Controle

A seguir, é explicado como o bloco da função de pertinência funciona. Quando o bloco FP recebe o comando *enable* do controlador principal (vide Figura 23), a máquina de estados da Figura 23 muda do estado *start* para *test_step*, onde é testado se este estágio corresponde à primeira ou última reta das funções de pertinência a serem calculadas.

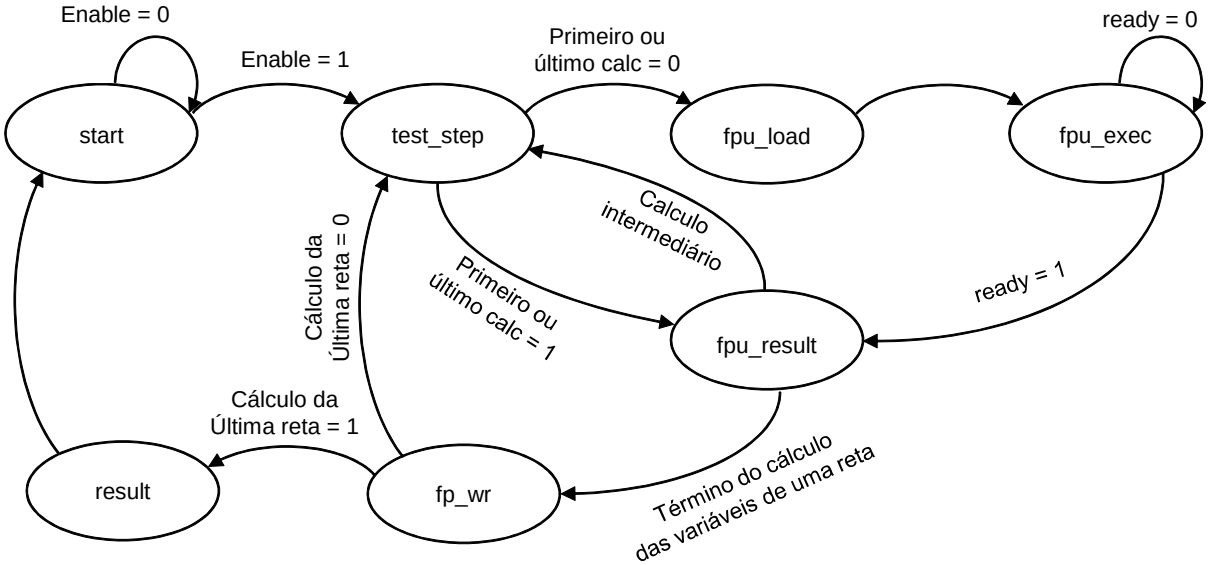


Figura 23: Máquina de estado do bloco de função de pertinência

Se for um destes casos mencionados, não há nada a fazer, porque a primeira e a última retas são constantes, como é possível ver na Figura 21. Portanto, o bloco FPU não é utilizado e a MEF (Máquina de Estados Finitos) vai para o estado *fpu_result*. Caso contrário, ou seja, se esta iteração não é nem a primeira, nem última reta, a MEF muda

Algoritmo 1 Configuração das funções de pertinência

Entrada: PM_i, IE_i e $ID_i,\ i = 1 \ldots Q$;
Saída: min, max, a e b;

```
Se enable = 1 Então
  Address ← 0;
  Para i ← 1 to Q Faça
    Para FP ← 1 to 2 Faça
      Write ← 0;
      Address ← Address + 1;
      Se i = 1 Então
        Se FP = 1 Então
          min ← −∞;
          max ← PM_i;
          a ← 0;
          b ← 1;
        Senão
          min ← PM_i;
          max ← PM_i + ID_i;
          a ← −1/ID_i;
          b ← (PM_i/ID_i) + 1;
      Se 1 < i < Q Então
        Se FP = 1 Então
          min ← PM_i − IE_i;
          max ← PM_i;
          a ← 1/IE_i;
          b ← −((PM_i − IE_i)/IE_i);
        Senão
          min ← PM_i;
          max ← PM_i + ID_i;
          a ← −1/ID_i;
          b ← (PM_i/ID_i) + 1;
      Se i = Q Então
        Se FP = 1 Então
          min ← PM_i − IE_i;
          max ← PM_i;
          a ← 1/IE_i;
          b ← −((PM_i − IE_i)/IE_i);
        Senão
          min ← PM_i;
          max ← +∞;
          a ← 0;
          b ← 1;
      write ← 1;
```

para o estado *fpu_load*, onde os valores de PM, IE e ID que serão usados, são carregados. Após isto, o estado *fpu_exec* é ativado, onde o bloco `FP` aguarda o bloco `FPU` entregar o

resultado. Tão logo ele o faça, a MEF vai para o estado *fpu_result*, onde o resultado é registrado. Note que para cada reta de cada termo linguístico, o bloco `FP` precisa obter quatro resultados, como será detalhado mais a frente na Seção 3.2.2. Consequentemente, se o resultado calculado não é o quarto, a MEF entra no estado *test_step*, e executa o processo de cálculo novamente, alterando os valores de entrada e o código da operação matemática, até obter o resultado das quatro variáveis que definem a reta. Sempre que todos os quatro resultados são calculados e armazenados, a máquina de estados vai para o estado *fp_wr*, onde habilita o comando de escrita em `MEM FP`. O bloco `FP` executa o mesmo processo até que os valores da última reta sejam escritos nos respectivos segmentos da memória `MEM FP`. Uma vez isso feito, a MEF muda para o estado *result*, onde entrega o sinal de término $FPOK_j, j = 1 \ldots N$, volta para o estado *start* e espera por um novo estágio de configuração, se houver. O sinal $FPOK$ gerado pela `UFP`, visto na Figura 20, é habilitado quando todos os $FPOK_j, j = 1 \ldots N$ o são.

O Algoritmo 1 mostra o código equivalente à descrição de *hardware* desenvolvida para o CNP, referente ao bloco de configuração das funções de pertinência, onde baseado nas variáveis de entrada, ponto máximo, intervalo esquerdo e intervalo direito de cada termo linguístico, é possível determinar os valores limites e coeficientes das retas que definem cada triângulo, i.e., termo linguístico. Como esta função é escalável, tem-se uma iteração, baseada no comando PARA e dentro deste, três testes para verificar qual tipo de termo linguístico esta sendo calculado. Os termos linguísticos inicial e final sempre existirão, por isso a limitação de haver no mínimo dois termos linguísticos em cada variável linguística, já o termo intermediário será executado $Q - 2$ vezes, podendo ser zero. Isso permite seu uso para quantos termos linguísticos forem necessários. Após o cálculo das especificações de cada reta (min, max, a e b), estes são armazenados em memória, conforme explicado na próxima seção.

3.2.2 Memória da função de pertinência

Conforme explicado anteriormente, este bloco de memória responde a comandos de escrita recebidos de `UFP` e comandos de leitura gerados por `UF`. Cada palavra desta memória possui quatro informações que permitem o cálculo completo do grau de pertinência correspondente a um dado termo linguístico. A memória é de quatro vias, sendo uma via para cada valor de especificação de uma reta (min, max, a e b).

Toda vez que o bloco FP habilitar um comando de escrita, este bloco de memória registra estes valores em um endereço, que representa o número da ordem da reta dentre todas as retas que precisam ser processadas. Vale lembrar que, existe um bloco de memória MEM FP associado a cada variável linguística de entrada ou bloco Fuzzy, como é possível ver na Figura 19.

Este bloco também permite a configuração do número de retas que podem ser registradas na memória, o qual dependerá do parâmetro Q, que determina o número de termos linguísticos por variável. Note que para cada termo linguístico serão duas palavras armazenadas, o equivalente a duas retas, formando o triângulo. Desta forma, existem N memórias, onde cada uma possui $2 \times Q$ palavras de 32 bits.

3.2.3 Unidade de fuzzificação

O proposito da unidade de fuzzificação UF consiste em traduzir os valores escalares de entrada, recebidos por alguns sensores, para termos linguísticos ativos e seus respectivos graus de pertinência. Isto é feito usando as informações que definem as funções de pertinência correspondentes aos termos linguísticos, armazenados em MEM FP. Isto é realizado para cada variável de entrada do controlador nebuloso, onde há um bloco Fuzzy associado a ela.

O bloco Fuzzy executa o cálculo necessário para obter a versão nebulosa do valor de entrada escalar, representado por x na Equação 23. A execução do cálculo é habilitada pela comparação entre o valor de entrada e os limites min e max armazenados em MEM FP, ou seja, quando o valor de entrada estiver dentro destes limites, significa que o termo linguístico está ativo e, portanto, o cálculo da multiplicação de x por a e da soma deste resultado com b será executado. Este cálculo resulta no valor do grau de pertinência do termo linguístico ativo. Este passo é repetido $Q \times 2$ vezes para todas as retas de todas as funções de pertinência existentes na variável de entrada que está sendo codificada.

3.2.3.1 Arquitetura

A micro-arquitetura do bloco Fuzzy é mostrada na Figura 24[2]. Este bloco inclui um Comparador que determina em qual região dos termos linguísticos o valor de entrada está, dois conjuntos de *flip-flops* que permitem manter o resultado da comparação, ou seja, seus conteúdos identificam os termos linguísticos ativos/inativos. Note que mais de um

[2]As linhas finas representam sinais de 1 bit, enquanto as mais grossas representam sinais de 32 bits

termo linguístico pode estar ativo para o mesmo valor da variável de entrada. Existem dois conjuntos de *flip-flops*, pois cada termo linguístico é representado por duas retas. O bloco `Fuzzy` também possui um bloco `FPU` que é responsável pelos cálculos necessários de multiplicação e adição. Os resultados obtidos para as duas retas, modelando o termo linguístico, são mantidos em dois registradores de 32-bits distintos. Estes são os valores dos graus de pertinência, que são registrados após o término do cálculo pelo bloco `FPU`. O bloco utiliza dois conjuntos de registradores de 32-bits, nomeados de `TuFP1` e `TuFP2`, em cada conjunto há um registrador para cada reta e um conjunto para cada função de pertinência compondo os termos linguístico, compondo as funções de pertinência associados ás variável de entrada.

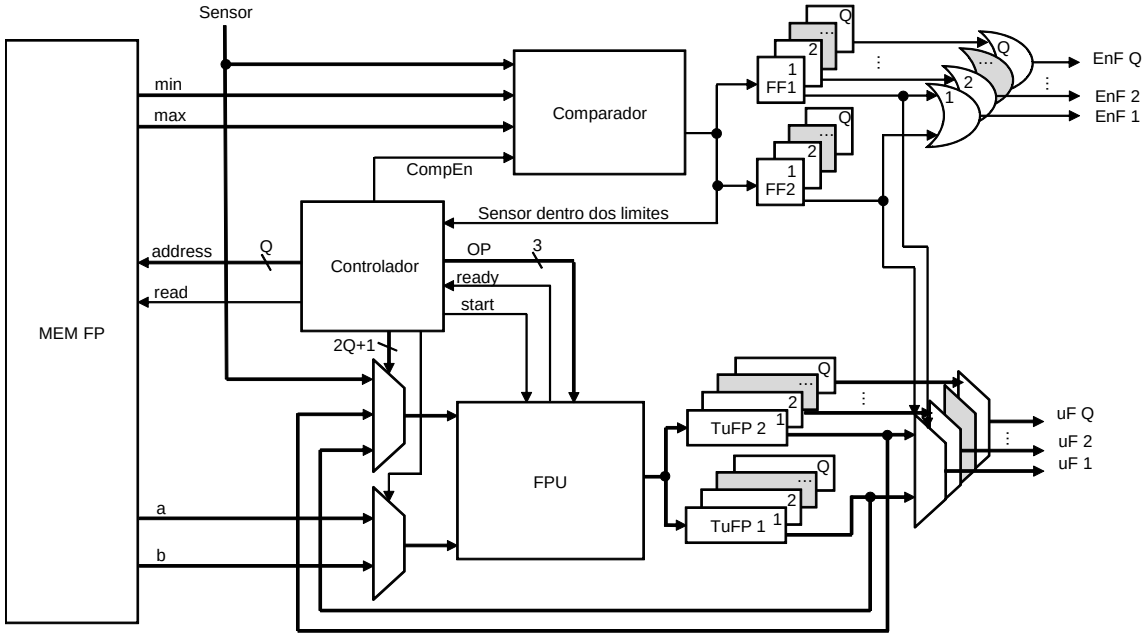


Figura 24: Micro-arquitetura do bloco Fuzzy

As entradas de um bloco `Fuzzy` são: a variável de entrada e as características das funções de pertinência associadas à ela. Estas características são a, b, min e max armazenadas em um segmento de `MEM FP`, como explicado na Seção 3.2.2. As saídas de um bloco `Fuzzy` são: sinal EnF_i, $i = 1 \ldots Q$, ou seja, um bit para cada termo linguístico existente e sinal uF_i, $i = 1 \ldots Q$, ou seja, um valor em ponto flutuante 32-bit representando o grau de pertinência para cada termo linguístico correspondente. Note que os termos linguísticos não ativos possuem o valor $0,0$ como grau de pertinência. Quando o bit EnF_i está ativo, isto indica que o termo linguístico de número i da variável é ativo com grau de pertinência $uF_i \neq 0$.

3.2.3.2 Controle

A Figura 25 ilustra a dinâmica imposta pela máquina de estados finitos que controla a operação do bloco `Fuzzy`. Tão logo a requisição de leitura seja enviada para o segmento `MEM FP`, com endereço da palavra de memória, o conteúdo da mesma contendo as características das funções de pertinência, é disponibilizado para o bloco `Fuzzy`. Os valores de *min* e *max* são usados para testar se a variável de entrada está dentro dos limites da função de pertinência do respectivo termo linguístico. Caso positivo, o cálculo subsequente do grau de pertinência associado é iniciado. Senão, o resultado é $0,0$.

Na sequência, é dada uma ideia geral sobre o funcionamento do bloco `Fuzzy`. Quando este bloco receber o comando *enable* do controlador principal (vide Figura 19), habilitando-o a ser executado, a MEF muda do estado *start* para o estado *fp_rd* e inicia o comando de leitura do `MEM FP`.

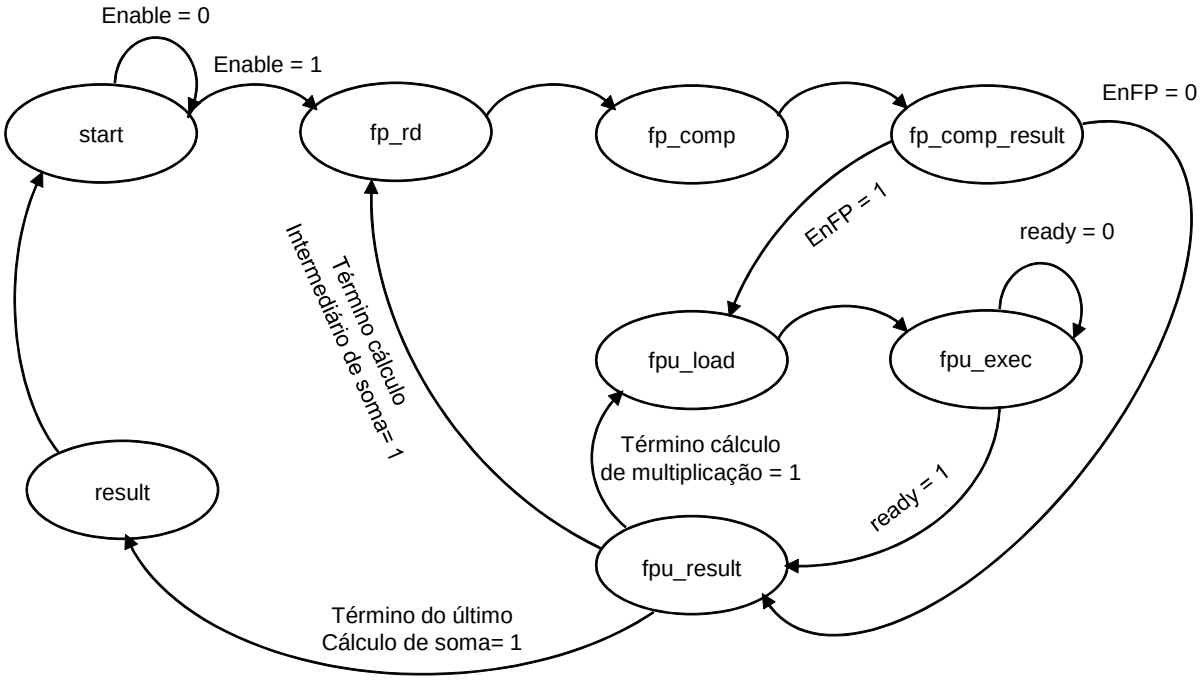


Figura 25: Diagrama de transição da máquina de estados do controlador de fuzzificação

Assim que a palavra de memória é recebida, a MEF entra no estado *fp_comp*, onde, usando os valores de mínimo e máximo, inicia o `Comparador` para executar a comparação necessária do teste, verificando se o valor de entrada está dentro dos limites da reta do termo linguístico. Então, a MEF muda para o estado *fp_comp_result*, onde testa o resultado da comparação. Como todo termo linguístico tem duas retas, cada resultado é armazenado nos *flip-flops* `FF1` e `FF2`, conforme representado na Figura 24. Quando a

comparação é igual a zero, ou seja, o valor de entrada está fora da região entre os limites da função de pertinência, a MEF transita para o estado *fpu_result*. Caso contrário, i.e., esta muda para o estado *fpu_load* para carregar os valores dos cálculos de multiplicação e soma, de acordo com a Equação 23, e logo em seguida, mudando para o estado *fpu_exec*. Então, o controle vai para o estado *fpu_result* para registrar o grau de pertinência obtido e depois retornar ao estado *fpu_load*. A MEF repete este processo até que os cálculos de multiplicação e soma sejam concluídos. Sempre que a MEF entra no estado *fpu_result* e ainda há algum grau de pertinência a ser calculado, esta muda para *fp_rd* para aguardar por uma nova palavra proveniente da memória `MEM FP`. Caso contrário, a última palavra da memória, ou seja, a última reta das funções de pertinência foi processada, então a MEF vai para o estado *result*, gerando o sinal de término $FuzzyOK_j, j = 1 \ldots N$ para o controlador principal, retornando para o estado *start* para aguardar o próximo ciclo. O sinal $FuzzyOK$ gerado pela `UF`, visto na Figura 20, é habilitado quando todos os $FuzzyOK_j, j = 1 \ldots N$ o são.

Algoritmo 2 Cálculo durante a etapa de fuzzificação

Entrada: $sensor, min, max, a$ e b;
Saída: uF_i e $EnF_i,\ i = 1 \ldots 2 \times Q$;

```
Se enable = 1 Então
  Para i ← 1 to 2 × Q Faça
    Address ← i;
    read ← 1;
    Se min < sensor < max Então
      FPUout_i ← sensor × a + b;
      COMPout_i ← 1;
    Senão
      FPUout_i ← 0;
      COMPout_i ← 0;
  read ← 0;
  k ← 1;
  Para i ← 1 to Q Faça
    EnF_i ← (COMPout_k OR COMPout_{k+1});
    Se COMPout_k = 1 Então
      uF_i ← FPUout_k;
    Senão Se COMPout_{k+1} = 1 Então
      uF_i ← FPUout_{k+1};
    Senão
      uF_i ← 0;
    k ← k + 2;
```

Todo sinal de saída EnF_i, $i = 1 \ldots Q$ é gerado por um conjunto de portas OR dos bits registrados pelos *flip-flops* `FF1` e `FF2`. Por outro lado, todo grau de pertinência uF_i, $i = 1 \ldots Q$ é o conteúdo de um dos registradores `TuFP1` e `TuFP2`, dependendo do valor do bit registrado nos *flip-flops* `FF1` e `FF2` correspondentes. Note que o grau de pertinência uF_i será usado apenas no estágio de inferência seguinte, se e somente se $EnF_i = 1$.

O Algoritmo 2 mostra o código equivalente à descrição de *hardware* desenvolvida para o CNP, referente ao bloco `Fuzzy`, onde baseado no valor de entrada a ser fuzzificado e os dados das funções de pertinência, provenientes da memória, a fuzzificação é realizada. Primeiramente, os dados de cada reta são lidos e o valor de entrada é verificado se está dentro dos limites *min* e *max* da reta. Caso positivo, a Equação 23 da reta é usada com o valor de entrada. Este processo é realizado a cada iteração para todas as retas. Em um segundo momento, os resultados são integrados dois a dois, pois cada termo linguístico é formado por duas retas. Ao final deste processo, tem-se as informações de quais termos linguísticos estão ativos com seus respectivos graus de pertinência.

3.2.3.3 Simulação

O projeto proposto para o controlador nebuloso foi modelado usando VHDL (NAVABI, 1998). A simulação funcional do modelo obtido foi feito usando o ModelSim da Mentor Graphics (MODELSIM..., 2012). Muitas configurações de controladores foram simuladas e todas funcionaram corretamente.

Para validação, são mostrados alguns diagramas de tempo da simulação dos principais estágios do CNP: fuzzificação, inferência e defuzzificação. Isto é feito com o mesmo exemplo do pêndulo, detalhado no Capítulo 1. Vale lembrar que, o controlador nebuloso precisa de 2 variáveis de entradas, as quais são ângulo (A) e velocidade angular (V), e apenas uma variável de saída, força (F). Todas as variáveis são modeladas usando cinco termos linguísticos. (Vide Figuras 9, 10 e 11 no Capítulo 1 para detalhes sobre as funções de pertinência destas variáveis.) A base de regras possui 11 de 25 regras possíveis, como mostrado na Tabela 2. O controle do problema mostrado em todos os diagramas de simulação, presentes neste Capítulo, é relativo ao caso, $A = 6$ e $V = -1$. Por uma questão de clareza, este caso é o mesmo detalhado no Capítulo 1.

A Figura 26 mostra o início da simulação do processo de fuzzificação dos valores de entrada, enquanto a Figura 27 mostra o final da simulação deste processo. O teste simulado do bloco de fuzzificação é configurado com 5 termos linguísticos, então após o

bloco de função de pertinência ter sido executado, todas as informações são armazenadas no segmento de `MEM FP`. Assim que o valor de entrada está pronto no sinal $SensorInput$, neste caso com o valor $6,0$, o sinal $EnFuzzy$ habilita o processo de fuzzificação. O sinal $MFAddr$ indica o endereço das caraterísticas para cada reta (a, b, min e max) das funções de pertinência na memória `MEM FP`. Observe que o sinal $inRange$ vai para 1 quando $MFAddr$ é 5 e 6, os quais correspondem à reta direita da função de pertinência $insignificante$ e a reta esquerda de *p-pequeno*, respectivamente, na variável A (vide Figura 12 no Capítulo 1). Vale lembrar que os termos linguísticos, na Figura mencionada, são contados da esquerda para a direita e que o bit menos significativo no sinal EnF está na direita, ou seja, o mesmo é referenciado da direita para a esquerda. O sinal EnF para a variável A é 0000100000 (vide Figura 26) quando a reta número 5 é testada e 0001100000 (vide Figura 27) quando a reta número 6 é testada. Esta quantidade de 10 bits é relativa à quantidade de retas que formam todas as funções de pertinência da variável. Apenas após a operação da porta OR que os sinais são agrupados dois a dois e representados em quantidade de termos linguísticos (5) ativos ou não. Todos os cálculos são feitos usando o bloco `FPU`, o qual recebe suas entradas pelos sinais $fpuInput1$, $fpuInput2$ e o código da operação que precisa ser executada no momento, que é dado pelo sinal fpu_op, sendo 000 para a adição, 001 para a subtração, 010 para a multiplicação e 011 para a divisão. A operação de execução inicia quando o sinal fpu_start vai a 1 e termina quando o sinal fpu_ready é gerado. Observe que o `FPU` executa duas operações (1 multiplicação e 1 adição/subtração) para cada reta selecionada. A saída de `FPU` é mostrada no sinal $fpuOutput$. Usando a visualização em hexadecimal, o primeiro grau de pertinência calculado é `3ECCCCCC` e o segundo é `3F19999A`, os quais correspondem exatamente à $0.39999998 \approx 0,4$ para o termo linguístico $insignificante$ e $0,6$ para *p-pequeno*, respectivamente. O estado atual da MEF de fuzzificação é representado pelo sinal $FuzzyState$, enquanto que o estado global do controlador é representado por $MainState$.

3.2.4 Unidade de inferência

O propósito da unidade de inferência é identificar, para cada uma das variáveis de saída do controlador nebuloso, os termos linguísticos que estão ativos, juntamente com seus respectivos graus de pertinência. Isso é feito usando o resultado da unidade de fuzzificação e o conjunto de regras que estão armazenados em `Regras`. Esta é a unidade mais

Figura 26: Operação simulada do bloco de fuzzificação – Início

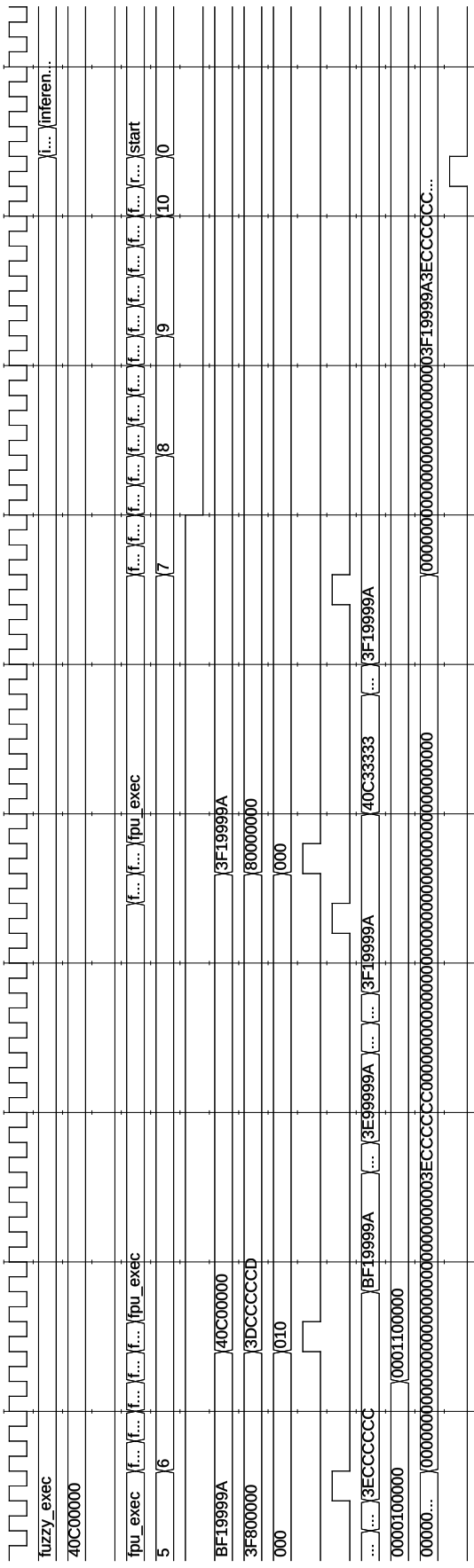


Figura 27: Operação simulada do bloco de fuzzificação – Fim

complexa, tendo em vista, que a arquitetura é escalável, e que a sua estrutura depende dos 4 parâmetros do projeto N, M, Q e P.

3.2.4.1 Arquitetura

Antes de apresentar os detalhes da unidade de inferência, primeiramente, será introduzida a estrutura usada para compor as regras do sistema nebuloso. Como ilustrado no Capítulo 1, uma regra $\mathcal{R}$ é definida por duas partes: a premissa $\mathcal{P}$ e o consequente $\mathcal{C}$, como descrito na Equação 27, onde $\mathcal{I}_i$, $i = 1 \ldots N$ são as variáveis de entrada e $\mathcal{T}_k^{\mathcal{I}_i}$, $k = 1 \ldots Q$ são os termos linguísticos associados a ela, $\mathcal{O}_j$, $j = 1 \ldots M$ são as variáveis de saída e $\mathcal{T}_\ell^{\mathcal{O}_j}$, $\ell = 1 \ldots Q$ são os termos linguísticos associados a ela. Em geral, o número de termos linguísticos é diferente de uma variável para outra. Entretanto, neste projeto, assume-se, sem perda da generalização, que todas as variáveis, ambas de entrada e saída, são modeladas usando o mesmo número de termos linguísticos Q. Além disto, note que uma dada regra pode conter apenas algumas das N variáveis de entrada do controlador nebuloso e, também, pode habilitar apenas algumas das variáveis de saída.

$$
\begin{aligned}
&\mathcal{R}: \quad \mathcal{P} \Rightarrow \mathcal{C} \\
&\mathcal{P}: \quad \left(\mathcal{I}_1 = \mathcal{T}_j^{\mathcal{I}_1}\right) e \left(\mathcal{I}_2 = \mathcal{T}_k^{\mathcal{I}_2}\right) e \ldots e \left(\mathcal{I}_N = \mathcal{T}_\ell^{\mathcal{I}_N}\right), j, k, \ell = 1 \ldots Q \\
&\mathcal{C}: \quad \left(\mathcal{O}_1 = \mathcal{T}_j^{\mathcal{O}_1}\right) e \left(\mathcal{O}_2 = \mathcal{T}_k^{\mathcal{O}_2}\right) e \ldots e \left(\mathcal{O}_M = \mathcal{T}_\ell^{\mathcal{O}_M}\right), j, k, \ell = 1 \ldots Q
\end{aligned}
\tag{27}
$$

A memória `Regras` da base de regras tem um tamanho de palavra que permite armazenar uma regra completa, todas com a mesma estrutura. Esta representa todas as variáveis de entrada e saída. O formato binário de uma regra, como ela é manuseada no projeto, é representada na Figura 28. São Q bits para cada variável de entrada e saída, sendo um bit para cada termo linguístico existente. Portanto, uma regra ocupa um total de $(N + M) \times Q$ bits.

A memória que implementa a base de regras possui P regras e terá, portanto, $P \times (N + M) \times Q$ bits. Uma requisição de leitura para `Regras` com um determinado endereço retornará uma regra inteira. Os bits da premissa são usados, primeiramente, para testar se a regra carregada pode ser disparada, e se for o caso, iniciar o cálculo do grau de pertinência do termo linguístico do consequente da regra para as variáveis de saída.

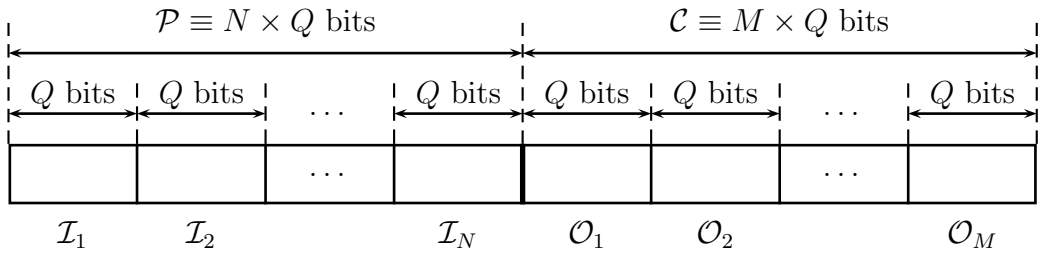


Figura 28: Formato binário das regras de inferência

Uma regra dispara quando todos os sinais $EnF_i, i = 1 \ldots Q$, gerado por `UF`, são iguais à premissa da regra, ou seja, todos os termos linguísticos ativos de cada variável de entrada, também estão ativos na premissa da regra. Além disso, na regra testada, todo termo linguístico de qualquer variável de saída no consequente da regra disparada precisa ser repassado para a unidade de defuzzificação `UD`. Vale lembrar que há no máximo M blocos `Defuzzy`, sendo um para cada variável de saída. Além disso, `UD` precisa também receber o grau de pertinência para cada um destes termos validados.

O grau de pertinência de um termo linguístico de uma variável de saída é inferido de tal forma que no primeiro passo o resultado preliminar é o menor grau de pertinência, considerando todos aqueles associados aos termos linguísticos das variáveis de entrada fuzzificados presentes na premissa da regra disparada. Quando o mesmo termo linguístico da variável de saída está ativo em duas ou mais regras disparadas, resultando em um valor de grau de pertinência para cada uma destas, o maior grau de pertinência é usado. Esta verificação é feita com todas as regras que disparam. Este processo é denominado inferência *max-min* (TANSCHEIT; GOMIDE; TEIXEIRA, 2007), conforme apresentado no Capítulo 1.

A Figura 29[3] mostra a micro-arquitetura do bloco `Inferência`. Suas entradas consistem de N sinais EnF_i, de 1 bit, $i = 1 \ldots Q$ e correspondentemente N grupos de graus de pertinência uF_i, de 32 bits, $i = 1 \ldots Q$, os quais são os resultados de saída de `UF`, como descrito na Seção 3.2.3. Suas saídas são o conjunto de sinais de Q-bits EnD_i, $i = 1 \ldots M$, que identificam os termos linguísticos que foram inferidos e seus respectivos graus de pertinência uD_i, $i = 1 \ldots M$, os quais são sinais de $Q \times 32$ bits.

Na Figura 29, a função dos multiplexadores é "filtrar" os sinais recebidos da fuzzificação, de acordo com a premissa das regras. Portanto, estes recebem como seletor a premissa de cada regra considerada na inferência e como entrada, nos superiores, os sinais

[3]As linhas finas representam sinais de 1 bit, enquanto as mais grossas representam sinais de 32 bits

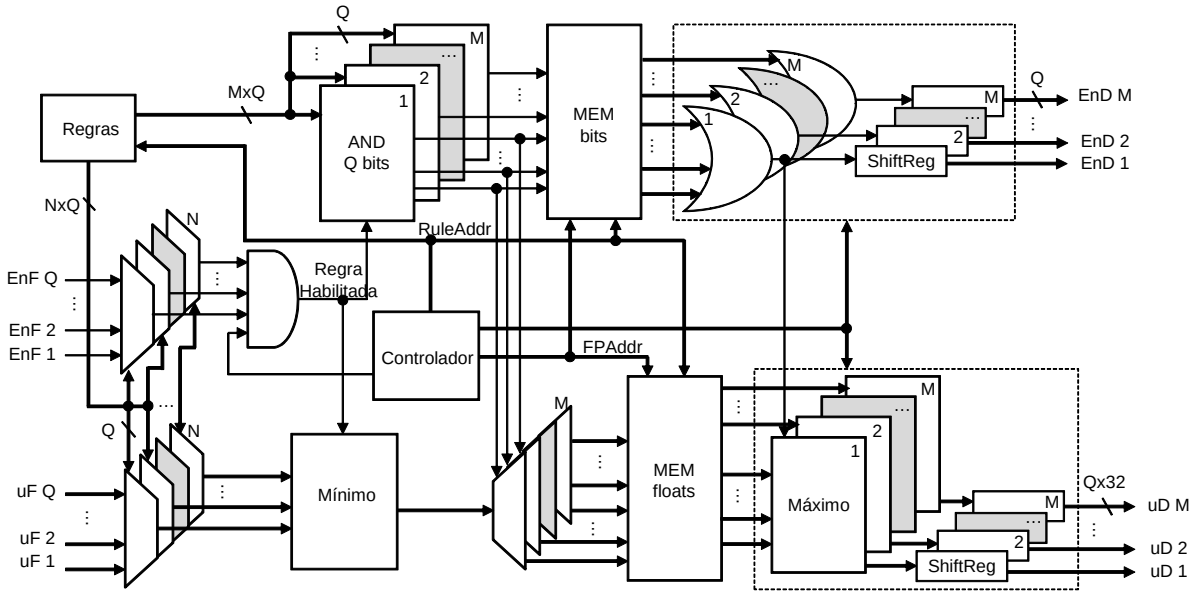
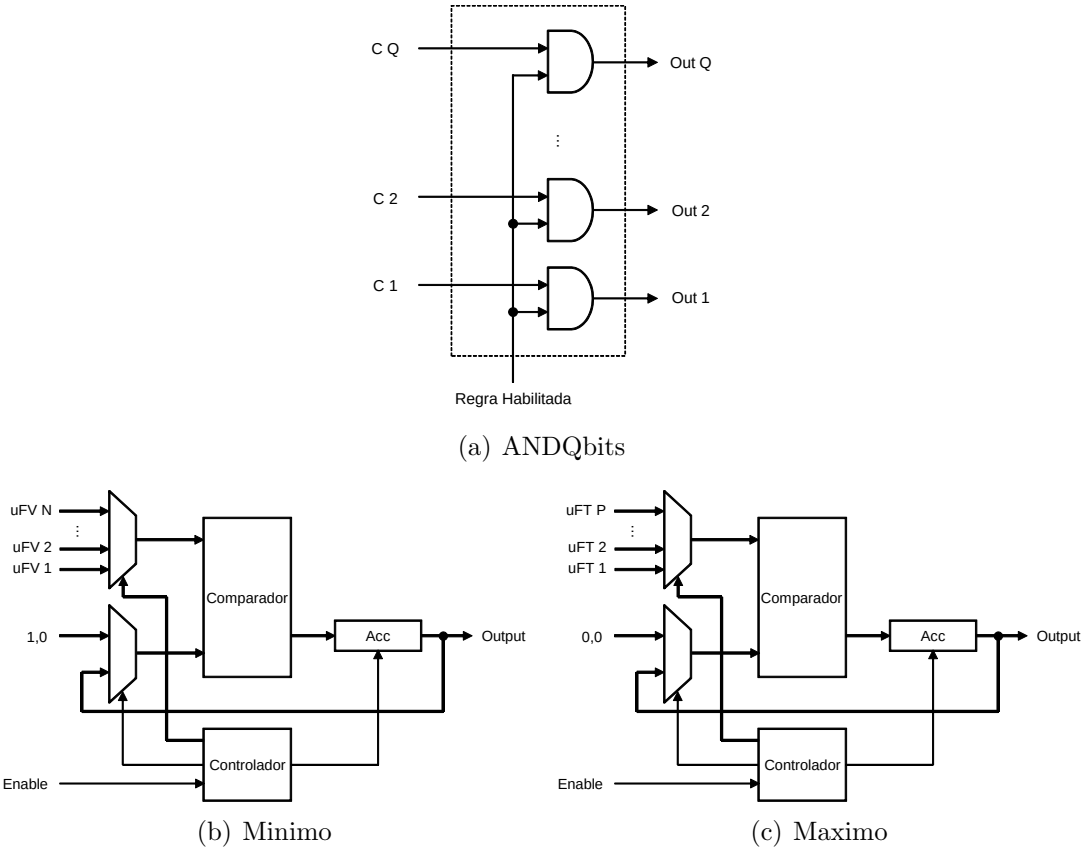


Figura 29: Micro-arquitetura do bloco de inferência

dos termos linguísticos ativos ou não EnF e, nos inferiores, os seus respectivos graus de pertinência uF. Note que há um par para cada variável de entrada em um total de N. A porta AND determina se a regra atual deve ser disparada. O componente `ANDQbits` é um vetor de portas AND, o qual é descrito na Figura 30(a). Este recebe como entrada o consequente de cada regra a ser inferida, representado por $C_i, i = 1 \ldots Q$, e de acordo com o sinal habilitador *RegraHabilitada*, proveniente da porta AND, permite a passagem do consequente da regra para a memória `MEMbits`, que será mostrado mais a frente, nesta seção. Note que há um componente `ANDQbits` para cada variável de saída, em um total de M.

Neste projeto, utiliza-se o processo de inferência max-min. Então, os componentes `Minimo` e `Maximo` retornam o menor valor de N valores em ponto flutuante e o maior valor de P valores em ponto flutuante, respectivamente. Suas estruturas internas são apresentadas nas Figuras 30(b) e 30(c), respectivamente. Note que para o bloco `Minimo`, inicialmente, a primeira entrada é comparada com $1,0$. Dessa forma, na primeira execução o resultado é sempre o primeiro valor de entrada e na sequencia a verificação de mínimo é o resultado anterior e uma nova entrada. Este processo é repetido até que o último valor de entrada seja testado. Um processo semelhante acontece com o componente `Maximo`, mas com a constante inicial de comparação $0,0$.

Voltando à descrição da Figura 29, a função dos demultiplexadores é posicionar o valor do grau de pertinência minimizado na saída $\mathcal{O}_k, k = 1 \ldots M$ e termo linguístico



(a) ANDQbits

(b) Minimo

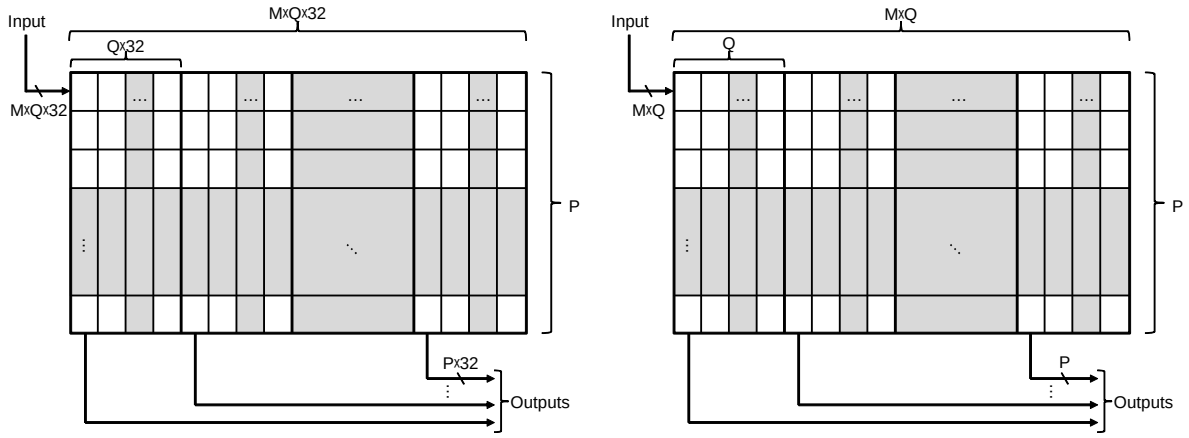
(c) Maximo

Figura 30: Estrutura interna dos componentes auxiliares: seleção do consequente da regra disparada, o mínimo e o máximo dos graus de pertinência

$\mathcal{I}_i, i = 1 \ldots Q$ corretos, de acordo com o consequente da regra inferida. Então, o mesmo recebe como seletor a saída do componente `ANDQbits`, que representa o consequente da regra, e como entrada a saída do bloco `Minimo`. Note que há um demultiplexador para cada variável de saída em um total de M.

O bloco `Inferência` possui paralelismo no tratamento das N entradas fuzzificadas e, também, na atribuição dos sinais inferidos para as M saídas que deverão ser defuzzificadas. Apenas as P regras são tratadas de forma sequencial. Por este motivo, a quantidade de regras, configuradas no CNP, influencia no tempo de execução deste bloco.

A `Inferência` possui dois blocos de memória: a memória do grau de pertinência `MEMfloats` e a memória de bits `MEMbits`. Além disso, este bloco utiliza a base de regras representada pela memória `Regras`. As respectivas estruturas são mostradas nas Figuras 31(a) e 31(b). Uma requisição de escrita em `MEMbits` ou `MEMfloats` grava uma linha, ou seja, $M \times Q$ ou $M \times Q \times 32$ bits, respectivamente, enquanto, uma requisição de leitura retorna $P \times M$ e $P \times M \times 32$ bits. Dessa forma, a `MEMbits` possui um total de $P \times M \times Q$ bits, já a `MEMfloats` possui um total de $P \times M \times Q \times 32$ bits.



(a) MEMfloats (b) MEMbits

Figura 31: Estrutura interna das memórias auxiliares para termos linguísticos inferidos e seus respectivos graus de pertinência

Se r é uma regra que deve ser disparada, então, as informações armazenadas na linha r de `MEMbits` consistem do consequente desta regra disparada. Caso contrário, todos os bits da linha r são resetados. Da mesma forma, as informações armazenadas na linha r de `MEMfloats` consistem do menor valor obtido dos graus de pertinência para aquela regra em todas as colunas, que representam cada termo linguístico de cada variável de saída. Após todas as regras serem consideradas no processo de cálculo do mínimo, os dois blocos de memória `MEMbits` e `MEMfloats` são lidos coluna a coluna, ou seja, termo linguístico a termo linguístico, para todas as variáveis de saída, realizando o processo de cálculo do máximo, no caso de `MEMfloats` e a lógica OR no caso de `MEMbits`. Esta última conterá a informação de qual termo linguístico estará ativo na variável de saída, para todas as regras inferidas, onde cada uma representa uma linha. De forma similar, a memória `MEMfloats` conterá a informação do valor do grau de pertinência de todos os termos linguísticos na variável de saída, para todas as regras inferidas. Os quais serão processados pelo bloco `Maximo`, coluna a coluna.

3.2.4.2 Controle

Como é possível ver na Figura 32, a máquina de estados também foi otimizada para ter o menor número de estados possíveis. Isto foi feito tendo em mente a reconfiguração do número de termos linguísticos, regras e variáveis de entrada e saída. Basicamente, existem duas repetições no controle imposto pela MEF. A primeira permite a leitura de regras, uma após a outra, e a identificação do grau de pertinência mínimo associado.

Os resultados de cada iteração são armazenados na respectiva linha de `MEMfloats`. Em seguida, a segunda repetição usa o conteúdo de `MEMfloats`, coluna após coluna, para identificar o maior grau de pertinência para cada termo linguístico, para o caso em que mais de uma regra gerou um grau de pertinência diferente de zero para um mesmo termo linguístico.

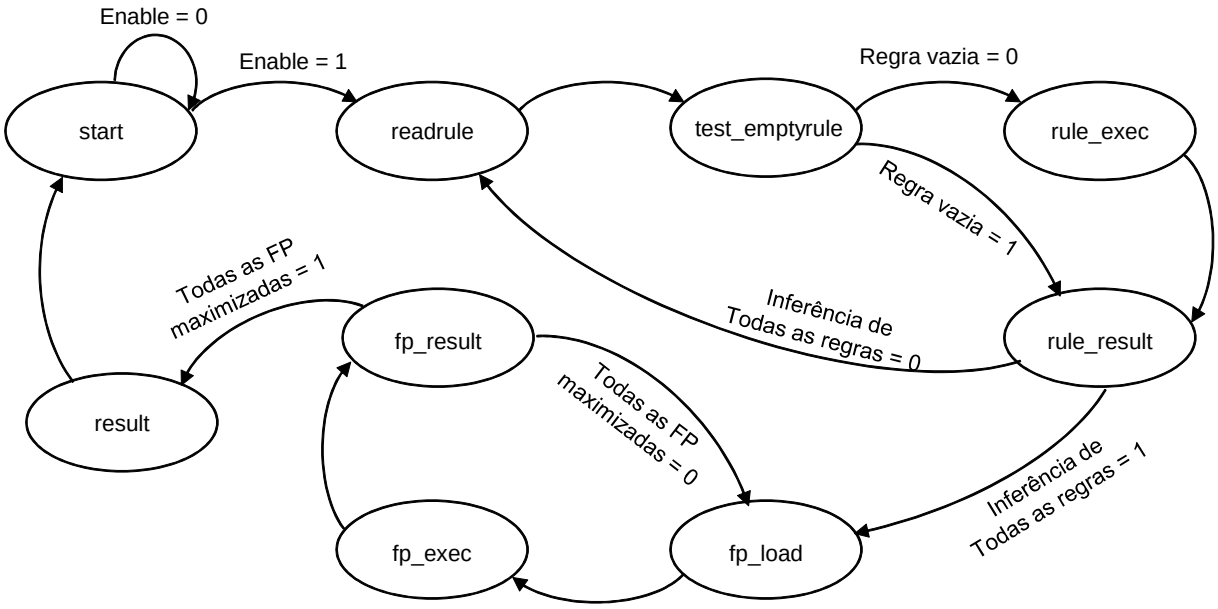


Figura 32: O diagrama de transição da máquina de estados do controlador de inferência

A seguir, é esboçado o controle da operação do bloco de inferência. Quando o bloco `Inferência` recebe o comando *enable* do controlador principal habilitando-o a ser executado, a máquina de estados muda do estado *start* para o estado *readrule*, onde uma regra específica a ser testada é selecionada e lida da memória `Regras`. Então, o controle passa para o estado *test_emptyrule*, onde a regra carregada é testada se é válida. Caso positivo, o MEF vai para o estado *rule_result*. Caso contrário, entra no estado *rule_exec*. Como mostrado na Figura 29, a informação da premissa da regra é enviada para controlar a operação dos dois conjuntos de multiplexadores. Note que existem dois multiplexadores para cada variável de entrada: um para os sinais binários de termos linguísticos ativos e outro para os valores de graus de pertinência. No estado *rule_exec*, para cada regra disparada, o `ANDQbits` e `Minimo` operam simultaneamente e o resultado obtido é armazenado em `MEMbits` e `MEMfloats`, respectivamente. Na sequência, antes que a escrita ocorra, o resultado passa por um conjunto de M demultiplexadores para associar o menor grau de pertinência selecionado para cada um dos termos linguísticos do consequente da regra testada. O processo descrito até então, é iterado para todas

as regras existentes na base. Se a regra testada não for a última, o estado *readrule* é visitado novamente e o processo é repetido. Caso contrário, i.e., a última regra foi processada, a MEF vai para o estado *fp_load*, onde a segunda repetição é iniciada. Neste estado, os graus de pertinência do mesmo termo linguístico de todas as regras são lidos de `MEMfloats` para prover a entrada para o componente `Maximo`, o qual opera tão logo a MEF entre no estado *fp_exec*. Este processo é iterado Q vezes, o que permite o processamento de todo o conteúdo de `MEMfloats`. Após isto, a MEF entra no estado *fp_result* para deslocar o resultado no *shift register* mostrado na Figura 29. Se ainda existir M colunas em `MEMfloats` para tratar, a MEF retorna para o estado *fp_load*. Caso contrário, entra no estado *result*, gerando o sinal de término $InfOK$ para o controlador principal e, em seguida, retornando para o estado *start*, para esperar pelo próximo ciclo.

O Algoritmo 3 mostra o código equivalente à descrição de *hardware* desenvolvida para o CNP, referente ao bloco de inferência. Baseado nos valores e sinais recebidos dos blocos de fuzzificação, este bloco realiza a inferência das regras. A cada iteração uma regra é lida e sua premissa verificada em relação aos dados recebidos da fuzzificação. Caso os termos linguísticos ativos sejam idênticos aos da premissa da regra, então a regra é disparada, de tal forma que entre os graus de pertinência de cada variável fuzzificada dos determinados termos linguísticos da premissa dessa regra, é selecionado o menor valor como sendo o grau de pertinência da regra inferida. O consequente da regra assim inferida determina com qual termo linguístico este grau de pertinência estará associado. Então, as informações de termos linguísticos ativos e seus graus de pertinência inferidos de cada regra são armazenados em suas respectivas memórias `MEMbits` e `MEMfloats`. Após o término do processo de tratamento de todas as regras, é necessário verificar se mais de uma regra gerou um valor de grau de pertinência diferente de zero para um mesmo termo linguístico. Desta forma, as memórias são lidas, de termo linguístico em termo linguístico, considerando todas as regras e o maior valor é selecionado como resultado final para o termo linguístico em consideração. Este resultado será usado no bloco de defuzzificação. Como este bloco é escalável, todos os processos descritos podem ser executados quantas vezes forem necessárias, de acordo com a quantidade de fuzzificadores (N), defuzzificadores (M), regras (P) e termos linguísticos (Q).

Algoritmo 3 Cálculo da inferência

Entrada: $uF_i^j,\ EnF_i^j,\ i = 1 \ldots Q,\ j = 1 \ldots N$ e $Regras_r^l,\ r = 1 \ldots P,\ l = 1 \ldots (N + M) \times Q$;
Saída: uD_i^k e $EnD_i^k,\ k = 1 \ldots M,\ i = 1 \ldots Q$;

```
Se enable = 1 Então
  Para r ← 1 to P Faça
    R ← Regras_r;
    Se R válida Então
      Para j ← 1 to N Faça
        I ← R^j;
        AndInput_j ← I AND EnF^j;
        MinInput_j ← uF^j;
      RuleFired ← AND(AndInput);
      Se RuleFired Então
        Min ← MIN(MinInput);
        Para k ← 1 to M Faça
          O ← R^(N+k);
          Para i ← 1 to Q Faça
            Se O_i = 1 Então
              MEMfloats_r^(k×i) ← Min;
              MEMbits_r^k ← O;
            Senão
              MEMfloats_r^(k×i) ← 0;
              MEMbits_r^k ← 0;
    Senão
      MEMfloats_r ← 0;
      MEMbits_r ← 0;
  Para k ← 1 to M Faça
    Para i ← 1 to Q Faça
      uD_i^k ← MAX(MEMfloats^(k×i));
      EnD_i^k ← OR(MEMbits^(k×i));
```

3.2.4.3 Simulação

A Figura 33 mostra o início da simulação do processo de inferência, já a Figura 34 mostra o meio do processo e, finalmente, a Figura 35 mostra o final da inferência. Os diagramas de tempo do processo de fuzzificação referentes à simulação da variável de entrada V foram omitidas na Seção 3.2.3, o qual foi executado simultaneamente com o da variável A, que foi descrita anteriormente no processo de fuzzificação. Para a variável V as retas 3 e 4 estão ativas e seus correspondentes graus de pertinência são `3F4CCCCD` e `3E4CCCCD`, os quais correspondem exatamente a $0,8$ e $0,2$, respectivamente. O sinal EnF para a variável V deve ser 00110. É possível verificar isso na Figura 33, o sinal EnF na entrada do bloco `Inferência` é composto pelos sinais de termos linguísticos ativos ou não das

duas variáveis de entrada A e V, onde os cinco bits menos significativos representam os sinais da entrada A. O mesmo vale para o sinal uF, mas de 32 em 32 bits para os valores dos graus de pertinência. São utilizadas 11 das 25 possíveis regras. Este bloco recebe suas entradas de 2 blocos de fuzzificação e entrega seus resultados para 1 bloco de defuzzificação.

Observe que o sinal *rules* mostra a configuração da premissa e consequente das regras que serão usadas no processo de inferência, onde os cinco bits menos significativos representam o consequente e os outros representam a premissa. A variável A está nos bits menos significativos da premissa. O sinal $EnInf$ habilita o processo de inferência. O estado atual da MEF de inferência é representado pelo sinal $InfState$, enquanto que o estado global do controlador nebuloso é, como antes, representado por $MainState$. O sinal $RuleAddr$ representa o número da regra que está sendo testada. Observando o sinal $RuleFired$, note que as regras 1, 2, 5 e 6 foram disparadas. O valor `3F800000` é o valor 1.0, usado como uma das entradas do bloco `Minimo` na primeira iteração. Os graus de pertinência obtidos por este bloco são `3E4CCCCD`, `3E4CCCCD`, `3ECCCCCC` e `3F19999A` (ou $0,2$, $0,2$, $0,4$ e $0,6$), como mostrado no sinal $MinOutput$ nas Figuras 33 e 34. Infelizmente, não é possível ver todos os dígitos hexadecimais no sinal, mas baseado nos valores de entrada do bloco `Minimo` pelo sinal $MinInput$ é possível ver qual dos valores foi selecionado. Estes graus de pertinência são obtidos pelas regras 1, 2, 5 e 6 (vide sinal $RuleFired$ e $RuleAddr$), respectivamente. Após receber o menor valor de grau de pertinência para cada regra disparada, é necessário calcular o maior grau de pertinência associado a cada termo linguístico. O processo é habilitado pelo sinal $EnMaxExec$ (vide Figura 35). Da mesma forma que a comparação do bloco `Minimo`, não é possível ver todos os dígitos hexadecimais no sinal $MaxOutput$, o mesmo ocorre no sinal $MaxInput$, que possui 11 valores de 32 bits, sendo, um para cada regra. Entretanto, no endereço 2 do termo linguístico, dado pelo sinal $FPAddr$, ou seja, o termo *zero* da variável de saída *força* (F), dois valores são recebidos pelo sinal $MaxInput$ (`3ECCCCCC` e `3E4CCCCD`) e na comparação pelo bloco `Maximo`, o resultado de saída é `3ECCCCCC`. Os outros dois graus de pertinência (`3E4CCCCD` e `3F19999A`) foram inferidos para termos linguísticos diferentes, ou seja, apenas uma regra associou o valor do grau de pertinência `3E4CCCCD` ao termo linguístico *n-pequena* e apenas uma regra associou o valor `3F19999A` ao termo *p-pequena* da variável de saída. O resultado final do processo de inferência é dado pelos sinais EnD e uD. O primeiro representando

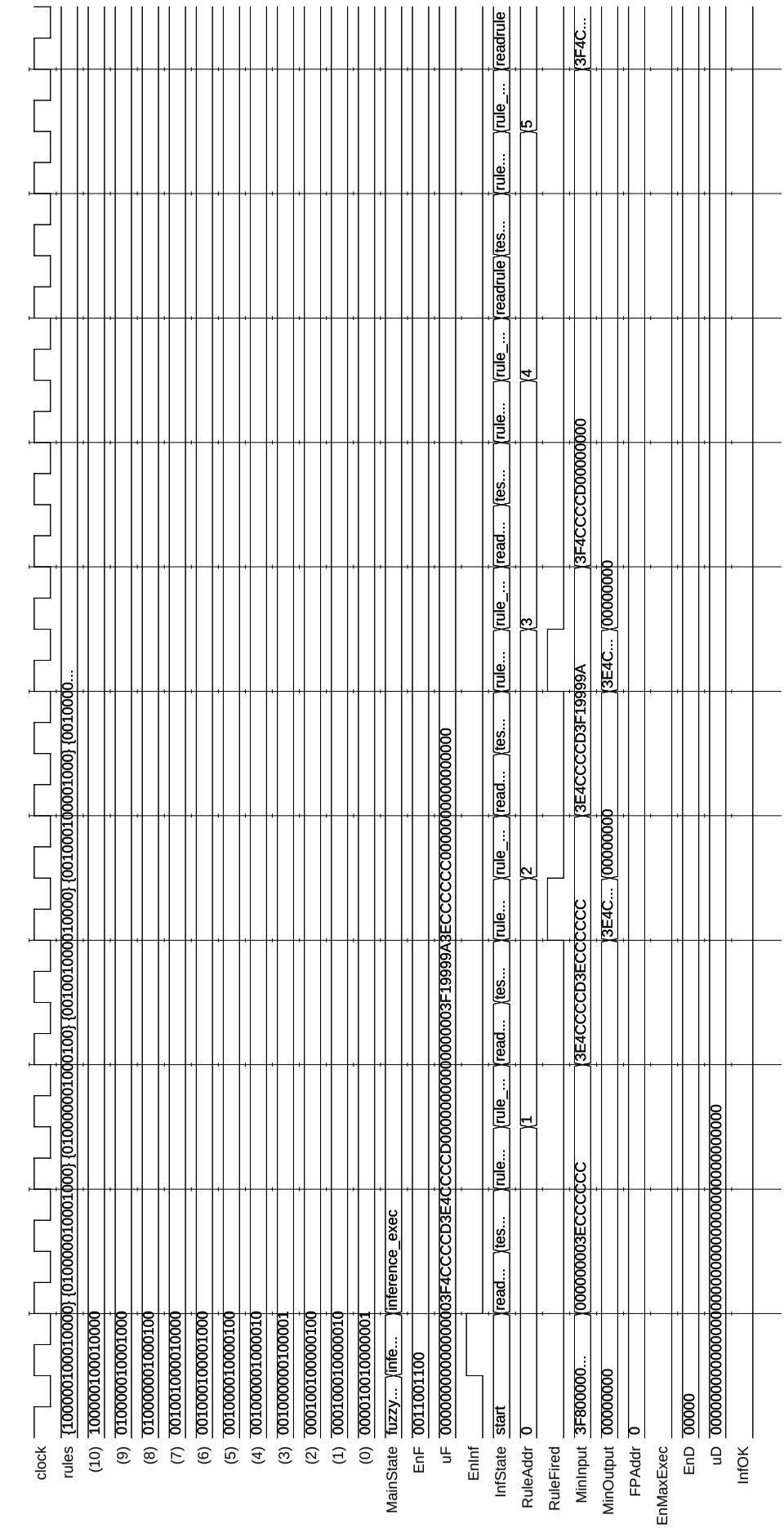


Figura 33: Operação simulada do bloco de inferência – Início

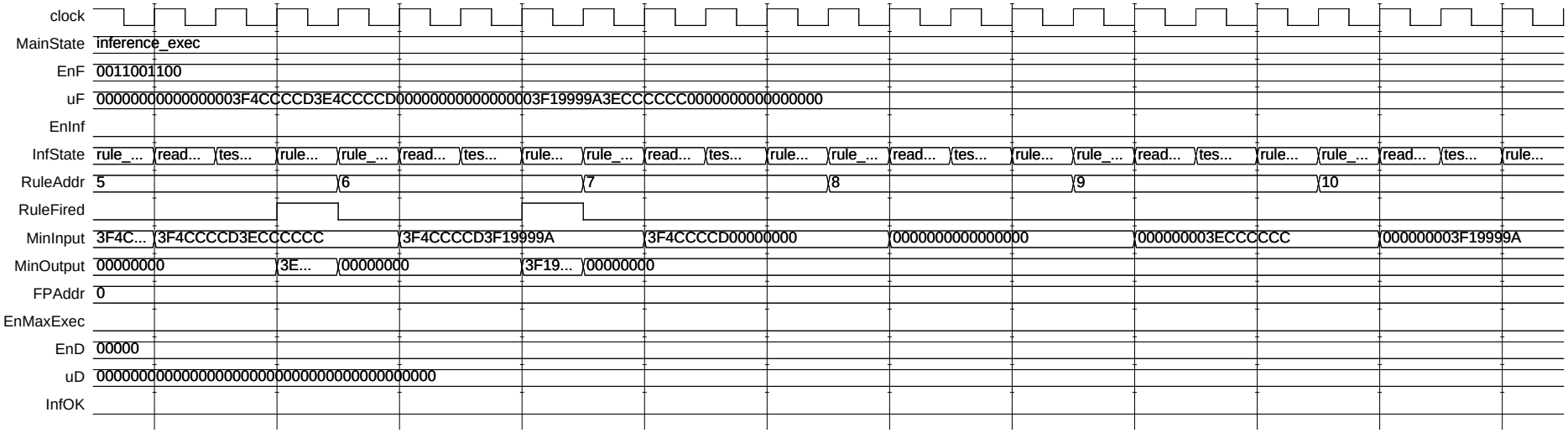


Figura 34: Operação simulada do bloco de inferência – Meio

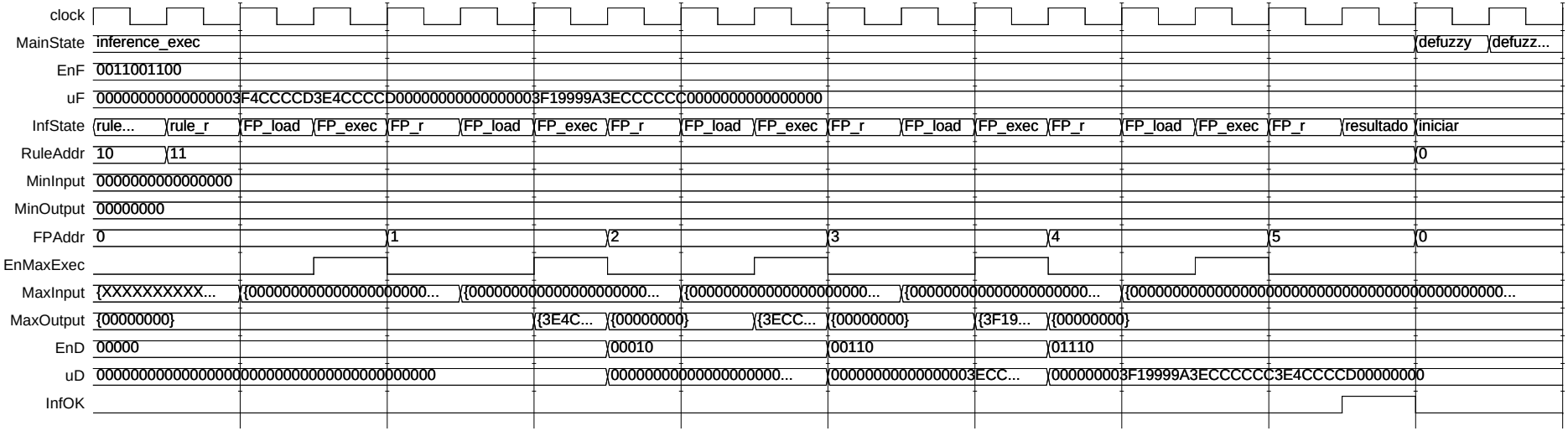


Figura 35: Operação simulada do bloco de inferência – Fim

os termos linguísticos ativos e o segundo representando os graus de pertinência associados aos termos linguísticos, onde 3 termos foram inferidos com graus de pertinência diferentes de zero (`3ECCCCCC`, `3E4CCCCD` e `3F19999A`), como mostrado no sinal uD da Figura 35.

3.2.5 Unidade de defuzzificação

O propósito da unidade de defuzzificação é calcular o valor escalar associado às variáveis linguísticas de saída do CNP, dados os termos linguísticos e seus correspondentes graus de pertinência, como identificado e calculado pela unidade de inferência. O centroide, conforme definido na Equação 28, é usado para executar o processo de defuzzificação no CNP. Vale lembrar que uD_i, $i = 1 \ldots Q$ são os graus de pertinência dos termos linguísticos associados com a variável de saída $\mathcal{O}$. Este método calcula o centro geométrico das funções de pertinência de saída, considerando os termos linguísticos ativos, recebidos do bloco de inferência, junto com seus respectivos graus de pertinência. O cálculo é feito de acordo com os passos do Algoritmo 4.

$$\mathcal{O} = \frac{\sum_{i=0}^{Q} uD_i \times PM_i}{\sum_{i=0}^{Q} uD_i} \tag{28}$$

3.2.5.1 Arquitetura

A Figura 36[4] mostra a micro-arquitetura do bloco `Defuzzy`. Suas entradas consistem dos sinais EnD_i, de 1 bit, $i = 1 \ldots Q$ e correspondentemente de seus graus de pertinência uD_i, de 32 bits, $i = 1 \ldots Q$, os quais são os resultados de saída do bloco `Inferência`, como descrito na Seção 3.2.4. Sua saída é o valor escalar de um dos resultados finais do CNP. Este bloco possui, basicamente, um bloco `FPU` para realizar os cálculos em ponto flutuante necessários.

3.2.5.2 Controle

A Figura 37 mostra o diagrama de estados da MEF que controla a operação do bloco `Defuzzy`. A seguir, são esboçados os passos principais para este controle. Quando este bloco receber o comando *enable* do controlador principal habilitando-o a ser executado, a MEF sai do estado *start* para o estado *test_empty*, onde testa a possibilidade de não ter

[4] As linhas finas representam sinais de 1 bit, enquanto as mais grossas representam sinais de 32 bits

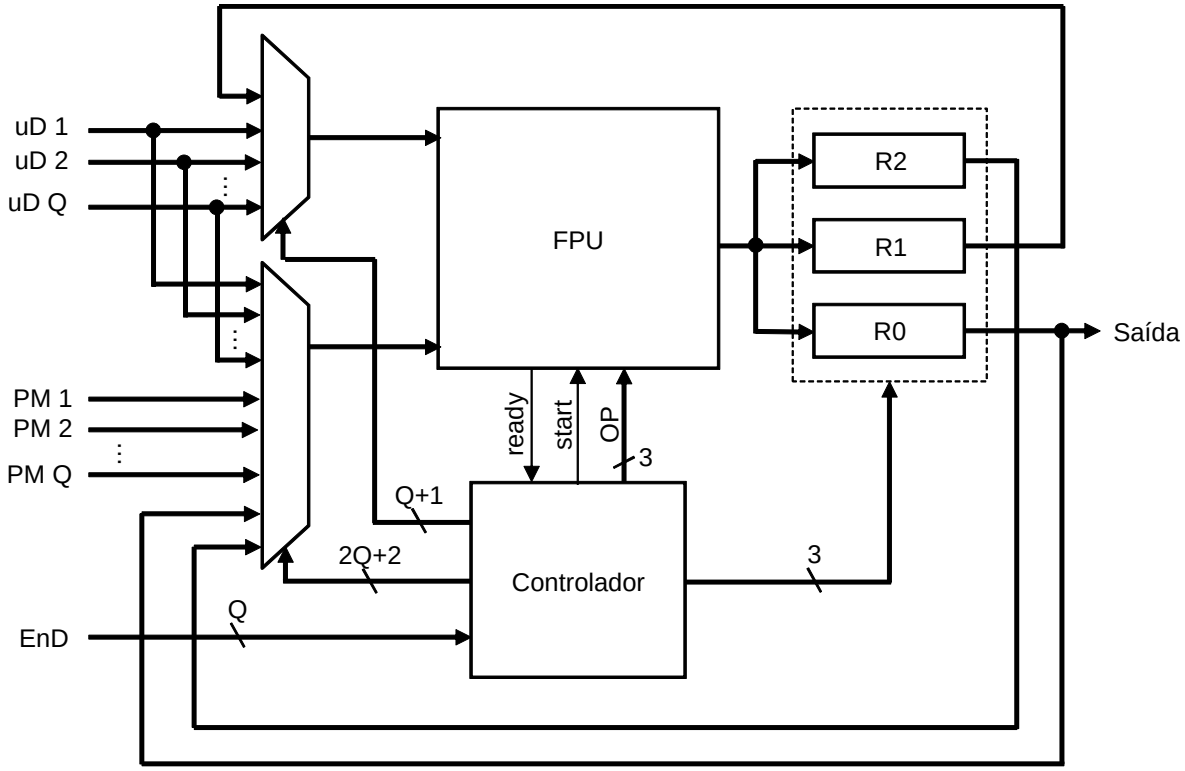


Figura 36: Micro-arquitetura do bloco de defuzzificação

nenhum termo linguístico ativo. Caso positivo, a MEF vai para o estado *result*, o qual permite retornar $0,0$ como resultado de saída do `Defuzzy`. Caso contrário, i.e., se pelo menos um termo linguístico para a variável de saída que está sendo processada estiver ativo, a MEF seguirá para *fpu_result* ou *fpu_load*. Então se o atual termo não estiver ativo, a MEF vai imediatamente para o estado *fpu_result*, por que não há nada a ser calculado. Entretanto, se o termo estiver ativo, a MEF vai para *fpu_load*, onde o controle permite que os valores do cálculo específico sejam carregados e, em seguida, vai para o estado *fpu_exec*, onde o cálculo descrito no Algoritmo 4 é executado. Assim que a execução de uma iteração é completada, a MEF muda para o estado *fpu_result*, onde testa se todos os EnD_i e os respectivos uD_i, $i = 1 \ldots Q$ foram considerados. Caso negativo, a MEF retorna para o estado *test_empty* e o mesmo processo é repetido. Caso contrário, o resultado é imediatamente gravado e disponibilizado no registrador R_1 (vide Figura 36) e então, a MEF entra no estado *result*, gerando o valor da saída do CNP e um sinal de término $DefuzzyOK_k, k = 1 \ldots M$, e o bloco se torna pronto novamente para operar a partir do início. O sinal $DefuzzyOK$ gerado pela `UD`, visto na Figura 20, é habilitado quando todos os $DefuzzyOK_k, k = 1 \ldots M$ o são.

O Algoritmo 4 mostra o código equivalente à descrição de *hardware* desenvolvida

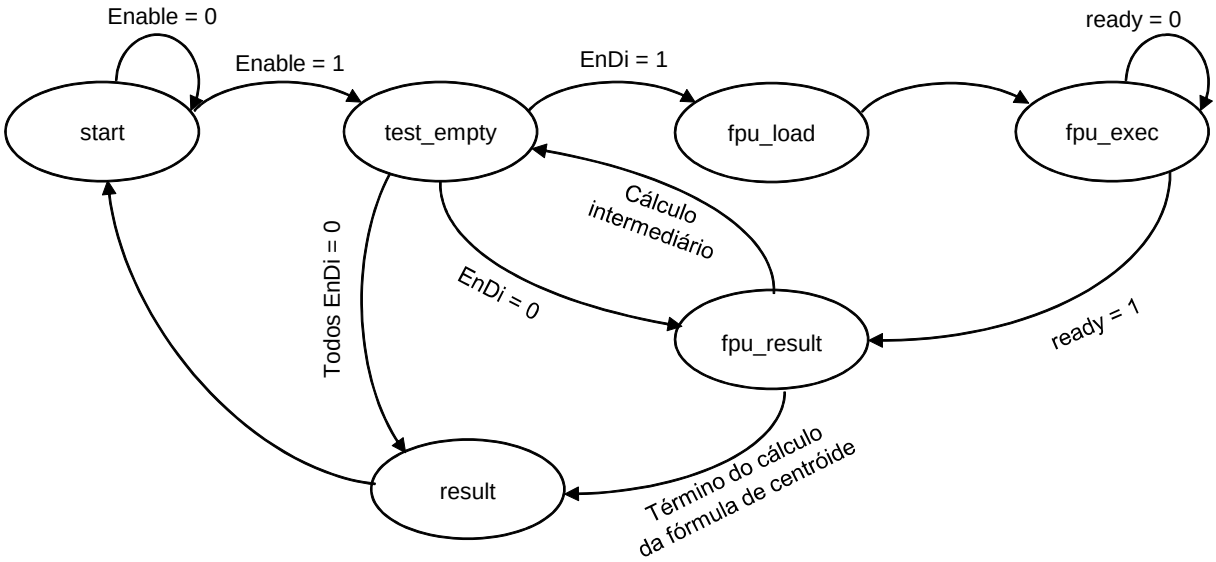


Figura 37: Diagrama de transição da máquina de estados que controla o processo de defuzzificação

Algoritmo 4 Cálculo do centroide de saída $\mathcal{O}$

Entrada: EnD_i e $uD_i,\ i = 1 \ldots Q$ para a variável de saída $\mathcal{O}$;
  $R_1 \leftarrow 0$;
  $R_2 \leftarrow 0$;
  Se $EnD \neq 00 \ldots 0$ **Então**
    Para $i \leftarrow 1$ *to* Q **Faça**
      Se $EnD_i = 1$ **Então**
        $R_0 \leftarrow uD_i \times PM_i$;
        $R_1 \leftarrow R_1 + R_0$;
        $R_2 \leftarrow R_2 + uD_i$;
    $R_0 \leftarrow R_1/R_2$;
  Retorna R_1;

para o CNP, referente ao bloco de defuzzificação, onde baseado nos valores e sinais recebidos do bloco de inferência, o cálculo do centroide é realizado. Primeiramente, é calculado o numerador e denominador considerando cada termo linguístico ativo e na sequência, é realizada a divisão entre eles para determinar o valor escalar da defuzzificação do controlador proposto.

3.2.5.3 Simulação

A Figura 38 mostra o início do processo de defuzzificação, configurado para trabalhar com 5 termos linguísticos, enquanto a Figura 39 mostra o final dessa simulação. O sinal $EnDefuzzy$ habilita o processo. Então, é iniciado o cálculo do valor escalar da variável de saída, baseado na identificação dos termos linguísticos testados e seus respectivos graus de

pertinência. Isto é feito usando o bloco `FPU`, o qual, como no processo de fuzzificação, recebe seus valores de entrada nos sinais $fpuInput1$ e $fpuInput2$ e o operador a ser efetuado no sinal fpu_op. O início de execução da operação pode ser identificado pelo pulso no sinal fpu_start e o término ocorre quando um pulso aparece no sinal fpu_ready. Por exemplo, na Figura 38, a primeira operação é a multiplicação do grau de pertinência `3E4CCCCD` $= 0,2$ por PM `BF800000` $= -1$, como mostrado nos sinais $fpuInput1$ e $fpuInput2$, respectivamente. Após o término da operação o resultado é `BE4CCCCD` $= -0,2$, como mostrado no sinal $fpuOutput$. O estado atual da MEF é representado pelo sinal $DefuzzyState$, enquanto que o estado global do controlador nebuloso é, como anteriormente, representado por $MainState$. Para este exemplo, as variáveis PMs são `40000000` $= 2,0$, `3F800000` $= 1,0$, `00000000` $= 0,0$, `BF800000` $= -1,0$ e `C0000000` $= -2,0$, como pode-se ver no sinal $MaxPoints$. A soma ponderada deverá então ser $1,0 \times 0,6 + (-1,0) \times 0,2 = 0,4$ e a soma de todos os graus de pertinência $0,2 + 0,4 + 0,6 = 1,2$. Na Figura 39, observe que a saída final do controlador é o resultado da divisão de `3ECCCCCE` $= 0,40000004$ por `3F99999A` $= 1,2$, ou seja, `3EAAAAAB` $= 0,33333334$, o qual é o valor escalar da variável de saída *força* (F) retornada pelo CNP em resposta aos valores de entrada $A = 6$ e $V = -1$. A diferença infinitesimal ($6,67 \times 10^{-9}$) e, portanto, inofensiva no dígito final é devido à representação de precisão simples de ponto flutuante.

3.3 Considerações Finais do Capítulo

Neste capítulo foi apresentada a macro-arquitetura do CNP, assim como a micro-arquitetura das unidades funcionais, detalhando seus principais sinais, máquinas de estados e funcionamento. Além de mostrar sua simulação utilizando o exemplo do pêndulo invertido, demonstrando os sinais de controle de cada passo no controlador e a apresentação do resultado correto baseado nos valores de entrada de ângulo e velocidade. O capítulo seguinte apresenta os resultados quanto a utilização do controlador nebuloso proposto em algumas aplicações.

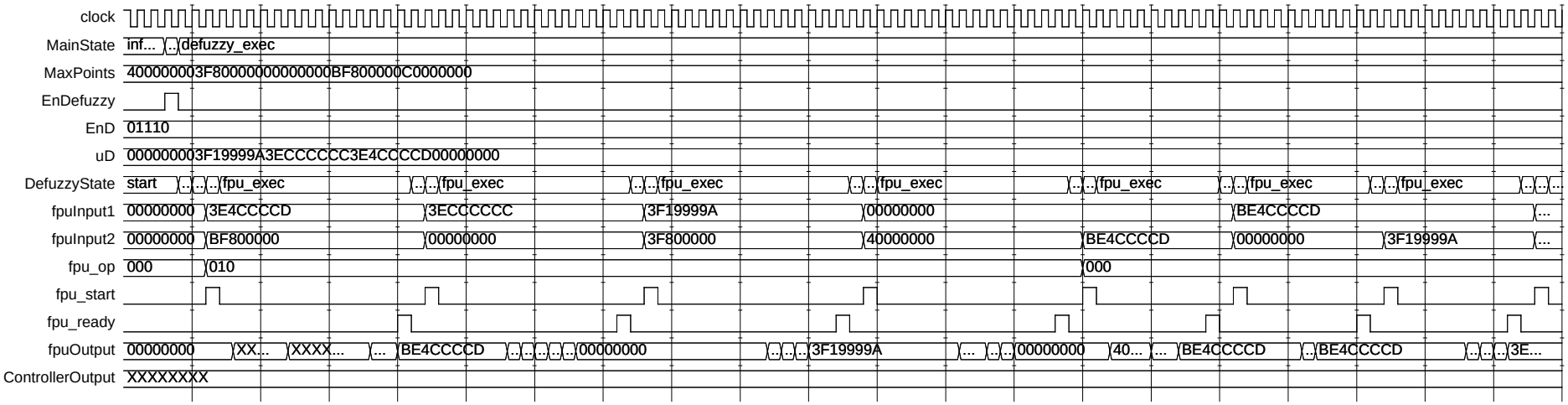


Figura 38: Operação simulada do bloco de defuzzificação – Início

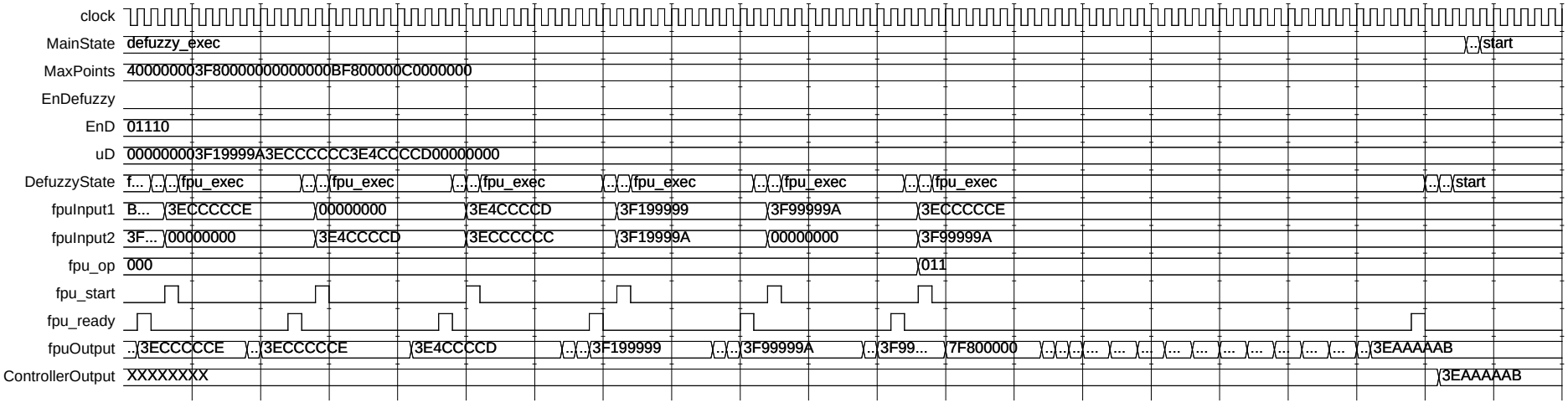


Figura 39: Operação simulada do bloco de defuzzificação – Fim

Capítulo 4

RESULTADOS EXPERIMENTAIS

NESTE capítulo são apresentados os resultados de implementação com informações de área utilizada, frequência máxima de operação e os resultados numéricos de saída do controlador nebuloso proposto (CNP), juntamente com o erro ocasionado. Todo o *hardware*, descrito neste trabalho, foi sintetizado com o *software Xilinx Platform Studio* e implementado em FPGA com a placa Virtex 5 XC5VLX110T (XILINX, 2012).

Os testes apresentados foram realizados em FPGA com cinco aplicações distintas para verificar o correto funcionamento e a generalização do *hardware* do CNP, são eles: o pêndulo invertido já apresentado no Capítulo 1; uma aplicação para controle de velocidade de exaustores em túneis urbanos (GOMIDE; GUDWIN; TANSCHEIT, 1995), de acordo com o nível de poluição; uma aplicação em robótica para controlar a direção e velocidade de um robô autônomo (LI; CHANG; CHEN, 2003), que deverá seguir um segundo robô; uma aplicação para controle de ciclo de trabalho de um semáforo (CHANGLIANG; HONGHAI, 2010), de acordo com a densidade de fluxo de carros; e por último, mas não menos importante, uma aplicação para navegação de robôs (LIN; HUANG; CHUANG, 2005).

4.1 Controle do Pêndulo Invertido

De acordo com a aplicação apresentada no Capítulo 1, o pêndulo invertido possui 2 variáveis de entrada (*ângulo* e *velocidade*), 11 regras, 5 termos linguísticos e 1 variável de saída (*força*) (SANDRES; NEDJAH; MOURELLE, 2013b) (SANDRES; NEDJAH; MOURELLE, 2013a). Com esta configuração pode-se ver na Figura 41, a quantidade necessária do número de ciclos para a execução de cada bloco do controlador. Apenas o bloco `Defuzzy` possui valores diferentes de número de ciclos, dependendo da quantidade de termos linguísticos que estão ativos, estes sendo resultados do bloco `Inference`. Nesta aplicação apenas quatro

regras, no máximo, podem ser disparadas ao mesmo tempo, dependendo do valor dos sensores de entrada dos blocos `Fuzzy` e, de acordo com essas regras disparadas, no máximo, 3 termos linguísticos estarão ativos para o cálculo da defuzzificação. Mantendo isso em mente, Defuzzy0 significa que nenhum termo linguístico está ativo, Defuzzy1 significa que apenas 1 termo linguístico está ativo, e assim por diante, conforme descrito na Tabela 3.

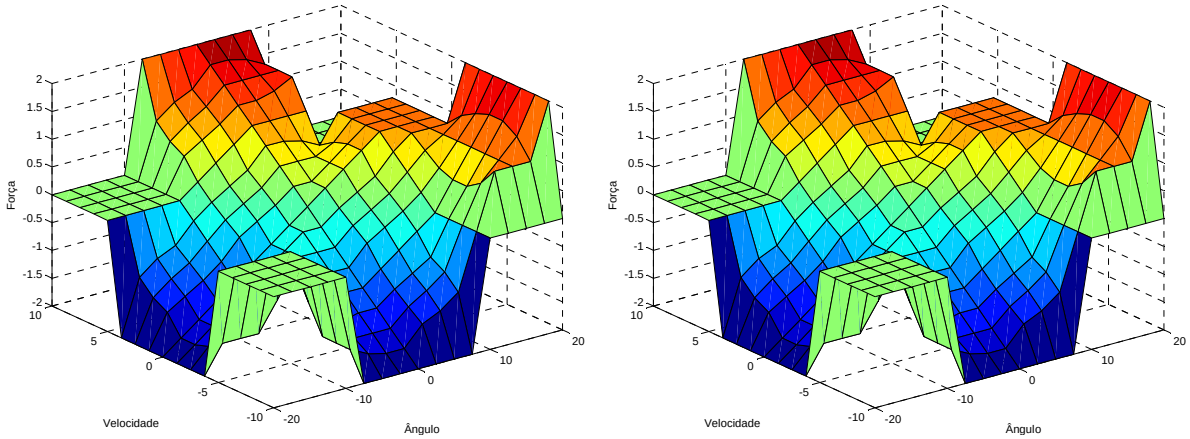
Alguns testes foram realizados para verificar o correto funcionamento do CNP, a Tabela 3 mostra os valores testados na entrada dos sensores, onde pode-se ver os valores dos sensores de cada entrada, são eles *Velocidade* e *Ângulo*, além das regras disparadas. Também é possível ver a quantidade de termos linguísticos ativos na entrada da defuzzificação, o número de ciclos de clock, o tempo de execução em microsegundos, baseado na frequência máxima permitida de 103,928 MHz e o valor escalar do resultado da defuzzificação ou da saída do CNP.

Tabela 3: Resultados do CNP para a aplicação do pêndulo invertido

Ângulo	Velocidade	Regras Disparadas	Qtd de Defuzzy	Ciclos de Clock	Tempo (μseg)	Força
-11	-6	–	0	703	6,76	$0,0000$
$+1$	-11	r_0	1	895	8,61	$-2,0000$
-5	-1	r_1, r_4 e r_5	2	942	9,06	$-0,5000$
$+6$	-1	r_1, r_2, r_5 e r_6	3	989	9,52	$+0,3333$
$+6$	-6	r_0, r_1 e r_2	3	989	9,52	$-0,6667$
-3	$+7$	r_8, r_9 e r_{10}	3	989	9,52	$+1,0000$
-3	$+1$	r_4, r_5, r_8 e r_9	3	989	9,52	$-0,0833$

Os valores utilizados para teste, conforme a Tabela 3, foram selecionados, de tal forma, que algumas premissas sejam satisfeitas, como: quase todas as regras sejam disparadas; haja uma quantidade diferente de regras disparadas; ocorra ou não a maximização no final do cálculo da inferência; e nenhum, um, dois ou três termos linguísticos estejam ativos no resultado da inferência para o cálculo da defuzzificação.

A Figura 40, mostra a superfície de controle baseada na configuração do controlador nebuloso para esta aplicação, onde pode-se ver no plano inferior os eixos representando as variáveis de entrada do controlador e a altura representando a saída do controlador. Neste gráfico, também é possível inferir de forma visual o valor de saída do controlador, baseado nos valores de entrada. O mesmo é montado de acordo com as funções de pertinência e as regras configuradas para cada aplicação.



(a) MATLAB (b) CNP

Figura 40: Superfície de controle para a aplicação do pêndulo invertido

A Figura 40(a) foi obtida pelo software MATLAB (MATHWORKS,), utilizando o *Fuzzy Logic Toolbox* (FIS), no qual a aplicação foi configurada e testada para comparar os valores obtidos com o *hardware* do CNP. Já a Figura 40(b) mostra o gráfico da superfície de controle gerada pelos valores obtidos pelo CNP. Para fins de verificação é calculado o erro quadrático, de acordo com a Equação 29, onde xh_i é o resultado do *hardware* do CNP e xm_i é o resultado da toolbox FIS do MATLAB, para ambas as variáveis, i é o índice de um determinado resultado, de acordo com seus valores de entrada no controlador. Desta forma n é o número de pontos calculados para a determinação da superfície. Foram usados 17 pontos para cada variável de entrada, o que resulta em 289 pontos ou resultados obtidos para a determinação da superfície de controle. Aplicando a Equação 29, tem-se o resultado de $Erro = 2,7682 \times 10^{-10}$, o que demonstra uma excelente precisão do CNP em relação ao MATLAB. Vale lembrar que em todos os valores de resultados obtidos com o *hardware* do CNP foram consideradas apenas quatro casas decimais, caso a comparação fosse feita com toda a precisão disponível, o erro seria ainda menor.

$$Erro = \frac{\sum_{i=1}^{n}(xh_i - xm_i)^2}{n} \tag{29}$$

Usando uma frequência de clock de 100 MHz na FPGA, todo o controlador é executado, no pior dos casos, com Defuzzy3, em 2.051 ciclos de clock ou 20,51 microsegundos. Este é um dos possíveis valores de frequência de clock que pode ser selecionada na placa Virtex 5, mas os resultados da síntese mostram, que a frequência de clock máxima aceita pelo projeto desenvolvido para esta aplicação é de 103,928 MHz, o que resultaria em 19,73

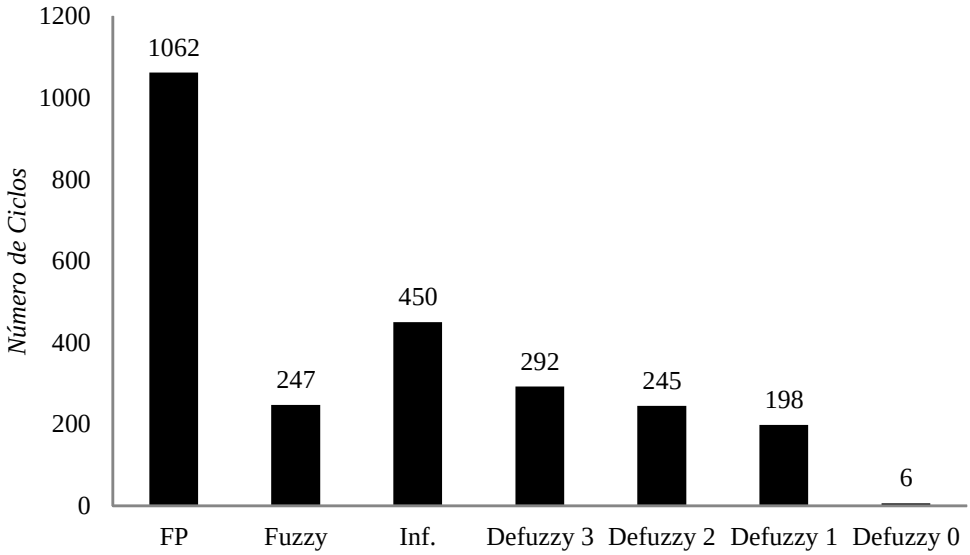


Figura 41: Número de ciclos de clock necessários para a execução dos blocos na FPGA

microsegundos. Vale lembrar que o bloco `FP` será executado apenas uma vez para o cálculo das características das funções de pertinência. Logo, o mesmo não é contabilizado no ciclo normal do controlador em uma malha de controle, que será de no máximo 989 ciclos de clock ou 9,52 microsegundos, considerando o valor máximo permitido de frequência de clock. A Figura 41 mostra o número de ciclos de clock para cada bloco do CNP, inclusive as variações nos números de ciclos para o bloco `Defuzzy`. Já a Figura 42 mostra os tempos de execução de cada bloco, utilizando a frequência máxima de clock.

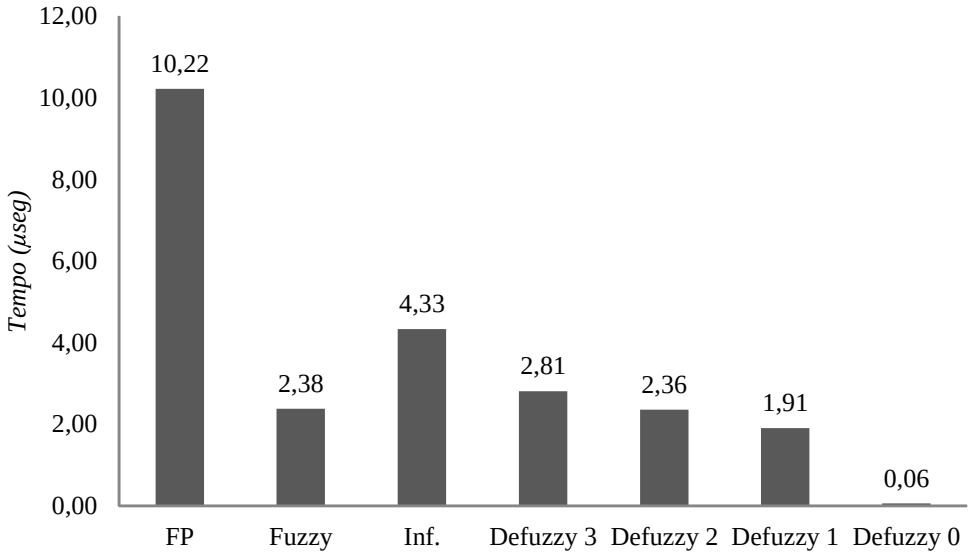


Figura 42: Tempos de execução dos blocos na FPGA em microsegundos

O bloco `Inference` não varia o número de ciclos na execução, da mesma forma

que o bloco `Defuzzy`, pois todas as regras são inferidas no processo de minimização, além de no processo de maximização todos os termos linguísticos serem maximizados, independentemente de haver ou não valores diferentes de zero.

A Figura 43 mostra a área de *hardware* usada na FPGA para programar todo o controlador nebuloso. Já que a placa Virtex 5 utilizada possui 17.280 *Slices* e cada um tem 4 LUTs e 4 *flip-flops*, então o total é de 69.120 LUTs, dos quais são usados 51,5%.

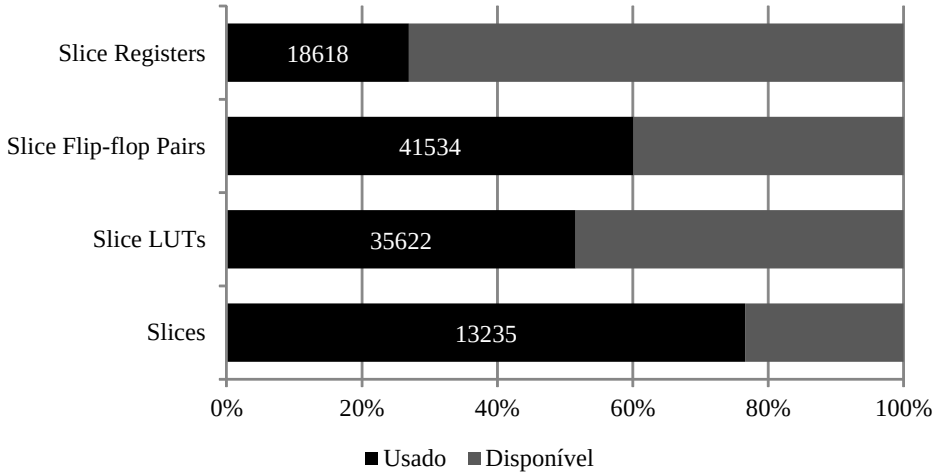


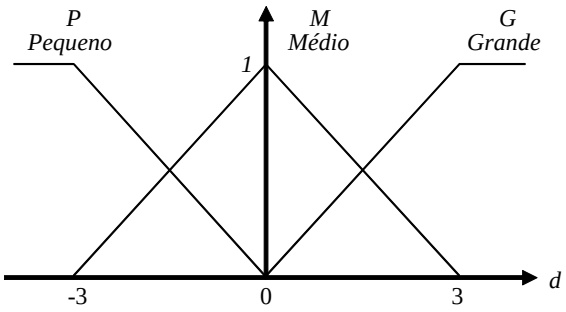
Figura 43: Utilização de área de *hardware*

4.2 Controle de Poluição em Túneis

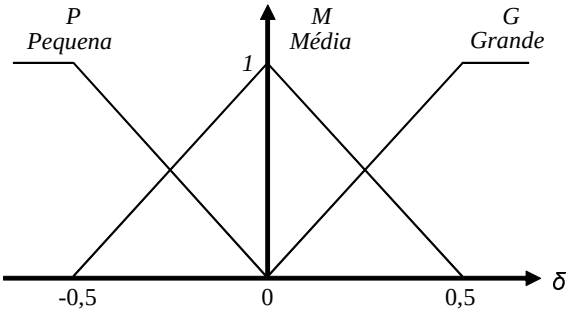
De acordo com a modelagem apresentado em (GOMIDE; GUDWIN; TANSCHEIT, 1995), o controle de poluição em túneis urbanos possui 2 variáveis de entrada, que modelam o erro na percentagem de CO_2 e a variação do erro na percentagem de CO_2, 9 regras descritas na Tabela 4, 3 termos linguísticos para cada variável e 1 variável de saída que representa o incremento na velocidade de exaustão. A Figura 44 mostra as funções de pertinência utilizadas para cada variável de entrada e saída. O Controlador deverá aumentar ou reduzir a velocidade do exaustor, baseado no erro da percentagem de CO_2 e a sua variação.

Com a configuração descrita acima, foram adotadas as mesmas considerações e modelo de análise do controle do pêndulo invertido (Seção 4.1).

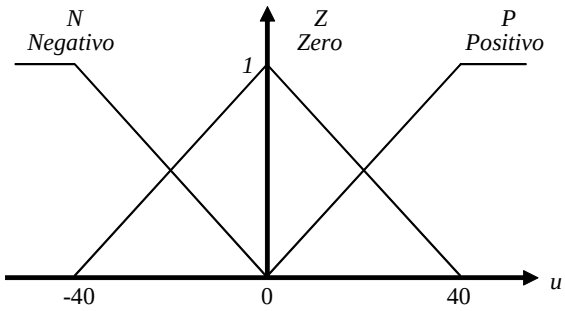
A Tabela 5 mostra os valores testados na entrada dos sensores, onde pode-se ver além destes valores, as regras disparadas, de acordo com a Tabela 4. Também é possível ver a quantidade de termos linguísticos ativos na entrada da defuzzificação, o número de



(a) Variável de entrada erro na percentagem de CO_2



(b) Variável de entrada variação do erro na percentagem de CO_2



(c) Variável de saída incremento na velocidade de exaustão

Figura 44: Gráficos das Funções de Pertinência

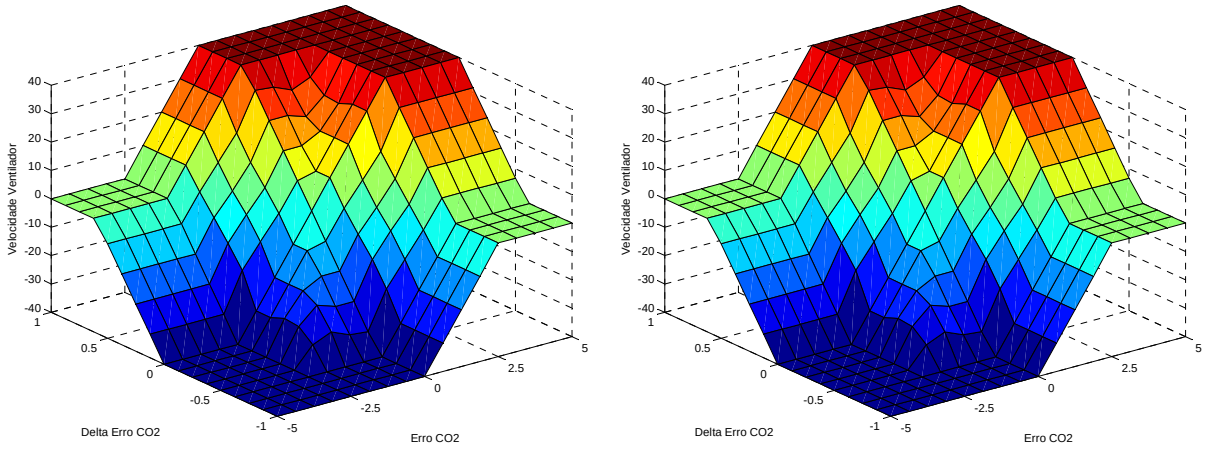
Tabela 4: Regras nebulosas para a aplicação do controle de poluição

regras		ΔErro % CO2		
		Pequena	*Média*	*Grande*
Erro % CO2	*Pequeno*	r_0: *Negativo*	r_1: *Negativo*	r_2: *Zero*
	Médio	r_3: *Negativo*	r_4: *Zero*	r_5: *Positivo*
	Grande	r_6: *Zero*	r_7: *Positivo*	r_8: *Positivo*

Tabela 5: Resultados do CNP para a aplicação do controle de poluição

Erro% CO2	Δ Erro% CO2	Regras Disparadas	Qtd de Defuzzy	Ciclos de Clock	Tempo (μseg)	Velocidade Ventilador
$+0,8$	$+0,8$	r_5 e r_8	1	653	5,86	$+40,0000$
$-4,0$	$-0,4$	r_0 e r_1	1	653	5,86	$-40,0000$
$+2,0$	$+0,4$	r_4, r_5, r_7 e r_8	2	700	6,28	$+30,7692$
$-2,0$	$-0,4$	r_0, r_1, r_3 e r_4	2	700	6,28	$-30,7692$
$+2,0$	$-0,4$	r_3, r_4, r_6 e r_7	3	747	6,70	$-4,4444$
$-2,0$	$+0,4$	r_4, r_5, r_8 e r_9	3	747	6,70	$+4,4444$

ciclos de clock, o tempo de execução em microsegundos, baseado no clock de 111,508 MHz e o valor escalar do resultado da defuzzificação que é a saída do CNP.



(a) MATLAB (b) CNP

Figura 45: Superfície de controle para a aplicação do controle de poluição

A Figura 45, mostra a superfície de controle baseada na configuração do controlador nebuloso para esta aplicação. O cálculo do erro quadrático com a Equação 29, tem-se o resultado de $Erro = 2.5489 \times 10^{-7}$, o que demonstra uma excelente precisão do CNP em relação ao MATLAB.

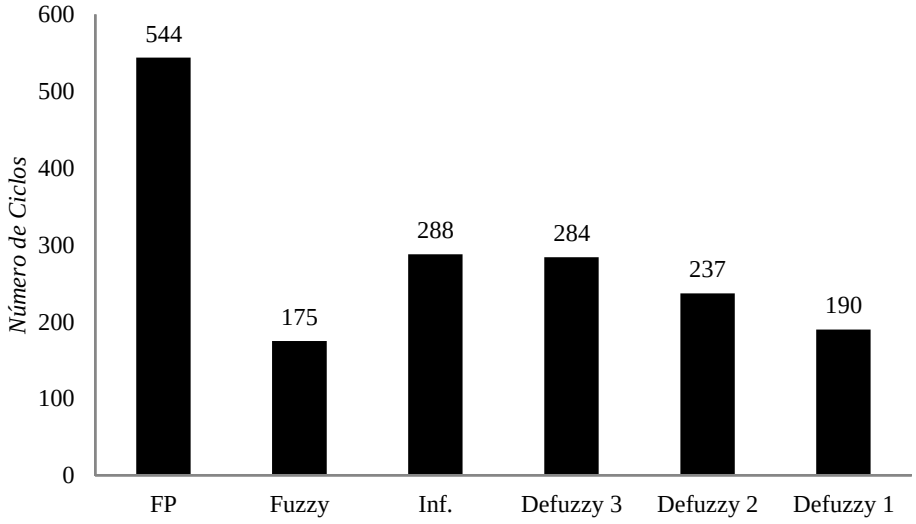


Figura 46: Número de ciclos de clock necessários para a execução dos blocos na FPGA

Usando a frequência de clock de 100 MHz na FPGA, todo o controlador é executado, no pior dos casos, com Defuzzy3, em 1.291 ciclos de clock ou 12,91 microsegundos. Os resultados da síntese mostram, que a frequência de clock máxima aceita pelo projeto desenvolvido para esta aplicação é de 111,508 MHz, o que resultaria em 11,58 microsegundos. Como o bloco `FP` não é contabilizado no ciclo normal do controlador em uma malha de controle, a mesma será de no máximo 747 ciclos de clock ou 6,70 microsegun-

dos, considerando o valor máximo permitido de frequência de clock. A Figura 46 mostra o número de ciclos de clock para cada bloco do CNP, inclusive as variações nos números de ciclos para o bloco `Defuzzy`. Já a Figura 47 mostra os tempos de execução de cada bloco, utilizando a frequência máxima de clock.

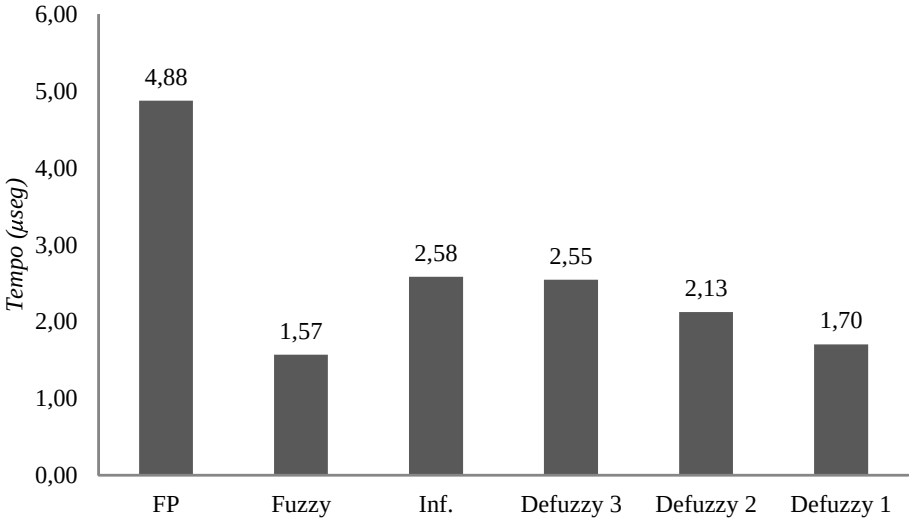


Figura 47: Tempos de execução dos blocos na FPGA em microsegundos

A Figura 48 mostra a área usada na FPGA para implementar todo o controlador nebuloso. Dos 69.120 LUTs da placa Virtex 5 utilizada, são usados 48,2%.

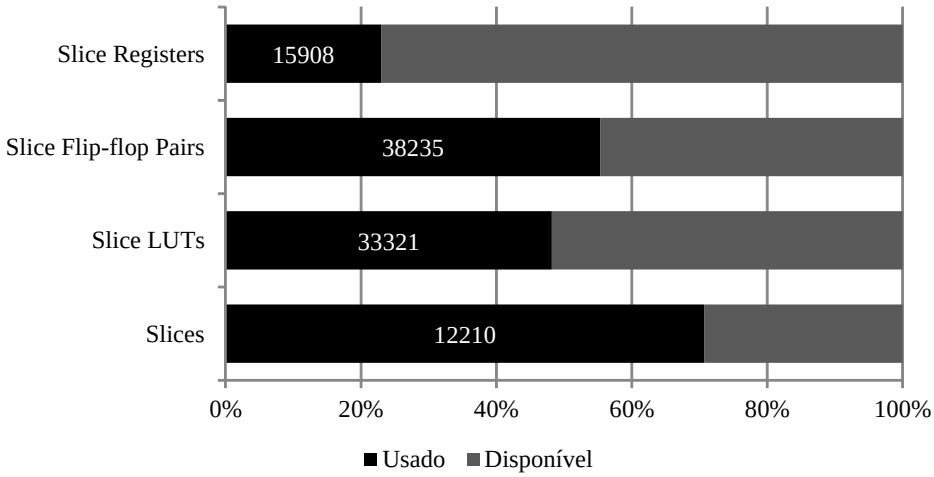
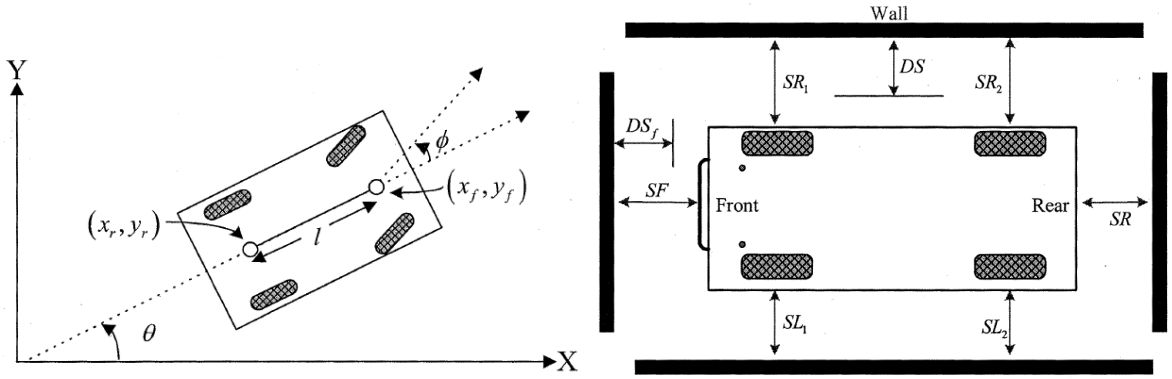


Figura 48: Utilização de área de *hardware*

4.3 Controle de Direção em Robôs

De acordo com a aplicação do trabalho desenvolvido por Tzuu-Hseng (LI; CHANG; CHEN, 2003), o controle de direção para robôs, que se movimentam como carros simulando habilidades humanas de direção, possui dois controles nebulosos encadeados, um para o controle do ângulo das rodas e o segundo para o controle de velocidade do robô.



(a) Modelo cinético do robô.

(b) Definições de distâncias entre o robô e as paredes

Figura 49: Modelagem do robô para aquisição de variáveis para os controles (LI; CHANG; CHEN, 2003)

As Figuras 49(a) e 49(b) mostram as informações utilizadas para o cálculo das variáveis de entrada dos controladores nesta aplicação. O primeiro para o controle da angulação das rodas (ou direção do carro) possui as variáveis de entrada de acordo com as Equações 30 e 31, onde a primeira (x_d) verifica o posicionamento do robô entre as paredes laterais e a segunda (x_e) verifica o quão paralelo o robô está à parede direita.

$$x_d = SR_1 - SL_1 \tag{30}$$

$$x_e = SR_1 - SR_2 \tag{31}$$

A segunda parte do controle gera a saída de velocidade para o robô no qual uma das variáveis de entrada é o resultado do primeiro controlador, ou seja, o ângulo das rodas. A segunda variável de entrada é x_f dada pela Equação 32, onde DS_f é a menor distância possível de aproximação da parede frontal, dessa forma, x_f é a distância entre o robô e a parede frontal com uma distância de segurança que permite evitar colisões.

$$x_f = SF - DS_f \tag{32}$$

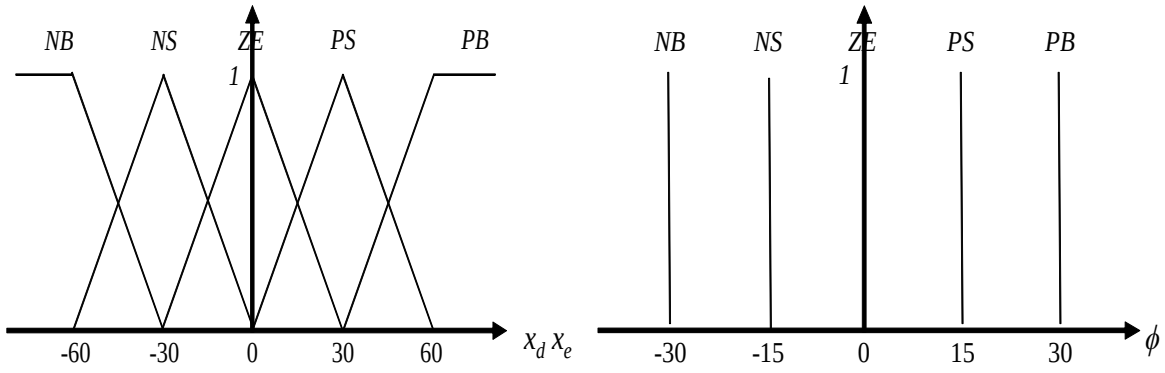
4.3.1 Controle do ângulo das rodas

Como mencionado anteriormente, este controle possui 2 variáveis de entrada, que modelam a centralização do robô entre as paredes laterais e o paralelismo à parede direita, 25 regras descritas na Tabela 6, 5 termos linguísticos e 1 variável de saída que representa o ângulo das rodas frontais do robô.

Tabela 6: Regras nebulosas para a aplicação do controle do ângulo das rodas

regras		X_d				
		NB	*NS*	*ZE*	*PS*	*PB*
X_e	*NB*	r_0: *NB*	r_1: *NB*	r_2: *NS*	r_3: *NS*	r_4: *ZE*
	NS	r_5: *NB*	r_6: *NS*	r_7: *NS*	r_8: *ZE*	r_9: *PS*
	ZE	r_{10}: *NS*	r_{11}: *NS*	r_{12}: *ZE*	r_{13}: *PS*	r_{14}: *PS*
	PS	r_{15}: *NS*	r_{16}: *ZE*	r_{17}: *PS*	r_{18}: *PS*	r_{19}: *PB*
	PB	r_{20}: *ZE*	r_{21}: *PS*	r_{22}: *PS*	r_{23}: *PB*	r_{24}: *PB*

A Figura 50 mostra as funções de pertinência utilizadas para cada variável de entrada e saída. Vale lembrar que as duas entradas possuem as mesmas funções de pertinência.



(a) Entradas centralização do robô entre as paredes laterais e paralelismo à parede direita

(b) Variável de saída ângulo das rodas frontais do robô

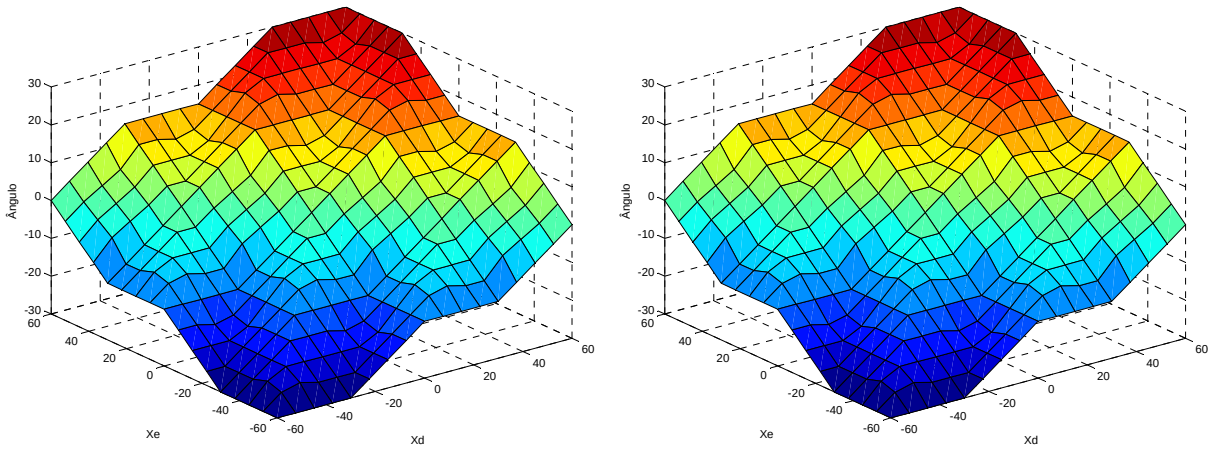
Figura 50: Funções de pertinência

A Tabela 7 mostra os valores testados na entrada dos sensores, onde pode-se ver além desses valores, as regras disparadas, de acordo com a Tabela 6. Também é possível ver a quantidade de termos linguísticos ativos na entrada da defuzzificação, o número de ciclos de clock, o tempo de execução em microsegundos, baseado no clock de 119,574 MHz e o valor escalar do resultado da defuzzificação ou da saída do CNP.

Tabela 7: Resultados do CNP para a aplicação do controle do ângulo das rodas

x_d	x_e	Regras Disparadas	Qtd de Defuzzy	Ciclos de Clock	Tempo (μseg)	Ângulo das Rodas
$+15$	$+15$	r_{12}, r_{13}, r_{17} e r_{18}	2	1460	12,21	$+7,5000$
$+21$	$+18$	r_{12}, r_{13}, r_{17} e r_{18}	2	1460	12,21	$+10,0000$
-10	-30	r_1, r_2, r_6 e r_7	2	1460	12,21	$-15,0000$
-10	-35	r_1, r_2, r_6 e r_7	2	1460	12,21	$-18,0000$
-21	$+18$	r_{11}, r_{12}, r_{16} e r_{17}	3	1507	12,60	$-1,1538$
$+39$	-39	r_3, r_4, r_8 e r_9	3	1507	12,60	$0,0000$
$+39$	-30	r_3, r_4, r_8 e r_9	3	1507	12,60	$+4,5000$
$+15$	0	r_7, r_8, r_{12} e r_{13}	3	1507	12,60	$+7,5000$

A Figura 51, mostra a superfície de controle baseada na configuração do controlador nebuloso para esta aplicação. O cálculo do erro quadrático com a Equação 29, tem-se o resultado de $Erro = 8.3045 \times 10^{-10}$, o que demonstra uma excelente precisão do CNP em relação ao MATLAB.



(a) MATLAB (b) CNP

Figura 51: Superfície de controle para a aplicação do controle de ângulo das rodas

Usando a frequência de clock de 100 MHz na FPGA, todo o controlador é executado, no pior dos casos, com Defuzzy3, em 2.569 ciclos de clock ou 25,69 microsegundos. Este é um dos possíveis valores de frequência de clock que pode ser selecionada na placa Virtex 5, mas os resultados da síntese mostram, que a frequência de clock máxima aceita pelo projeto desenvolvido para esta aplicação é de 119,574 MHz, o que resultaria em 21,48 microsegundos. Como o bloco `FP` não é contabilizado no ciclo normal do controlador em uma malha de controle, a mesma será de no máximo 1.507 ciclos de clock ou 12,60 microsegundos, considerando o valor máximo permitido de frequência de clock. A Figura 52

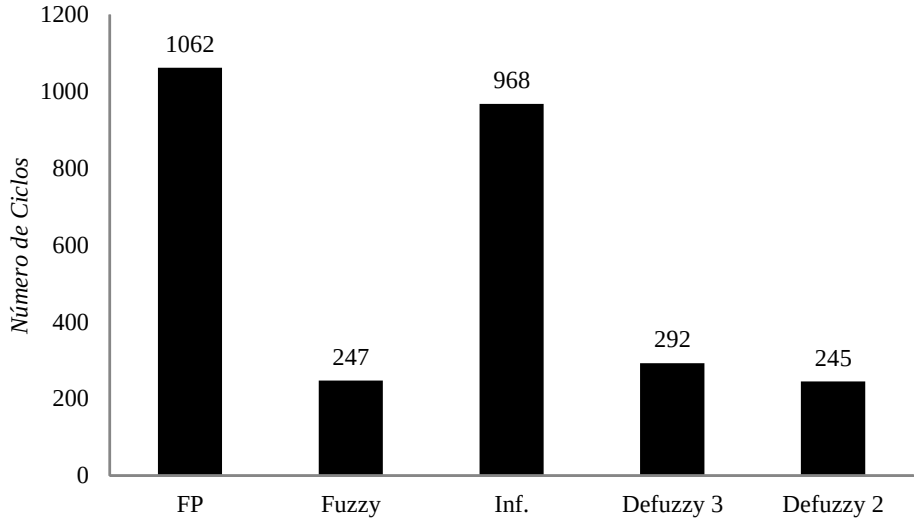


Figura 52: Número de ciclos de clock necessários para a execução dos blocos na FPGA

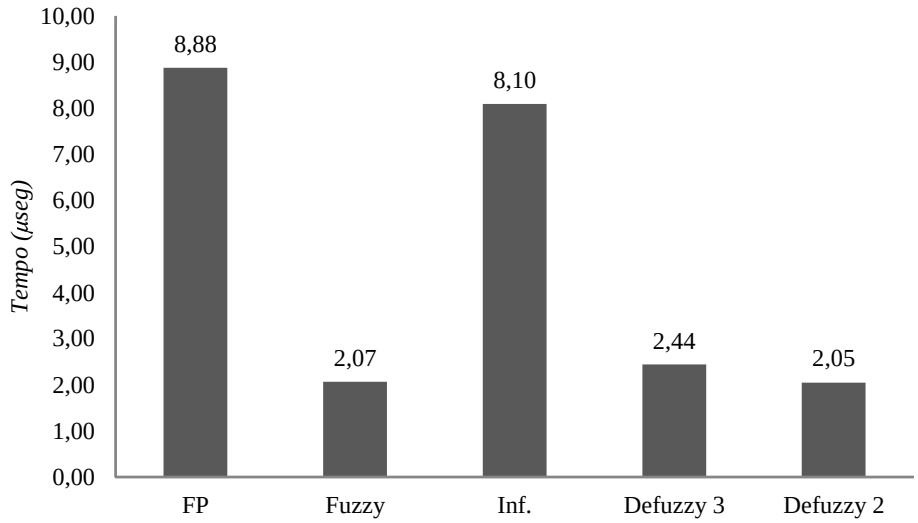


Figura 53: Tempos de execução dos blocos na FPGA em microsegundos

mostra o número de ciclos de clock para cada bloco do CNP, inclusive as variações nos números de ciclos para o bloco `Defuzzy`. Já a Figura 53 mostra os tempos de execução de cada bloco, utilizando a frequência máxima de clock.

A Figura 54 mostra a área usada na FPGA para executar todo o controlador nebuloso. Dos 69.120 LUTs disponíveis na FPGA utilizada, são usados 48,7%.

4.3.2 Controle da velocidade do robô

Como mencionado anteriormente, este controle possui 2 variáveis de entrada, o ângulo das rodas e a distância à parede frontal, 9 regras descritas na Tabela 8, 3 termos linguísticos e 1 variável de saída, velocidade de movimento do robô.

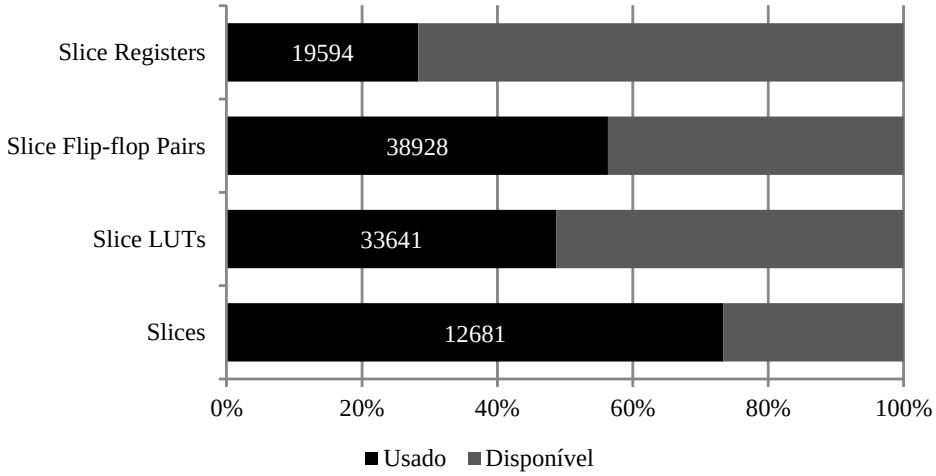
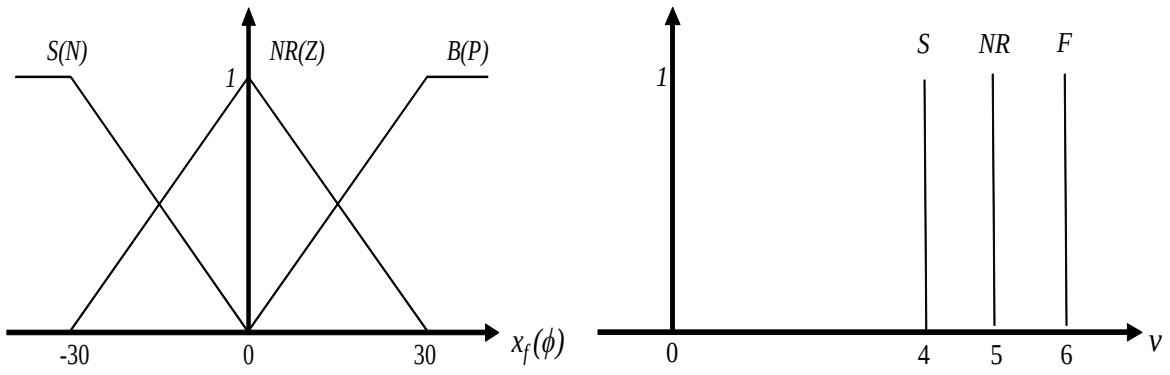


Figura 54: Utilização de área de *hardware*

Tabela 8: Regras nebulosas para a aplicação do controle da velocidade do robô

regras		X_f		
		S	NR	B
ϕ	P	r_0: S	r_1: S	r_2: NR
	Z	r_3: S	r_4: NR	r_5: F
	N	r_6: S	r_7: S	r_8: NR

A Figura 55 mostra as funções de pertinência utilizadas para cada variável de entrada e saída. Vale lembrar que as duas entradas possuem as mesmas funções de pertinência.



(a) Entradas ângulo das rodas e a distância à parede frontal

(b) Variável de saída velocidade de movimento do robô

Figura 55: Gráficos das Funções de Pertinência

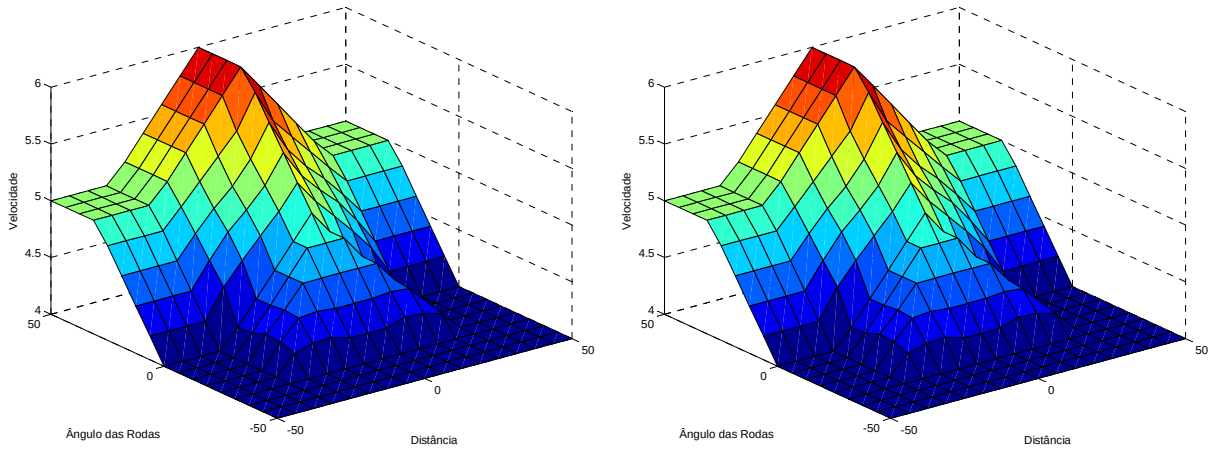
A Tabela 9 mostra os valores testados na entrada dos sensores, onde pode-se ver além destes, as regras disparadas, de acordo com a Tabela 8. Também é possível ver a

quantidade de termos linguísticos ativos na entrada da defuzzificação, o número de ciclos de clock, o tempo de execução em microsegundos, baseado no clock de 119,574 MHz e o valor escalar do resultado da defuzzificação ou da saída do CNP.

Tabela 9: Resultados do CNP para a aplicação do controle da velocidade do robô

x_f	ϕ	Regras Disparadas	Qtd de Defuzzy	Ciclos de Clock	Tempo (μseg)	Velocidade
-20	-35	r_0 e r_1	1	653	5,67	$+4,0000$
-10	$+10$	r_3, r_4, r_6 e r_7	2	700	6,08	$+4,6667$
15	0	r_4 e r_5	2	700	6,08	$+5,5000$
$+10$	$+10$	r_1, r_2, r_4 e r_5	3	747	6,49	$+5,0000$
$+5$	-25	r_4, r_5, r_7 e r_8	3	747	6,49	$+4,4286$

A Figura 56, mostra a superfície de controle baseada na configuração do controlador nebuloso para esta aplicação. O cálculo do erro quadrático obtido com a Equação 29, tem-se o resultado de $Erro = 1,1737 \times 10^{-9}$, o que demonstra uma excelente precisão do CNP em relação ao MATLAB.



(a) MATLAB (b) CNP

Figura 56: Superfície de controle para a aplicação do controle de velocidade do robô

Usando a frequência de clock de 100 MHz na FPGA, todo o controlador é executado, no pior dos casos, com Defuzzy3, em 1.291 ciclos de clock ou 12,91 microsegundos. Os resultados da síntese mostram, que a frequência de clock máxima aceita para esta aplicação é de 115,101 MHz, o que resultaria em 11,22 microsegundos. Como o bloco `FP` não é contabilizado no ciclo normal do controlador em uma malha de controle, a mesma será de no máximo 747 ciclos de clock ou 6,49 microsegundos, considerando o valor máximo permitido de frequência de clock. A Figura 57 mostra o número de ciclos de clock para

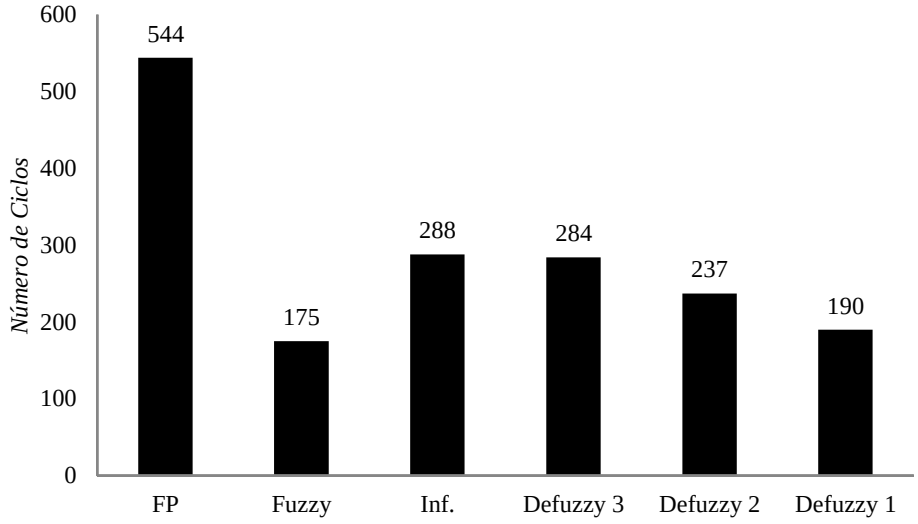


Figura 57: Número de ciclos de clock necessários para a execução dos blocos na FPGA

cada bloco do CNP, inclusive as variações nos números de ciclos para o bloco `Defuzzy`. Já a Figura 58 mostra os tempos de execução de cada bloco, utilizando a frequência máxima de clock.

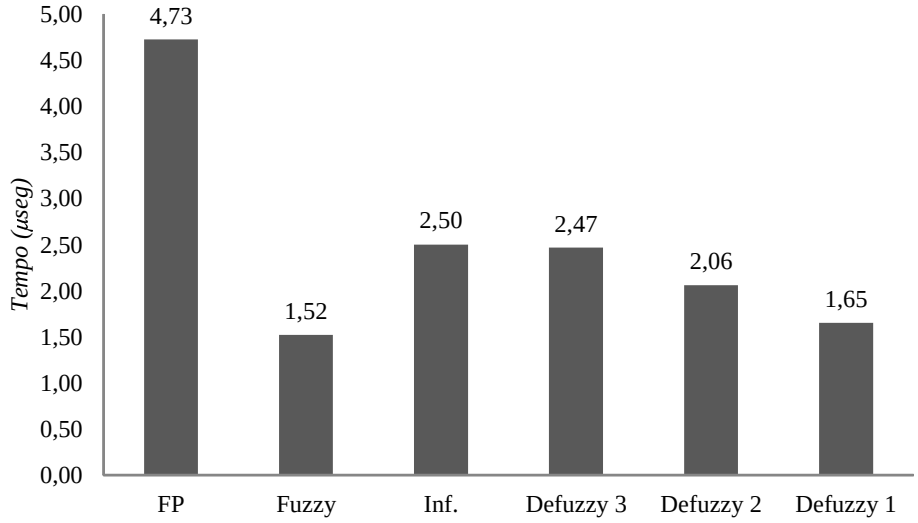


Figura 58: Tempos de execução dos blocos na FPGA em microsegundos

A Figura 59 mostra a área de *hardware* usada na FPGA para programar todo o controlador nebuloso. Dos 69.120 LUTs da FPGA são usados 41,8%.

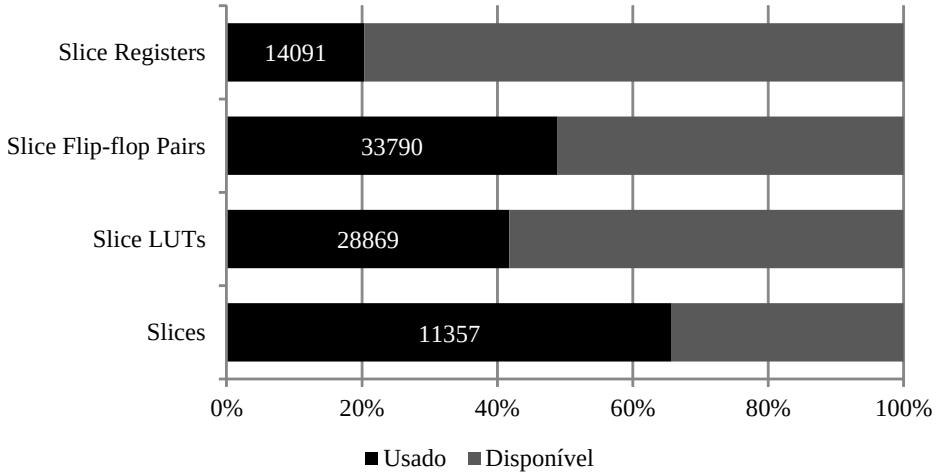


Figura 59: Utilização de área de *hardware*

4.4 Controle de Sinalização de Saída de uma Via Expressa

De acordo com a aplicação do trabalho desenvolvido por Yuan Changliang (CHANGLIANG; HONGHAI, 2010), o controle do semáforo da estrada auxiliar na saída de uma via expressa, ilustrada na Figura 60, pretende controlar a densidade da passagem de veículos da via expressa para a estrada auxiliar, atuando no ciclo de trabalho do semáforo para os veículos que já estão na estrada auxiliar.

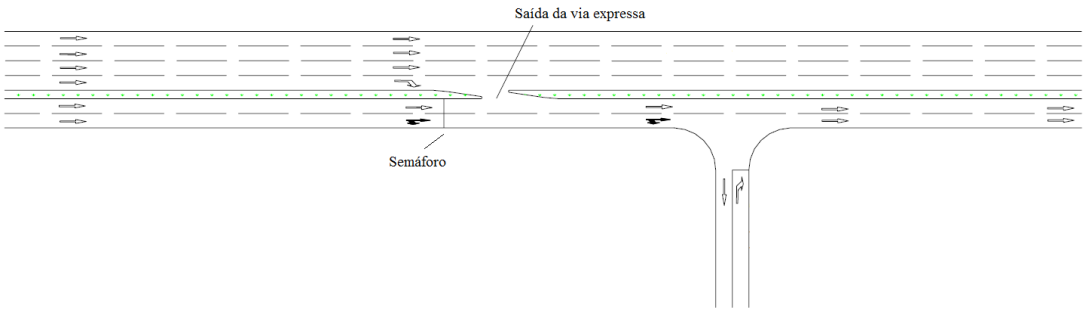
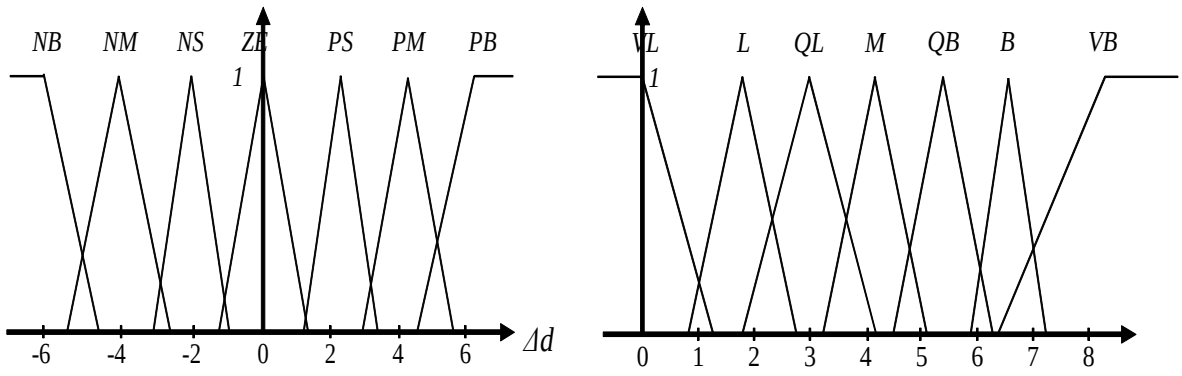


Figura 60: Modelo de via expressa para a estrada auxiliar em Beijing (CHANGLIANG; HONGHAI, 2010)

A saída deste controle nebuloso é a medição da taxa de veículos por hora na saída da via expressa, a qual é utilizada na Equação 33 para o cálculo do tempo de ciclo de trabalho do semáforo. Nesta equação C é o ciclo de trabalho, n é o número de pistas da estrada auxiliar, m é o número de carros que passam em cada pista a cada ciclo e r é a saída do controle nebuloso.

$$C = \frac{3600 \times (n \times m)}{r} \tag{33}$$

Este controle possui 1 variável de entrada que modela a variação da densidade de passagem de veículos na saída da via expressa, 6 regras descritas na Tabela 10, 7 termos linguísticos e 1 variável de saída que representa a medição da taxa de veículos por hora na saída da via expressa. A Figura 61 mostra as funções de pertinência utilizadas para cada variável de entrada e saída.



(a) Variável de entrada variação da densidade de passagem de veículos na saída da via expressa

(b) Variável de saída medição da taxa de veículos por hora na saída da via expressa

Figura 61: Funções de pertinência

Tabela 10: Regras nebulosas para a aplicação do controle de sinalização de via expressa

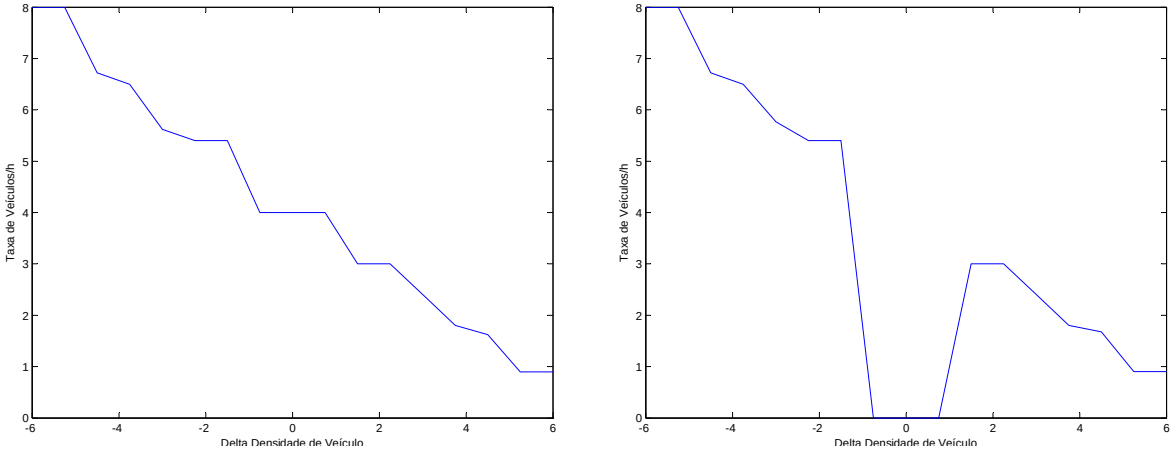
Δ densidade de passagem de veículos						
NB	*NM*	*NS*	*ZE*	*PS*	*PM*	*PB*
r_0: *VB*	r_1: *B*	r_2: *QB*	–	r_3: *QL*	r_4: *L*	r_5: *VL*

A Tabela 11 mostra os valores testados na entrada do controlador, onde pode-se ver, além destes, as regras disparadas. Também é possível ver a quantidade de termos linguísticos ativos na entrada da defuzzificação, o número de ciclos de clock, o tempo de execução em microsegundos, baseado no clock de 110,988 MHz e o valor escalar do resultado da defuzzificação ou da saída do CNP.

A Figura 62, mostra a curva de controle baseada na configuração do controlador nebuloso para esta aplicação. Com o cálculo do erro quadrático (Equação 29), tem-se o resultado de $Erro = 2.8250$. Nesta aplicação, a regra para o termo linguístico centrado em zero foi omitida (vide Tabela 10). Para o CNP, se nenhuma regra for disparada, a saída do controlador será zero, mas o MATLAB considera a saída do controlador como o

Tabela 11: Resultados do CNP para a aplicação do controle de sinalização de via expressa

Δd	Regras Disparadas	Qtd de Defuzzy	Ciclos de Clock	Tempo (μseg)	Taxa Veículos/h
$+0,50$	-	0	599	5,40	$0,0000$
$-4,05$	r_1	1	799	7,20	$6,5000$
$+1,00$	r_3	1	799	7,20	$3,0000$
$-2,80$	r_2	1	799	7,20	$5,4000$
$-5,20$	r_0	1	799	7,20	$8,0000$
$+2,00$	r_3	1	799	7,20	$3,0000$
$+5,50$	r_5	1	799	7,20	$0,9000$
$-5,00$	r_0 e r_1	2	846	7,62	$7,7007$
$+3,00$	r_3 e r_4	2	846	7,62	$2,4000$



(a) MATLAB (b) CNP

Figura 62: Curva de controle para esta aplicação

meio do domínio da variável de saída, que neste caso é 4. Por isso, ocasiona a diferença gritante entre as duas curvas de controle geradas por cada um e também pelo valor de erro tão alto. Entretanto, é possível verificar que, com exceção do centro do gráfico, que está, aproximadamente, entre os valores -1 e 1, as duas curvas de controle são semelhantes. Este infortúnio poderia ser resolvido, simplesmente, incluindo a regra faltante da Tabela 10. Entretanto, o comportamento do MATLAB é o correto e o CNP deverá ser corrigido prevendo esta condição.

Usando a frequência de clock de 100 MHz na FPGA, todo o controlador é executado, no pior dos casos, com Defuzzy2, em 2.426 ciclos de clock ou 24,26 microsegundos. Os resultados da síntese mostram, que a frequência de clock máxima aceita pelo projeto desenvolvido para esta aplicação é de 110,988 MHz, o que resultaria em 21,86 microsegundos. Como o bloco `FP` não é contabilizado no ciclo normal do controlador em uma

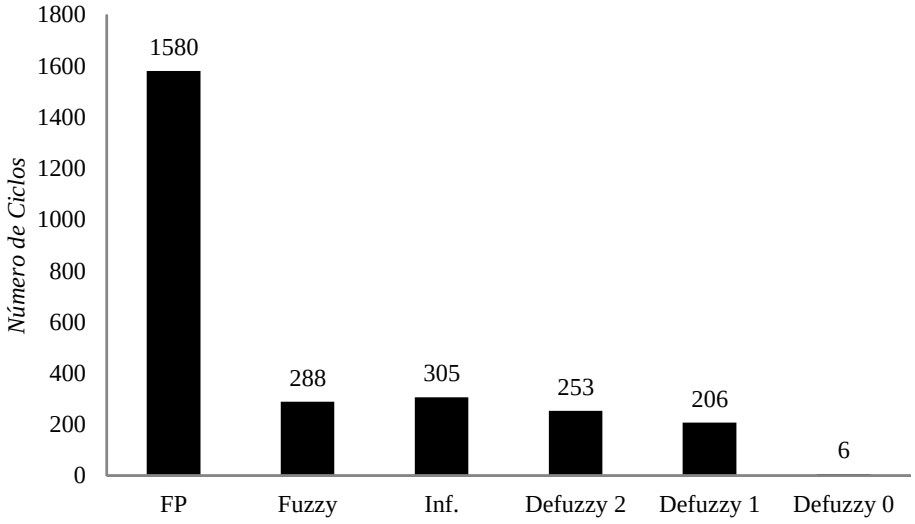


Figura 63: Número de ciclos de clock necessários para a execução dos blocos na FPGA

malha de controle, a mesma será de no máximo 846 ciclos de clock ou 7,62 microsegundos, considerando o valor máximo permitido de frequência de clock. A Figura 63 mostra o número de ciclos de clock para cada bloco do CNP, inclusive as variações nos números de ciclos para o bloco `Defuzzy`. Já a Figura 64 mostra os tempos de execução de cada bloco, utilizando a frequência máxima de clock.

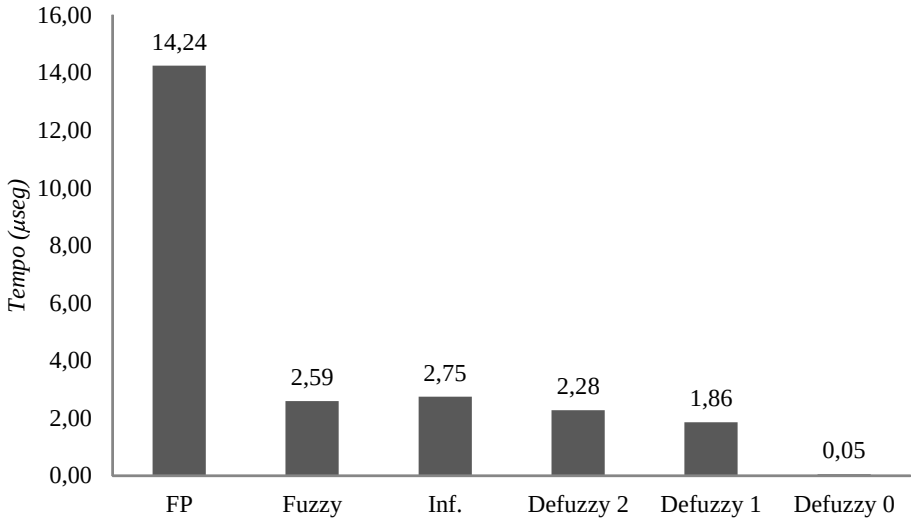


Figura 64: Tempos de execução dos blocos na FPGA em microsegundos

A Figura 65 mostra a área usada na FPGA para implementar todo o controlador nebuloso. Dos 69.120 LUTs da FPGA, são usados 35,1%.

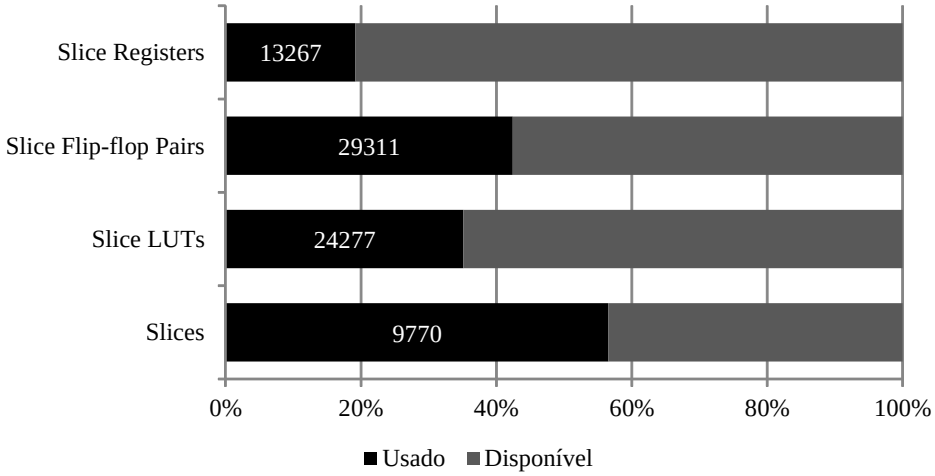


Figura 65: Utilização de área de *hardware* na FPGA

4.5 Controle de Navegação Autônoma para Robôs com Rodas

De acordo com a aplicação do trabalho desenvolvido por Lin (LIN; HUANG; CHUANG, 2005), o controle de navegação de robôs com rodas utiliza uma série de malhas de controles para navegar em uma superfície seguindo uma trajetória pré-definida. Na Figura 66 pode-se ver o esquema do robô utilizado.

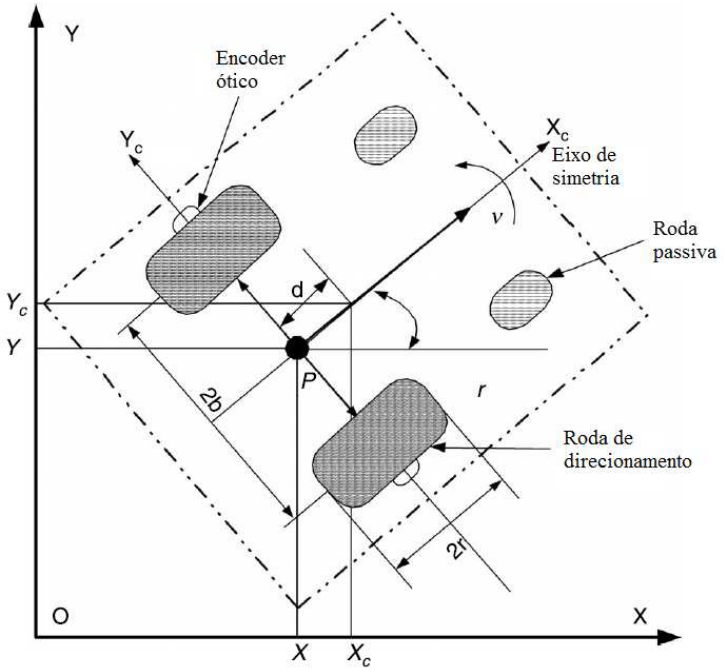


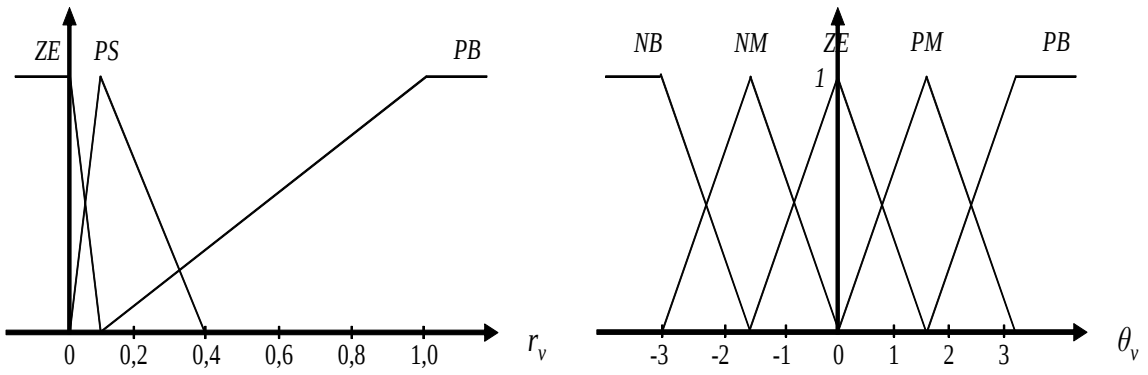
Figura 66: Modelo do robô com rodas utilizado nesta aplicação (LIN; HUANG; CHUANG, 2005)

Tabela 12: Regras nebulosas para a aplicação do controle de navegação autônoma

regras		Raio		
		ZE	*PS*	*PB*
Ângulo	*PB*	r_0: *ZE*	r_1: *NM*	r_2: *NB*
	PM	r_3: *ZE*	r_4: *PM*	r_5: *PB*
	ZE	r_6: *ZE*	r_7: *PM*	r_8: *PB*
	NM	r_9: *ZE*	r_{10}: *NM*	r_{11}: *NB*
	NB	r_{12}: *ZE*	r_{13}: *NM*	r_{14}: *NB*

Esta aplicação é formada por três subcontroladores, são eles: o controle de direção, que utiliza dois controladores com duas entradas e uma saída, cada um; o controle de velocidade linear, que utiliza um controlador com duas entradas e uma saída; e o controle de velocidade angular, que utiliza um controlador com duas entradas e uma saída. Apesar desta aplicação possuir quatro controladores, nesta dissertação será apresentado apenas um deles, pois os controladores são idênticos dois a dois, ou seja, as funções de pertinência e regras dos controladores de velocidade linear e angular são iguais entre si. O mesmo ocorre nos dois controladores para direção, sendo que este último não foi possível realizar os testes, por causa de algum problema no processo da síntese, que após 8 horas gera erro de memória.

A aplicação apresentada abaixo é do controle de velocidade angular, a qual possui 2 variáveis de entrada que modelam o raio e o ângulo na forma polar representando o erro e a variação do erro de velocidade, 15 regras descritas na Tabela 12, 5 termos linguísticos e 1 variável de saída que representa a velocidade linear de movimento. A Figura 67 mostra as funções de pertinência utilizadas para cada variável de entrada e saída.



(a) Variável de entrada raio

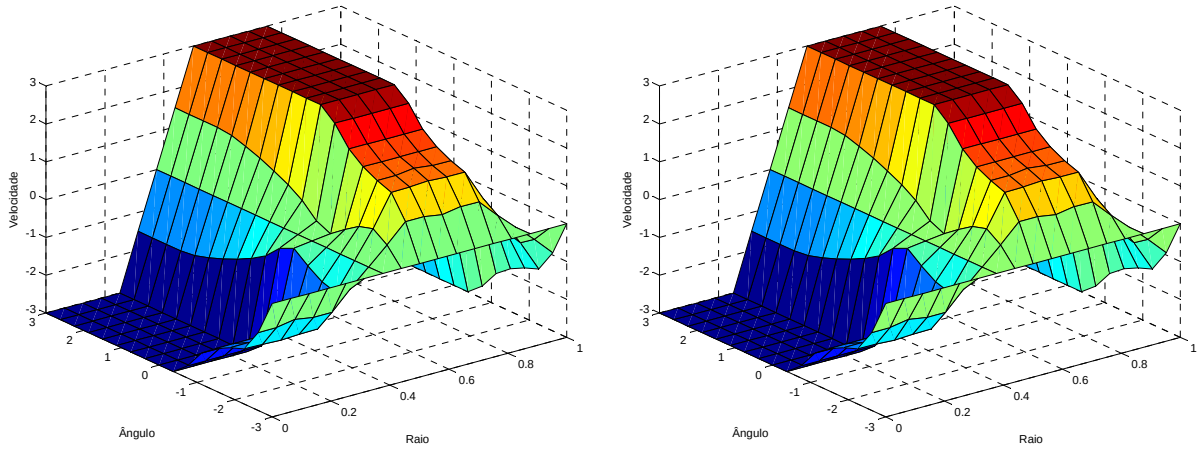
(b) Variável de entrada ângulo e variável de saída velocidade linear

Figura 67: Funções de pertinência

Tabela 13: Resultados do CNP para a aplicação do controle de navegação autônoma

Raio	Ângulo	Regras Disparadas	Qtd de Defuzzy	Ciclos de Clock	Tempo (μseg)	Velocidade
$0,00$	$+1,0$	r_3 e r_6	1	1043	9,28	$0,0000$
$0,80$	$-2,0$	r_{11} e r_{14}	1	1043	9,28	$-3,0000$
$0,50$	$+2,5$	r_2 e r_5	2	1090	9,70	$-0,4286$
$0,05$	$-1,0$	r_6, r_7, r_9 e r_{10}	3	1137	10,11	$-0,1875$
$0,09$	$+2,0$	r_0, r_1, r_3 e r_4	3	1137	10,11	$+0,4545$
$0,30$	$+2,0$	r_1, r_2, r_4 e r_5	4	1184	10,53	$0,0000$
$0,20$	$-0,5$	r_7, r_9, r_{11} e r_{12}	4	1184	10,53	$+0,4091$
$0,20$	$+2,5$	r_1, r_2, r_4 e r_5	4	1184	10,53	$-0,4091$

A Tabela 13 mostra os valores testados na entrada dos sensores, onde pode-se ver, além destes, as regras disparadas, de acordo com a Tabela 12. Também é possível ver a quantidade de termos linguísticos ativos na entrada da defuzzificação, o número de ciclos de clock, o tempo de execução em microsegundos, baseado no clock de 112,410 MHz e o valor escalar do resultado da defuzzificação ou da saída do CNP.



(a) MATLAB (b) CNP

Figura 68: Superfície de controle para a aplicação de Navegação Autônoma para Robôs com Rodas

A Figura 68, mostra a superfície de controle baseada na configuração do controlador nebuloso para esta aplicação. O cálculo do erro quadrático com a Equação 29, tem-se o resultado de $Erro = 3,1237 \times 10^{-7}$, o que demonstra uma excelente precisão do CNP em relação ao MATLAB.

Usando a frequência de clock de 100 MHz na FPGA, todo o controlador é executado, no pior dos casos, com Defuzzy4, em 2.246 ciclos de clock ou 22,46 microsegundos. Os resultados da síntese mostram, que a frequência de clock máxima aceita pelo projeto

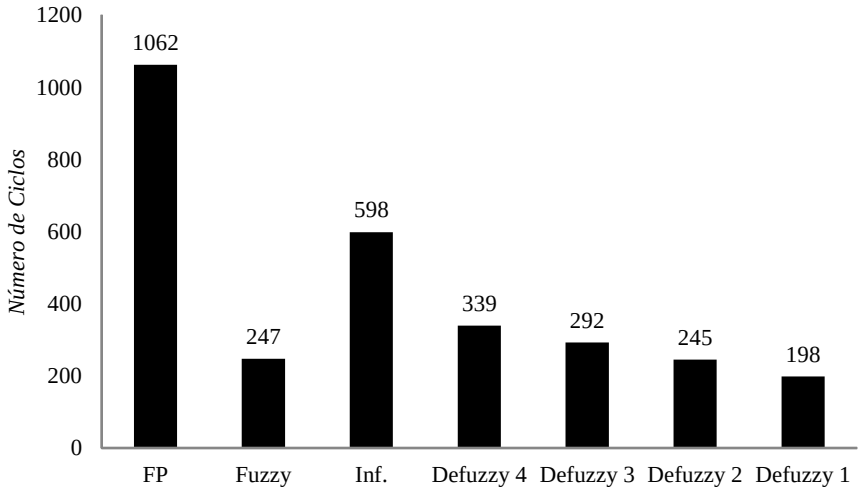


Figura 69: Número de ciclos de clock necessários para a execução dos blocos na FPGA

desenvolvido para esta aplicação é de 112,410 MHz, o que resultaria em 19,98 microsegundos. Como o bloco `FP` não é contabilizado no ciclo normal do controlador em uma malha de controle, a mesma será de no máximo 1.184 ciclos de clock ou 10,53 microsegundos, considerando o valor máximo permitido de frequência de clock. A Figura 69 mostra o número de ciclos de clock para cada bloco do CNP, inclusive as variações nos números de ciclos para o bloco `Defuzzy`. Já a Figura 70 mostra os tempos de execução de cada bloco, utilizando a frequência máxima de clock.

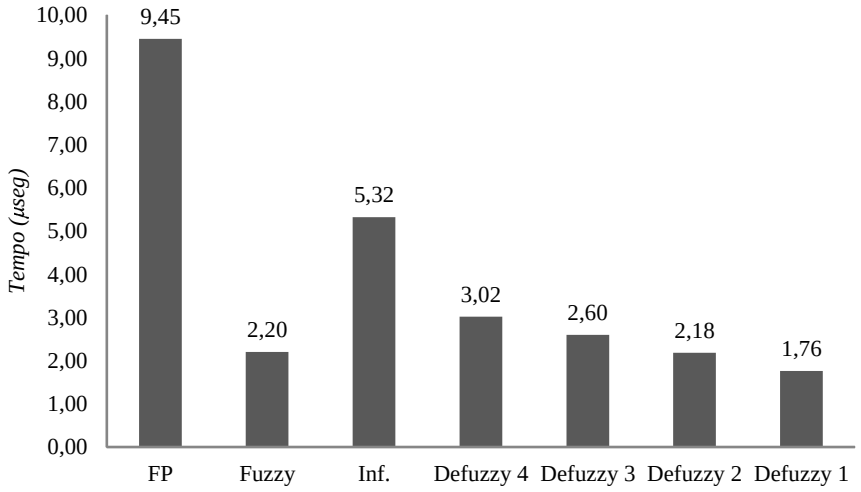


Figura 70: Tempos de execução dos blocos na FPGA em microsegundos

A Figura 71 mostra a área usada na FPGA para programar todo o controlador nebuloso. Dos 69.120 LUTs da FPGA, são usados 54,1%.

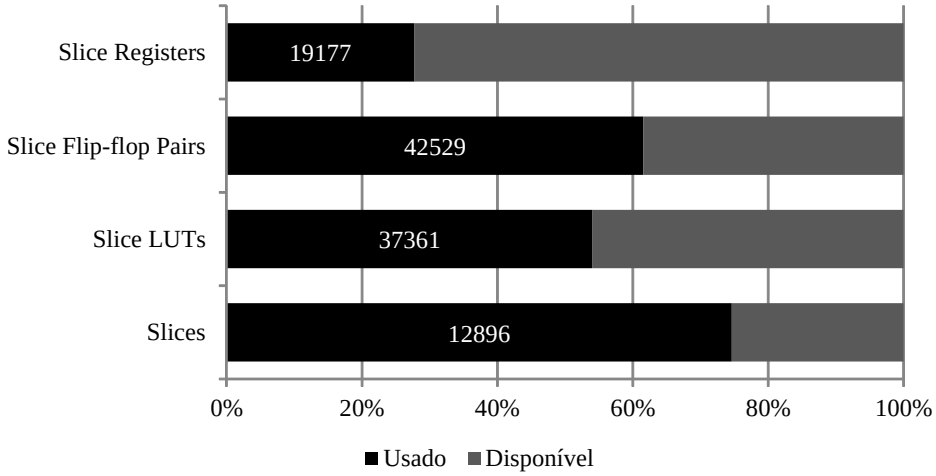


Figura 71: Utilização de área de *hardware* na FPGA

4.6 Considerações Finais do Capítulo

Neste capítulo, foram implementadas várias aplicações utilizando o *hardware* reconfigurável para o CNP, detalhando os valores de resultados na saída dos controladores, a comparação entre as superfícies de controle do MATLAB e do CNP, além dos resultados de síntese em FPGA como área utilizada e ciclos de clocks necessários para a execução do controlador. O capítulo seguinte finaliza este trabalho, abordando suas principais conclusões, bem como os pontos mais relevantes da presente dissertação. Também serão tratadas direções para trabalhos futuros.

Capítulo 5

CONCLUSÕES E TRABALHOS FUTUROS

O CONTROLADOR nebuloso proposto nesta dissertação foi descrito e implementado em *hardware*. Seu funcionamento foi validado através de simulação. Uma avaliação de custo e desempenho foi feita usando 6 diferentes aplicações. Neste Capítulo são apresentadas as conclusões pertinentes ao trabalho realizado e as possíveis melhorias a serem feitas.

5.1 Conclusões

Esta dissertação abordou a teoria de conjuntos nebulosos e seu uso em controladores nebulosos. Este estudo foi motivado pela carência do uso de controladores nebulosos na indústria Brasileira, os quais podem facilitar o controle de processos não-lineares, como uma alternativa aos controladores PID. O controlador nebuloso tem a capacidade de considerar tantas variáveis de processo e sinais de controle quantos necessários ao controle. Em geral, controladores PID não têm tanto sucesso para controlar processos com um grande número de variáveis de processos e sinais de controle. Na arquitetura proposta, não há limite máximo estabelecido para o número de variáveis de entrada e de saída. O limite é imposto pela área de *hardware* máxima da FPGA utilizada para implementar o controlador nebuloso.

O CNP realiza o ciclo de controle em microsegundos. Na área de automação industrial, qualquer CLP (Controlador Lógico Programável) usualmente executa todo o ciclo de controle em milisegundos. Isso torna o CNP elegível para uso na maior parte das aplicações de controle na indústria, em relação ao tempo de execução.

A capacidade de reconfiguração do CNP, alterando a quantidade de componentes na arquitetura, permite o uso consciente de área de *hardware*, de acordo com a aplicação. Isto evita um consumo excessivo para pequenas aplicações, como aconteceria na utilização de um controlador rígido, preparado para aplicações mais complexas. Um ponto forte do CNP é o esforço, mínimo, para reconfigurar o mesmo. É necessário, apenas, ter o controlador nebuloso objeto do projeto, ajustar os parâmetros do CNP, de acordo com a aplicação em questão e realizar o processo de síntese.

A utilização da unidade de ponto flutuante (FPU), para realizar os cálculos necessários ao CNP, permitiu obter valores precisos de saída em comparação aos obtidos pela ferramenta MATLAB, conforme Capítulo 4. Com esta característica e considerando o tempo de execução na ordem de microsegundos, o CNP possui um custo-benefício atrativo, mesmo impondo a utilização de uma grande área de *hardware*.

O cálculo do centroide, conforme a Equação 28, realizado pelo bloco `Defuzzy`, é executado, de tal forma que, a função de pertinência utilizada para cada termo linguístico é um *singleton*. Então, a quantidade de pontos utilizados para o cálculo é equivalente ao número de termos linguísticos para cada variável de saída. É possível verificar isto pelo número de iterações realizadas no Algoritmo 4. Esta estratégia foi adotada para minimizar o consumo de área de *hardware*, além do número de ciclos de execução do bloco.

A arquitetura proposta tem algumas restrições para garantir a entrega de resultados corretos, como o número mínimo de 2 termos linguísticos por variável tanto de entrada quanto de saída. Este é o caso em qualquer controlador nebuloso. Portanto, este fato não representa um problema para o uso do CNP em qualquer aplicação. Cada termo linguístico sempre será representado por uma função de pertinência triangular. Todavia, o valor máximo de regras não é pré-determinado. Então, o especialista pode usar qualquer número de regras, que podem ser acomodadas no dispositivo reconfigurável em uso. Obviamente, a configuração da quantidade de regras maior que o número máximo possível de regras para o controlador, acarretará na criação de uma área de memória, que nunca será usada, gerando um consumo de área de *hardware* desnecessário. Este número máximo de regras é imposto pela multiplicação dos números de termos linguísticos Q_i, de cada variável de entrada i, conforme: $\Pi_{i=1}^{N} Q_i$.

A forma como as regras são utilizadas pelo bloco `Inferencia` permite, na descrição de cada regra, que não seja necessário levar em consideração uma ou mais variáveis na premissa, além da possibilidade de não gerar os resultados inferidos para uma ou mais variáveis de saída no consequente. Esta característica do CNP pode ser utilizada, apenas, mantendo em zero os bits referentes à variável de entrada ou saída, que não devem ser levadas em consideração, em cada regra. Este caso se aplica na configuração do CNP para modelos nebulosos com 2 ou mais entradas e saídas. Esta possibilidade de configuração da regra aumenta a flexibilidade do CNP para qualquer aplicação, inclusive nas de controles mais complexos.

Esta arquitetura possui um grande benefício que é o paralelismo inerente à função dos blocos de fuzzificação e o paralelismo inerente à função dos blocos de defuzzificação. Dessa forma, é possível aumentar a quantidade de entradas e saídas sem afetar o tempo de processamento do controlador. Esta característica é um benefício para o uso do CNP em aplicações complexas, apesar deste fato acarretar no uso de uma área de *hardware* maior. Já o aumento na quantidade de termos linguísticos e de regras afetam o tempo de processamento nos blocos de fuzzificação e inferência, respectivamente. Isso, também aumenta a área de *hardware* necessária.

5.2 Trabalhos Futuros

Nesta seção, são citadas algumas possíveis modificações no CNP, com o intuito de melhorar sua capacidade de configuração e ampliar o uso nas mais diversas aplicações. Também são levantadas propostas para futuros trabalhos com o uso do CNP.

Sem dúvida, uma primeira sugestão é estender a implementação do CNP para trabalhar com funções de pertinência em outros formatos, como, por exemplo, trapézios ou função Gaussiana, ao invés de apenas triângulos. Isto permitirá a utilização do CNP em uma gama maior de aplicações. A implementação de outras formas de função de pertinência aumentará o consumo de área e o tempo de execução, principalmente no processo de *configuração* do CNP, ou seja, nos blocos de cálculo da função de pertinência. Esta alteração também acarretará em um acréscimo de tempo, referente ao cálculo de fuzzificação, pois no caso da função trapezoidal, serão tratadas 3 retas por termo linguístico, e não somente 2, como é o caso do triângulo. No caso da Gaussiana, o cálculo deverá ser feito baseado em uma aproximação da função Gaussiana realizadas por funções básicas,

tais como polinômios de 2^o e 3^o graus.

Uma segunda modificação será permitir a configuração do número de termos linguísticos de forma independente para cada variável de entrada e/ou saída. Isto gerará um ganho em consumo de área de *hardware* e também em tempo de execução.

Uma terceira modificação, diz respeito ao processo de defuzzificação, permitindo o uso de um método mais preciso de cálculo do centroide. Ao invés de usar o número de termos linguísticos da variável de saída como iterações do cálculo, este número deverá ser informado, como um parâmetro do controlador. Isto fará com que, este número de iterações do cálculo seja equivalente ao número de amostragem da discretização das funções de pertinência inferidas. Estes cálculos extras aumentarão tanto o consumo de área de *hardware* quanto o tempo de execução do bloco `Defuzzy`.

Por fim, seria interessante utilizar o CNP em uma malha de controle real e verificar o seu comportamento ao longo do tempo. Isto possibilitará a verificação do desempenho do mesmo em alcançar a estabilidade de controle. Para isto, será interessante o uso de um *benchmark* para analisar e comparar os resultados, da mesma forma que foram realizados os testes experimentais do Capítulo 4.

REFERÊNCIAS

AL-ERYANI, J. *Float Point Unit Manual*. 2006. Último acesso em outubro/2012. Disponível em: <http://opencores.org/websvn,filedetails?repname=fpu100& path=%2Ffpu100%2Fweb _uploads%2Ffpu_doc.pdf>.

ALVAREZ, J. et al. FPGA implementation of a fuzzy controller for automobile DC-DC converters. In: *IEEE International Conference on Field Programmable Technology*. Bangkok: IEEE, 2006. p. 237–240.

BERNSTEIN, D. *Control tutorial for MatLab: inverted pendulum*. December 2004. Disponível em: <www.engin.umich.edu/group/ctm/examples/pend/invpen.html>.

CHANGLIANG, Y.; HONGHAI, L. Fuzzy metering control on the auxiliary road signal of the expressway exit. In: *11th International Conference on Control, Automation, Robotics and Vision*. Singapura: IEEE, 2010. p. 2202–2207.

CHEKIRED, F. et al. Implementation of a MPPT fuzzy controller for photovoltaic systems on FPGA circuit. *Energy Procedia*, v. 6, p. 541–549, 2011.

CORDÓN, O.; HERRERA, F.; PEREGRÍN, A. Searching for basic properties obtaining robust implication operators in fuzzy control. In: *Fuzzy Sets and Systems*. Amsterdam: Elsevier North-Holland, Inc., 2000. v. 111, n. 2, p. 237–251.

DIAO, Y.; HELLERSTEIN, J.; PAREKH, S. Using fuzzy control to maximize profits in service level management. *IBM Systems Journal*, v. 41, n. 3, p. 403–420, 2002.

ESRAGH, F.; MAMDANI, E. A general approach to linguistic approximation. In: MAMDANI, E.; GAINES, B. (Ed.). *Fuzzy Reasoning and Its Applications*. London: Academic Press, 1981. p. 501–519.

FRANKE, K.; KÖPPEN, M.; NICKOLAY, B. Fuzzy image processing by using Dubois and Prade fuzzy norm. In: *Proceedings of 15th International Conference on Pattern Recognition*. Barcelona, Spain: IEEE, 2000. v. 3, p. 514–517.

GHIDARY, S. et al. Multi-modal human robot interaction for map generation. In: *Proceedings of International Conference on Intelligent Robots and Systems*. Maui, HI: IEEE, 2001. v. 4, p. 2246–2251.

GOMIDE, F. A. C.; GUDWIN, R. R.; TANSCHEIT, R. Conceitos fundamentais da teoria de conjuntos fuzzy, lógica fuzzy e aplicações. In: *Proceedings of the Sixth International Fuzzy Systems Association World Congress*. Brasil: International Fuzzy Systems Association, 1995. p. 1–38.

HAREL, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, v. 8, p. 231–274, 1987.

HASSAN, M. Y.; SHARIF, W. F. Design of FPGA based PID-like fuzzy controller for industrial applications. *IAENG International Journal of Computer Science*, v. 2, n. 34, Novembro 2007.

JUNHOR, P. et al. Control of machine of welding MIG using controller fuzzy. In: *Proceedings of International Conference on Electric Machines and Drives*. Texas, USA: IEEE, 2005. p. 1845–1849.

KIKUCHI, H.; S., N. Implication-based learning. In: PHUONG, N. H.; OHSATO, A. (Ed.). *The Vietnam - Japan Bilateral Symposium on Fuzzy Systems and Applications*. Halong Bay, Vietnam: University P.&M. Curie, 1998.

KIM, D. An implementation of fuzzy logic controller on the reconfigurable FPGA system. *IEEE Transactions on Industrial Electronics*, v. 47, n. 3, p. 703–715, Junho 2000.

LI, T.-H. S.; CHANG, S.-J.; CHEN, Y.-X. Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot. *IEEE Transaction on Industrial Electronics*, v. 50, n. 5, p. 867–880, 2003.

LIN, W.-S.; HUANG, C.-L.; CHUANG, M.-K. Hierarchical fuzzy control for autonomous navigation of wheeled robots. *IEE Proc.-Control Theory Appl.*, v. 152, n. 5, p. 598–606, 2005.

MACHADO, L. B. P. et al. Development of an embedded module using fpga technology and fuzzy. *Revista IEEE América Latina*, v. 9, p. 753–760, 2011.

MAGDALENA, L.; VELASCO, J. Fuzzy rule-based controllers that learn by evolving their knowledge base. In: HERRERA, F.; VERDEGAY, J. (Ed.). *Fuzzy Logic and Soft Computing*. Heidelberg: Physica-Verlag, 1996. p. 172–201.

MAMDANI, E.; ASSILIAN, S. An experiment in linguistic synthesis of fuzzy controllers. *International Journal of Man-Machine Studies*, v. 7, n. 1, p. 1–13, 1975.

MATHWORKS. *Matlab*. Último acesso em outubro/2012. Disponível em: <http://www.mathworks.com/products/matlab/>.

MCKENNA, M.; WILAMOWSKI, B. Implementing a fuzzy system on a field programmable gate array fuzzy sets and systems. In: *Proceedings IJCNN '01 - International Joint Conference on Neural Networks*. Washington, DC: IEEE, 2001. v. 1, p. 189–194.

MODELSIM Software. Mentor Graphics, 2012. Disponível em: <http://www.mentor.com/products/fv/modelsim/>.

NAVABI, Z. *VHDL: Analysis and Modeling of Digital Systems*. Second edition. USA: McGraw Hill College, 1998.

NEDJAH, N.; MOURELLE, L. Introducing you to fuzziness. In: KACPRZYK, J. (Ed.). *Fuzzy Systems Engineering - Theory and Practice*. Heidelberg, Berlin: Springer, 2005. v. 181, p. 1–21.

NIOS II Processor: The World's Most Versatile Embedded Processor. Altera Embedded Alliance, 2013. Último acesso em janeiro/2013. Disponível em: <http://www.altera.com/devices/processor/nios2/ni2-index.html>.

POORANI, S. et al. FPGA based fuzzy logic controllers for electric vehicle. *Journal of the Institution of Engineers*, Singapore, v. 45, n. 5, p. 1–14, 2005.

RACHEL, F. M. *Proposta de um controlador automático de trens utilizando lógica nebulosa preditiva*. Dissertação (Mestrado) — Escola Politécnica, Universidade de São Paulo, São Paulo, 2006.

RADECKI, T. An evaluation of the fuzzy set theory approach to information retrieval. In: TRAPPL, R.; FINDLER, N.; HORN, W. (Ed.). *Progress in Cybernetics and System Research*. NY: Hemisphere Publishing Company, 1982. v. 11.

RUBAAI, A.; JERRY, J.; SMITH, S. Performance evaluation of fuzzy switching position controller for automation and process industry control. *IEEE Transactions on Industry Applications*, v. 47, n. 5, p. 2274–2282, 2011.

SANDRES, P.; NEDJAH, N.; MOURELLE, L. Reconfigurable hardware for fuzzy controller. In: ATANASSOV, K. et al. (Ed.). *New Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics. Volume II: Applications*. Warsaw: IBS PAN, 2011. v. 2, p. 243–277.

SANDRES, P.; NEDJAH, N.; MOURELLE, L. Hardware reconfigurável para controlador difuso. In: BEDREGAL, B. et al. (Ed.). *Recentes Avanços em Sistemas "Fuzzy"*. Natal: II CBSF - Segundo Congresso Brasileiro de Sistemas Fuzzy, 2012. p. 888–902.

SANDRES, P.; NEDJAH, N.; MOURELLE, L. Massively parallel scalable reconfigurable hardware for fuzzy controllers. In: *LASCAS2013 - 4th Latin American Symposium on Circuits and Systems*. Cusco, Peru: IEEE, 2013. (À ser publicado em março).

SANDRES, P.; NEDJAH, N.; MOURELLE, L. Reconfigurable hardware for fuzzy controller. *International Journal of High Performance Systems Architecture (IJHPSA)*, 2013. (Aceito para publicação).

SHIM, D. et al. A comprehensive study of control design for an autonomous helicopter. In: *Proceedings of 37th. Conference on Decision and Control*. Tampa, FL: IEEE, 1998. v. 4, p. 3653–3658.

SINTHIPSOMBOON, K. et al. A hybrid of fuzzy and fuzzy self-tuning PID controller for servo electro-hydraulic system. In: *Proceedings of 6th IEEE Conference on Industrial Electronics and Applications*. Beijing, China: IEEE, 2011. p. 220–225.

SRINIVASAN, K.; LAKSHMI, P. Adaptive neuro-fuzzy controller for non-linear chemical mixing process. In: *Proceedings of 7th International Conference on Control, Automation, Robotics and Vision*. Singapore: IEEE, 2002. v. 3, p. 1626–1631.

SULAIMAN, N. et al. FPGA-based fuzzy logic: design and applications: a review. *IACSIT International Journal of Engineering and Technology*, v. 1, n. 5, p. 491–503, 2009.

TANSCHEIT, R.; GOMIDE, F.; TEIXEIRA, M. M. Modelagem e controle nebuloso. In: AGUIRRE, L. A. (Ed.). *Enciclopédia de Automática: controle e automação*. 1. ed. São Paulo: Blusher, 2007. III.

UMBERS, I.; KING, P. An analysis of human decision-making in cement kiln control and the implications for automation. *International Journal of Man-Machine Studies*, v. 12, n. 1, p. 11–23, 1980.

VARGENS, J. M.; TANSCHEIT, R.; VELLASCO, M. M. B. R. Previsão de produção agrícola baseada em regras linguísticas e lógica fuzzy. *Revista Controle & Automação*, v. 14, n. 2, p. 114–120, 2003.

WACHS, J.; STERN, H.; EDAN, Y. Parameter search for an image processing fuzzy C-means hand gesture recognition system. In: *Proceedings of IEEE International Conference on Image Processing*. Barcelona, Spain: IEEE, 2003. v. 3, p. 341–344.

WANG, W. Fuzzy PI+fuzzy ID controller for PMSM servo system. In: *Proceedings of Power and Energy Engineering Conference*. Wuhan, China: IEEE, 2011. p. 1–4.

XILINX, I. *Foundation series software*. 2012. Disponível em: <http://www.xilinx.com>.

ZADEH, L. Fuzzy sets. *Journal of Information and Control*, v. 8, n. 3, p. 338–353, 1965.

ZADEH, L. Fuzzy algorithms. *Journal of Information and Control*, v. 12, n. 2, p. 94–102, 1968.

ZADEH, L. Making computers think like people. *IEEE Spectrum*, v. 21, n. 8, p. 26–32, 1984.

ZADEH, L. Fuzzy logic. *IEEE Computer Journal*, v. 21, n. 4, p. 83–93, 1988.

ZHANG, J.; KNOLL, A. Designing fuzzy controllers by rapid learning. *Fuzzy Sets and Systems*, v. 101, n. 2, p. 287–301, 1999.

ZIMMERMANN, H.; ZYSNO, P. Latent connectives in human decision making. *Fuzzy Sets and Systems*, v. 4, n. 1, p. 37–51, 1980.