UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

DISSERTAÇÃO DE MESTRADO

Aplicando Lógica *Fuzzy* no Controle de Robôs Móveis usando Dispositivos Lógicos Programáveis e a Linguagem VHDL.

Maycon Mariano Nogueira

Orientadora: Prof. Dra. Suely Cunha Amaro Mantovani

Ilha Solteira – SP
2013

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Aplicando Lógica *Fuzzy* no Controle de Robôs Móveis usando Dispositivos Lógicos Programáveis e a Linguagem VHDL.

Maycon Mariano Nogueira

Orientadora: Prof. Dra. Suely Cunha Amaro Mantovani

Dissertação apresentada à Faculdade de Engenharia – UNESP – Campus de Ilha Solteira, para obtenção do título de Mestre em Engenharia Elétrica. Área de Conhecimento: Automação.

Ilha Solteira – SP
Maio

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Nogueira, Maycon Mariano.
N778a Aplicando lógica fuzzy no controle de robôs móveis usando dispositivos lógicos programáveis e a linguagem VHDL / Maycon Mariano Nogueira. -- Ilha Solteira: [s.n.], 2013
95 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento, Automação, 2013

Orientador: Suely Cunha Amaro Mantovani
Inclui bibliografia

1. Lógica fuzzy. 2. Robótica móvel. 3. Controlador inteligente. 4. Controlador fuzzy.

unesp UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE ILHA SOLTEIRA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

CERTIFICADO DE APROVAÇÃO

TÍTULO: Aplicando Lógica Fuzzy no Controle de Robôs Móveis usando Dispositivos Lógicos Programáveis e a Linguagem VHDL

AUTOR: MAYCON MARIANO NOGUEIRA
ORIENTADORA: Profa. Dra. SUELY CUNHA AMARO MANTOVANI

Aprovado como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica , Área: AUTOMAÇÃO, pela Comissão Examinadora:

Profa. Dra. SUELY CUNHA AMARO MANTOVANI
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. NOBUO OKI
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. CARLOS AURÉLIO FARIA DA ROCHA
Departamento de Engenharia Elétrica / Universidade Federal de Santa Catarina

Data da realização: 29 de maio de 2013.

DEDICATÓRIA

Dedico este trabalho à minha amada mãe, que nunca mediu esforços na dedicação e compromisso com o futuro dos seus filhos, ao meu amado pai, por ensinar a importância de uma boa educação e disciplina, a meu maravilhoso irmão com quem aprendi sobre a importância e o sabor da amizade entre irmãos e também ao meu avô e à minha avozinha (em memória), que são pessoas maravilhosas em minha vida.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais, Nelson Rodrigues Nogueira e Roseli Gomes Mariano Nogueira, que exaustiva e amorosamente lutaram a favor do meu crescimento pessoal e profissional, possibilitando mais este sucesso em minha vida. Não sei o que seria de mim sem eles.

À minha orientadora e professora, Suely Cunha Amaro Mantovani, pela sua grande contribuição em abrir meus horizontes e grande ajuda no processo de formação de novos conhecimentos.

Agradeço também aos professores Carlos Roberto Minussi, Ruben A. Romero Lázaro, José Roberto Sanches Mantovani e Alexandre por compartilharem o grande conhecimento que alcancei.

A toda equipe do Departamento de Engenharia Elétrica e também à Seção de Pós-Graduação, em especial à Márcia.

Aos amigos do trabalho na Petrobras, que me encorajaram nesta empreitada. Em especial, ao Hildebrando Pinho Filho que está sempre disposto a dar a sua visão experiente e ao Marcelo Santana Malta, pessoa que admiro muito, pela sabedoria em lidar com pessoas e as valorizar e por conduzir o ambiente de trabalho com muita propriedade.

Agradeço também às minhas queridas primas Kenia Nogueira da Silva, Kelly Nogueira da Silva e Keila Rodrigues dos Santos e tia Maura Rodrigues Nogueira por ofertarem um carinho muito especial a mim, trazendo motivação para a vida.

Sou eternamente grato também a alguns professores, que desde o início de minha vida escolar tiveram pulso firme na sua forma de ensinar e educar. Em especial à professora Joce Yeda, à professora Bela, professora Luzia, professor Jaime, professora Denise e professor Zezinho.

À CAPES pelo auxílio financeiro recebido, que permitiu que eu pudesse desenvolver esta pesquisa.

RESUMO

A proposta deste trabalho consiste no desenvolvimento de um sistema de navegação de um robô móvel autônomo utilizando-se de técnicas da lógica *fuzzy*, aqui desenvolvida na linguagem de descrição de hardware, VHDL, e a sua implementação em uma placa DE2 do fabricante Altera, usando o software de desenvolvimento, Quartus II, do mesmo fabricante. O objetivo do sistema de navegação proposto é o de que o protótipo do robô caminhe sem colidir em nenhum obstáculo (ande para frente, para esquerda, para direita e pare) usando para comunicação externa com o ambiente, três sensores de distância localizados um em cada lado do robô e um na frente, atuando através dessas informações em dois motores de passo. Os resultados desta implementação obtidos em simulação são satisfatórios e foram comparados aos resultados obtidos como o mesmo processo no MATLAB.

Palavras Chave: Lógica *Fuzzy.* Robótica móvel. Controlador inteligente. Sensores de distância. Linguagem de descrição de hardware. VHDL. Dispositivos lógicos programáveis.

ABSTRACT

The purpose of this work is to develop a navigation system of an autonomous mobile robot using *fuzzy* logic techniques, developed here in the hardware description language, VHDL, and its implementation on a DE2 – Altera board manufacturer, using the software development Quartus II, from the same manufacturer. The navigation system proposed in this work aims that the prototype robot rides without bumping into any obstacles (goes forward, to the left, to the right and stop) using for communication with the external environment three distance sensors, located one on each side of the robot and one at the front and acting through these information in two stepper motors. Simulation results obtained in the implementation developed here are reasonable compared to the same process done in MATLAB.

Keywods: Fuzzy Logic. Mobile robotics. Intelligent control. Distance sensors. Hardware description language. VHDL. Programmable logic devices.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A inteligência artificial vem tomando conta do mundo moderno nos mais diversificados meios. Não só na área de entretenimento, onde os jogos e celulares têm utilizado tal tecnologia, mas também em áreas como geração de energia, que conta com a técnica para a previsão de demanda por exemplo, na área de segurança (reconhecimento de face) entre muitas outras aplicações.

Estudos de meados de 1950 tornaram a Inteligência Artificial, até então teoria e imaginação, realidade prática. Hoje as pesquisas no assunto são desafiadoras e contribuem com a sociedade pelo desenvolvimento de aparelhos dotados de capacidade de aprender ou tomar atitudes sob condições incertas. A natureza é uma fonte de inspiração para o desenvolvimento das variadas técnicas e artifícios que possibilitam aos equipamentos eletrônicos a capacidade de tomar ações baseadas em algum tipo de inteligência. Pode-se citar, por exemplo o algoritmo genético cujo funcionamento é uma cópia do que acontece na reprodução dos animais e também as Redes Neurais, que procura implementar as redes de neurônios que formam os cérebros. Máquinas que aprendem (LM - Learning Machines) saíram do mundo da ficção e se tornaram realidade cada vez mais presente e evoluída contando com técnicas que são aprimoradas a cada dia.

A dificuldade em definir ou mesmo entender com objetividade o que é a inteligência abre espaço para discussões profundas e filosóficas muito interessantes. As observações sobre as capacidades da mente humana mostram uma série de propriedades as quais podemos chamar de elementos da inteligência, podendo ser enumerados alguns como: a habilidade de planejar, de raciocinar e de tomar decisões corretas mesmo sob condições imprecisas.

Um fato muito importante que vem tornando a inteligência artificial cada vez mais efetiva e promissora é o uso de *hardware* flexível, isto é, capaz de se modificar. Esta tecnologia alimenta a imaginação dos pesquisadores, que buscam aprimorar as técnicas sobre *Hardware* Evolutivo.

Neste trabalho descreve-se o projeto de aplicação de uma das técnicas da inteligência artificial, a lógica *fuzzy* – também chamada de lógica nebulosa ou difusa –, para um sistema de controle de um robô móvel autônomo com propósito de perambular sem colidir com obstáculos ou paredes, virar para esquerda, virar para a direita, seguir em frente ou parar. O controlador *fuzzy* é projetado especificamente em uma linguagem de descrição de hardware, VHDL (*Very High Speed Integrated Circuit*) *Hardware Description Language* para um robô que tem três sensores que medem a distância a um objeto, e dois motores de passo cujo controle é implementado em *hardware* usando a lógica difusa e os dispositivos lógicos programáveis- FPGAS, da placa DE2 - desenvolvimento educacional e o software Quartus II da Altera.

1.1 Motivação

A navegação de robôs móveis é um tema bastante explorado pela comunidade científica. Criar um robô capaz de tomar algumas decisões em um ambiente desconhecido é uma tarefa importante para a substituição do homem pelo robô, em atividades de risco como salvamento de pessoas em ambientes arriscados, como por exemplo, lugares em chamas, cavernas; aplicações industriais, aplicações médicas, dentre outras. A complexidade do desenvolvimento dos sistemas que integram o robô, classificado como um sistema dinâmico, o domínio de suas técnicas e as limitações das tecnologias disponíveis motivou o desenvolvimento deste trabalho.

1.2 Organização do Trabalho

A estrutura deste trabalho apresenta, além desta introdução, no capítulo 2, uma revisão bibliográfica sobre alguns dos principais artigos que tratam de lógica *fuzzy* aplicada a robôs e ao problema proposto. Apresentam-se no capítulo 3, o histórico e os conceitos envolvendo a Lógica *fuzzy*. No capítulo 4, descrevem-se a plataforma utilizada no projeto e os demais dispositivos. O desenvolvimento do sistema de navegação do robô é apresentado no capítulo 5, onde são mostrados os blocos dos circuitos escritos na linguagem VHDL que realizam a proposta do projeto e suas simulações. Por fim, as conclusões no capítulo 6, seguido das referências.

2 REVISÃO BIBLIOGRÁFICA

Os robôs móveis autônomos e o domínio de suas técnicas são alvo de muitas pesquisas por conta da enorme contribuição que estes podem oferecer aos homens.

Importantes artigos propõem as mais variadas técnicas para o desenvolvimento de sistemas de navegação de robôs, fornecendo subsídios para outros trabalhos cujos objetivos são os de alcançar sistemas eficientes para atividades específicas dos robôs. Existem dezenas de técnicas diferentes propostas para o desenvolvimento de sistemas de navegação de robôs móveis autônomos, e alguns pesquisadores mesclam técnicas criando sistemas híbridos. A revisão bibliográfica apresentada a seguir foi baseada na leitura destes artigos visando obter subsídios para o desenvolvimento desta dissertação.

Em Shindo (1999), é descrito um sistema de controle realizado através de um hardware e software flexíveis. Neste artigo são empregados algoritmos genéticos como ferramenta do processo evolucionário e uma estrutura paralela para o processo de evolução mais eficiente. Assim, através das tecnologias de dispositivos reconfiguráveis é mostrada no trabalho, a possibilidade de reprogramação intrínseca (todo o processo realizado no próprio hardware). O trabalho de Shindo está baseado no modelo do robô Khepera. Este robô possui oito sensores de distância codificados em valores de 3 bits e possui também dois motores independentes, cujas direções de rotações obedecem à codificação de 1 bit para cada motor. O robô é posto em um labirinto simples 8X8 o que leva a uma função objetivo de valor 64.

A análise dos resultados apresentada no texto do artigo, revela que após a quarta geração, é gerado um circuito que atende à função objetivo, adquirindo porém, a estabilidade após a décima segunda geração. Os autores fazem suas observações e dividem as atitudes do robô em cinco estados além de concluírem que apenas sete entradas das dezesseis foram utilizadas no circuito final. Segundo eles, o uso da técnica proposta é possível.

Sandi L. et al. (1998) já tratava a questão de um sistema para navegação e guiagem de robôs móveis autônomos utilizando para a navegação, integração sensorial via filtro de Kalman de dois subsistemas independentes: o *dead-reckoning*, que utiliza a leitura de pulsos de dois *encoders* alocados nas rodas direita e esquerda do veículo e uma bússola digital. Um controlador *fuzzy* utiliza a informação de posição e atitude do veículo para o sistema de guiagem, que é de sintonização simples e prescinde do conhecimento do modelo dinâmico do veículo. O sistema completo é implementado em tempo real, apresentando bom desempenho. Este é caracterizado por dois modos de visualização do controle dos robôs, um a navegação que designa a determinação de posição e orientação do veículo em um dado instante de tempo e outro a guiagem, que se refere ao controle da trajetória.

O *dead-reckoning* é um método clássico de navegação, cuja precisão depende diretamente da qualidade dos sensores utilizados, sendo inevitável o acúmulo de erro de posição, o que requer integração com outros tipos de sensores, de modo a compensar este erro. Alguns exemplos podem ser vistos em Fuke e Krotkov (1996), Murata e Hirose (1993) e Lages et al. (1996) referidos em Vuong (2006), para solucionar este erro.

Outro artigo importante nesta área é o de Costa et al. (2003) onde é descrito o projeto e a construção de um mini robô autônomo capaz de seguir um caminho definido na superfície sobre a qual se move a partir de imagens capturadas. Seguir pistas é um comportamento simples, mas importante para uso de robôs móveis em ambientes estruturados como nas indústrias. O robô construído tem tamanho padrão definido por campeonatos oficiais de robôs, consistindo de dois motores de passo, módulo de potência e uma câmera para obtenção de imagens montados sobre uma base metálica. O robô envia as imagens para um microcomputador e recebe os sinais de comando dos motores e de um hardware específico para seu controle implementado em um FPGA - Altera. No microcomputador é implementado uma arquitetura que possui o módulo de aquisição de imagens usando a placa de captura padrão BT-878, que recebe as imagens e as digitaliza , um módulo de identificação da pista , que a partir da imagem detecta o centro da pista que o robô deve seguir e o módulo de controle, que implementa um comportamento reativo - seguir uma linha baseada nas informações visuais recebidas e enviar as informações para

o hardware de controle do motor através de uma porta paralela. Neste artigo, os resultados dos testes relatados indicam a eficiência do projeto, onde o robô é capaz de seguir qualquer caminho desenhado por um usuário. Apesar de não serem usados obstáculos neste trabalho e sim pistas pré-definidas, o trabalho traz ideias de arquitetura e funcionamento dos módulos.

Um controlador lógico *fuzzy* implementado em FPGA é descrito em Daijin, (2000). Este controlador é dividido em muitos módulos funcionais independentes no tempo e cada módulo é implementado individualmente em um projeto automatizado. O sistema de desenvolvimento deve possibilitar a edição em uma linguagem de descrição de hardware, compilação, simulação, otimização do projeto e finalmente programação da placa contendo o dispositivo FPGA. Segundo o artigo, controladores *fuzzy* têm sido aplicados tanto em produtos ao consumidor como em processos industriais. Este tipo de controlador é muito efetivo e particularmente aplicado em processos imprecisos e complicados, para os quais não existe um modelo matemático ou o modelo matemático é severamente não linear.

Em Cabrera et al. (2003), é descrito uma arquitetura de baixo custo e de alto desempenho em termos de velocidade para implementar hardware de inferência difuso ao qual está baseado em um processador de regras ativas, à limitação do grau de envolvimento das funções de pertinência das entradas e à utilização de métodos de defuzificação simplificados. Neste artigo descreve-se o desenvolvimento de sistemas difusos através do *Xfuzzy*, ferramenta de CAD para lógica *fuzzy* com ênfase na ferramenta XFVHDL na qual permite a geração do código para os diferentes elementos da arquitetura descrita.

Vuong (2006) oferece uma grande ajuda para a programação *fuzzy* voltada para o VHDL. Propondo a implementação de um controlador *fuzzy* genérico em hardware reconfiguráveis, os autores desenvolvem seu trabalho apresentando uma explicação detalhada da programação de um controlador *fuzzy* simples. O texto é um importante material de referência para motivar iniciantes na tarefa de criar códigos VHDL para controladores *fuzzy*.

Nesta dissertação de mestrado optou-se por usar a lógica *fuzzy*. O controlador *fuzzy* implementado neste artigo para guiagem do robô, dispensa o modelo dinâmico do veículo necessitando apenas que os comandos de controle obtidos com base na experiência heurística do operador humano sejam expressos em variáveis linguísticas e regras de produção do tipo IF/THEN. Assim, para converter as variáveis numéricas de entrada do controlador em variáveis linguísticas é necessário um processamento denominado fuzificação. De igual modo, para transformar um comando de controle fornecido como variável linguística em valores numéricos que possam ser interpretados pelo atuador do sistema, é necessário outro processamento denominado defuzificação. Para simplificar o modelo *fuzzy* do controlador, supõe-se que o veículo não se movimenta para trás, ou seja, move-se sempre para frente e para os lados e considera-se que o controle de velocidade melhora o desempenho do sistema de posição.

3 LÓGICA *FUZZY*

A lógica *fuzzy,* também denominada lógica nebulosa ou difusa, é uma teoria que incorpora a experiência, a intuição, o conhecimento especialista e a natureza imprecisa do processo decisório humano através de um conjunto de regras ou heurísticas simples. Para o entendimento do procedimento adotado neste trabalho, faz-se uma introdução aos principais conceitos que envolvem a lógica *fuzzy*.

3.1 Histórico

Teorias como a teoria clássica dos conjuntos e a teoria das probabilidades, embora úteis, nem sempre conseguem captar a riqueza da informação fornecida por seres humanos. A lógica fuzzy tem como base a capacidade de raciocínio aproximado mostrando-se muito mais eficiente para uma grande variedade de problemas para os quais é difícil precisar informações.

Em 1965 Lotfi A. Zadeh (1965), engenheiro eletrônico, professor de Teoria dos Sistemas na Universidade da Califórnia – Berkeley, publicou o primeiro artigo sobre a teoria dos conjuntos nebulosos para tratar as incertezas não probabilísticas, o que foi denominado *Fuzzy Sets*. Este trabalho contribuiu para o desenvolvimento da Lógica *Fuzzy* que, diferentemente da lógica tradicional, supera a ideia de um elemento pertencer ou não a um determinado conjunto, permitindo que um elemento pertença a um conjunto com certo grau de pertinência. Ele tinha como objetivo fornecer ferramentas matemáticas capazes de lidar com o raciocínio lógico que contemplassem aspectos imprecisos e ambíguos, tipo de raciocínio não passível de processamento na lógica computacional fundamentada na lógica de *Boole.*

A teoria de Zadeh por ser menos restritiva, pode ser considerada mais adequada para o tratamento de informações fornecidas por seres humanos do que a teoria de probabilidades. Tais teorias têm sido cada vez mais usadas em sistemas que utilizam

informações fornecidas por seres humanos para automatização de processos.

Os primeiros trabalhos responsáveis pelo surgimento da lógica *fuzzy* tinham como objetivo tornar as máquinas capazes de "pensar", ou tomar decisões e raciocinar sob informações vagas, imprecisas, inexatas e ambíguas, como os humanos. A lógica *fuzzy* traz às máquinas este potencial, antes exclusividade humana.

Os sistemas que exigem altíssimos níveis de precisão e exatidão consomem muito tempo e tem alto custo. Além do mais, sistemas muito complexos são de caracterização imprecisas e inexatas. Zadeh afirma que em muitos problemas, para um aperfeiçoamento do trabalho, é necessário que se aceite informações imprecisas em certo nível. A teoria da lógica *fuzzy* oferece então, subsídios para sistemas convencionais cuja complexidade não é significante. Soluções preliminares e aproximadas são rápidas e muitas vezes estão dentro do esperado, o que justifica a preferência por tal técnica, que trata de forma mais simples os sistemas de modelagem extremamente difícil por apresentarem características não lineares.

São dois os tipos de sistemas em que a lógica *fuzzy* pode ser aplicada:- problemas muito complexos com comportamentos difíceis de serem compreendidos e problemas onde resultados aproximados são aceitáveis. Nas aplicações da lógica *fuzzy* destacam-se o controle de processos, a aproximação funcional, o apoio à decisão, etc.

Em aplicações complexas, a tradução de informação imprecisa utilizando a teoria tradicional é inviabilizada em razão da complexidade matemática que poderia resultar. Entretanto, a teoria dos conjuntos *fuzzy* proporciona grande facilidade para descrever e processar esse tipo de informação através de variáveis linguísticas e de uma base de conhecimento *fuzzy* representada por um conjunto de regras.

Dentre as vantagens da utilização da lógica *fuzzy* tem-se o mecanismo de raciocínio similar ao do ser humano, por meio do uso de termos linguísticos; modelagem de conhecimento de senso comum; conhecimento ambíguo e conhecimento impreciso, mas racional; técnica de aproximação universal; robustez e tolerância à falha; além do baixo custo de desenvolvimento e de manutenção. As limitações estão relacionadas à geração das

regras *fuzzy* e à definição das funções de pertinência; ambas, baseadas em uma avaliação subjetiva do conhecimento do especialista. Adiciona-se a isso a inexistência de técnicas de aprendizado.

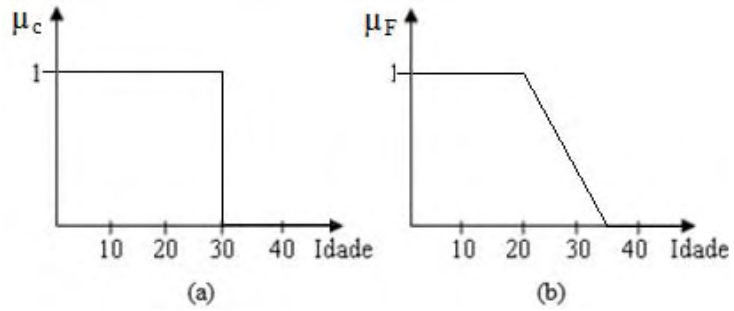
As primeiras aplicações industriais utilizando a lógica *fuzzy* começaram na década de 1970. Em 1980 apareceram as primeiras aplicações em engenharia de controle por algumas companhias japonesas. Para aplicação da teoria da lógica *fuzzy*, tem-se os seguintes conceitos: conjuntos *fuzzy*, funções de pertinência, as variáveis linguísticas, fuzificação e os modelos de inferência.

3.2 Conjuntos *fuzzy*

Um conjunto *fuzzy F* de um universo de discurso U é representado comumente por uma função de pertinência real mapeada por $\mu_F : U \rightarrow [0,1]$, a qual associa cada elemento $x \in U$ a um número real $\mu_F(x)$ pertencente ao intervalo fechado [0,1], o qual representa o grau de pertinência de *x* em *F* (REZENDE, 2005).

Na Figura 1 são mostradas duas funções de pertinência para o conjunto de indivíduos jovens. Na Figura 1 (a) está representada a função de pertinência para o conjunto clássico também denominado conjunto *crisp* e na Figura 1 (b) representa-se a função de pertinência para o conjunto *fuzzy*. Essas funções são definidas analiticamente pelas expressões (3.1) e (3.2), respectivamente. Observa-se que no conjunto *fuzzy* a transição entre o indivíduo totalmente jovem ($\mu_F = 1$) e o indivíduo não jovem ($\mu_F = 0$) ocorre de forma suave; ao contrário do conjunto crisp no qual tal transição ocorre de forma abrupta, descontínua.

Figura 1- Funções de pertinência para indivíduos jovens. (a) Conjunto crisp. (b) Conjunto *fuzzy*.



Fonte: (FARIA *et al,* 2012)

$$\mu_C = \begin{cases} 1, & se\ Idade \leq 30 \\ 0, & se\ Idade > 30 \end{cases} \quad (1)$$

$$\mu_F = \begin{cases} 1, & se\ Idade < 20 \\ \dfrac{35 - Idade}{15}, & se\ 20 \leq Idade \leq 35 \\ 0, & se\ Idade > 35 \end{cases} \quad (2)$$

3.2.1 Operações Básicas entre Conjuntos *Fuzzy*

As operações básicas entre conjuntos *fuzzy* propostas por Zadeh são a de complemento, união e interseção.

- **Complemento**: o complemento de um conjunto *fuzzy A* possui função de pertinência dada por:

$$\mu_{\bar{A}} = 1 - \mu_A(x) \quad (3)$$

O operador complemento corresponde ao conectivo "NÃO".

- **União**: a união de dois conjuntos fuzzy A e B podem ser representadas por $A \cup B$ ou por $A + B$. A união entre esses conjuntos fuzzy possui função de pertinência definida por:

$$\mu_{AUB} = \max[\mu_A(x_i), \mu_B(x_i)] \quad (4)$$

O operador união corresponde ao conectivo “OU”.

- **Interseção**: a interseção entre os conjuntos *fuzzy* A e B pode ser representada por $A \bigcap B$ ou por $A \cdot B$. Tal operação resulta na função de pertinência dada por:

$$\mu_{A \cap B} = \min[\mu_A(x_i), \mu_B(x_i)] \quad (5)$$

A interseção corresponde ao conectivo “E”.

As definições apresentadas para a união e para a interseção de conjuntos *fuzzy* são particulares e foram propostas por Zadeh.

3.3 Funções de pertinência

As funções de pertinência são funções contínuas. Podem ter diferentes formas, no entanto, as mais comuns são as funções triangulares, trapezoidais, gaussianas e exponenciais (MINUSSI, 2009).

Nos conjuntos *fuzzy* a pertinência é gradual, ao contrário do que ocorrem com os conjuntos da teoria de conjuntos clássica denominada conjuntos *crisp*. Nesses a pertinência assume apenas dois estados: compatível ou incompatível.

3.4 Variáveis linguísticas

A base de regras junto com a base de dados constitui a base de conhecimento do sistema *fuzzy*. Os conjuntos *fuzzy* podem ser usados para construir conjuntos de termos linguísticos. Os conjuntos de termos representam abstrações dos valores da variável, isto é, são uma partição *fuzzy* de seus possíveis valores. Em geral, uma variável linguística é associada a um conjunto de termos, no qual cada termo é definido no mesmo universo de discurso. A partição *fuzzy* determina quantos termos existirão no conjunto.

Define-se uma variável linguística como uma entidade utilizada para representar de

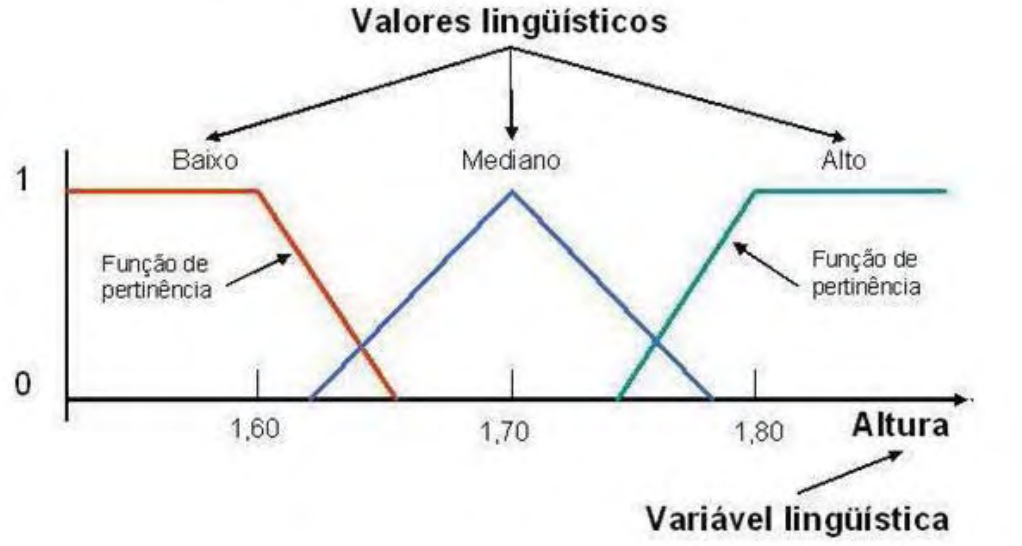
modo impreciso, através da linguagem cotidiana, um conceito ou uma variável de um dado problema (REZENDE, 2005). Em geral, a cada variável linguística, estão associadas expressões linguísticas. Tais expressões linguísticas qualificam as variáveis em termos linguísticos, portanto, de forma imprecisa. A capacidade de qualificar as variáveis de um problema de modo impreciso, em termos qualitativos em vez de quantitativos, fornece uma ideia do que é uma variável linguística.

A altura de uma pessoa é um exemplo de variável linguística. Desta forma, esta variável pode ser expressa através de números obtidos de uma medição com uma trena ou de valores subjetivos não precisos, tais como “baixo”, “mediano”, “alto”, por exemplo.

Esta variável é composta pelo nome (no caso do exemplo, a altura); pelos valores linguísticos (“baixo”, “mediano” e “alto”), que são os conjuntos nebulosos; pelo universo de discurso; e pelas funções de pertinência que associa um grau de pertinência a cada elemento do universo de discurso.

Um exemplo de aplicação da lógica *fuzzy*, a classificação da altura de uma pessoa, é mostrado na Figura 2. Observa-se nesta figura a relação de altura (eixo horizontal) com o grau de pertinência (eixo vertical). Com este exemplo podemos concluir que uma pessoa com 1,70m é classificada como de altura média com grau de pertinência 1, e uma pessoa com 1,65m aproximadamente seria classificada como altura média com grau de pertinência de 0,5.

Figura 2 - Exemplo de uma variável *Fuzzy* (altura de uma pessoa) com seus conjuntos (baixo, mediao e alto).



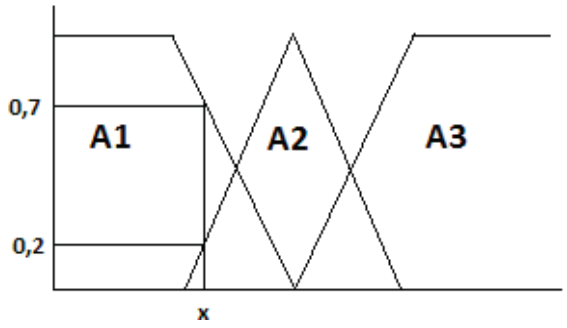
Fonte: (SHHEIBIA, 2001)

A caracterização das variáveis linguísticas pode ser tanto qualitativa, quando expressas pelo valor subjetivo e vago, quanto quantitativa, quando expressas pelo valor de pertinência.

3.5 Fuzificação

Fuzificação é a codificação das entradas em graus de pertinência μ, para cada conjunto *fuzzy*. Essa codificação é baseada no conhecimento do especialista. Tomando-se o exemplo da Figura 2, onde a variável *fuzzy* é a altura de uma pessoa e chamando A_1=baixo, A_2 =mediano, A_3 =alto, reapresenta-se a Figura 2, na forma mostrada na Figura 3. Desta figura, tem-se $\mu(x = A_1) = 0{,}7; \mu(x = A_2) = 0{,}2; \mu(x = A_3) = 0$.

Figura 3 - Variável *Fuzzy*, altura de uma pessoa e seus conjuntos (baixo, mediano e alto).



Fonte : Própria do autor

3.6 Regras *fuzzy*

As regras *fuzzy* fornecem uma descrição qualitativa do sistema em estudo. O conhecimento em um Sistema de Inferência *Fuzzy* (SIF) é armazenado em geral na forma de regras composta pelos termos antecedente e o consequente, **SE** <antecedente> **ENTÃO** <consequente> e também das operações nebulosas, como União, Interseção, entre outras (MINUSSI, 2009).

O antecedente é composto por um conjunto de condições envolvendo variáveis *fuzzy* e expressões linguísticas que, quando satisfeitas, mesmo que parcialmente, determinam o processamento da parte consequente da regra através de um mecanismo de inferência *fuzzy*. Tal processo é chamado disparo da regra.

O consequente é composto por um conjunto de ações que são geradas com o disparo da regra. Os consequentes de todas as regras disparadas são processados em conjunto, para gerar uma resposta determinística para cada variável de saída do Sistema de Inferência *Fuzzy*. No SIF é importante que existam tantas regras quantas forem necessárias para mapear totalmente as combinações dos termos das variáveis, formando uma base de regras completa.

As regras de controle são baseadas no conhecimento e expectativa do projetista sendo que cada uma delas demanda uma ação de controle. A ordem em que são dispostas as regras, não afeta o resultado, pois elas são declarativas e não sequenciais.

O número total de regras em um sistema de controle difuso depende do número de entradas do controlador e do número de conjuntos difuso de cada entrada. Se em um controlador tem-se *p* entradas com a mesma quantidade de conjuntos difusos *n*, o número total de regras, R é dado (CABRERA et al., 2003) por:

$$R = n^p \tag{6}$$

Portanto, se um sistema tem duas variáveis de entrada e cada uma delas tem três conjuntos difusos, a quantidade de combinações de entrada é de nove regras de controle.

O processo de inferência propriamente dito é o responsável pela aplicação das regras apoiando-se em declarações do tipo **Se-Então** e operações nebulosas. Nesta etapa ocorre a tomada de decisões. Por conta da simplicidade, optou-se neste trabalho por utilizar o método de inferência MAMDANI.

A literatura classifica os controladores *fuzzy* considerando as características do método de tomada de decisão que eles empregam. Apesar dos muitos métodos citados e apresentados na literatura, Sugeno (1985) os separa em dois grandes grupos: os chamados controladores do tipo MAMDANI e os controladores do tipo SUGENO (SUGENO, 1974).

Os controladores do tipo MAMDANI são frutos do trabalho publicado em 1973 por Mamdani onde é definido o algoritmo destes sistemas. Esses controladores convertem os valores quantitativos em qualitativos (*fuzzy*) e após isso, através de inferência, em outros valores ainda qualitativos, sendo necessário o papel do defuzificador para a resposta final quantitativa.

De fácil modelagem por basear-se na intuição, os controladores MAMDANI (MAMDANI, 1973) são bons quando um controle grosseiro é aceitável. Para controle mais fino, o controlador SUGENO apresenta desempenho superior, porém a modelagem tem o prejuízo de não ser mais intuitiva e sim matemática.

Nos controladores SUGENO, o resultado da inferência de suas regras é dado por um valor numérico através de funções das variáveis linguísticas de entrada, isto é, cada regra conduz a consequências que são funções das variáveis nebulosas de entrada, em forma matemática tem-se:

Se x é B, então y é C -> *tipo Mamdani*

Se x é B, então y e f(x) -> *tipo Sugeno*

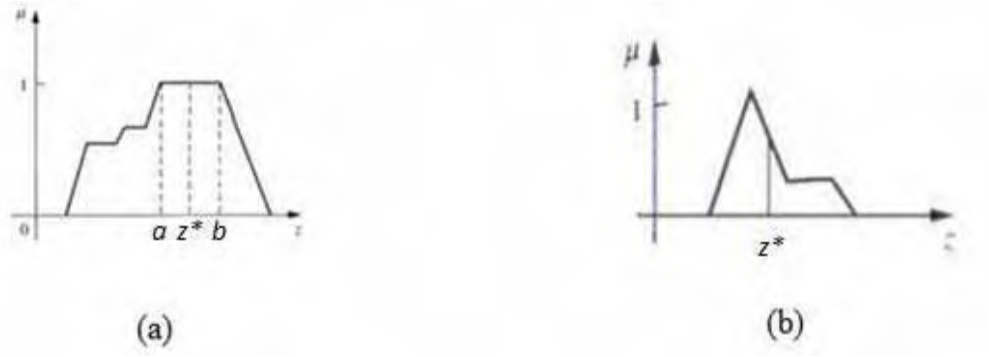
A resposta de um controlador do tipo SUGENO é dada pela média ponderada das respostas das regras não sendo necessário o defuzificador.

3.7 Defuzificação

A conversão de um conjunto *fuzzy* para um valor escalar transforma informações qualitativas em informações quantitativas. Tal processo é dito defuzificação. Os métodos de defuzificação mais comuns são o método da média dos máximos (Figura 4a) e o método do centroide (Figura 4b).

O método do centroide expresso pela equação (7) calcula, para um conjunto *fuzzy* de saída, a abscissa (no universo de discurso definido para a variável de saída em questão) do ponto do centro de massa correspondente.

Figura 4 - (a) Método: média dos máximos. (b) Método: centroide



Fonte: (MINUSSI , 2009)

$$\hat{y}2 = \frac{\sum_{y \in U_{y2}} y.\mu_{B_i'}(y)}{\sum_{y \in U_{y2}} \mu_{B_i'}(y)} \tag{7}$$

No método da média dos máximos, conforme expressão (8), o valor numérico da saída corresponde ao ponto do universo de discurso que corresponde à média dos pontos de máximo locais da função de pertinência do conjunto de saída produzida pelo processo de inferência.

$$\bar{y}2 = \frac{\sum_{\hat{y} \in U_{y2}} \hat{y}_k.\mu_{B_i'}(\hat{y}_k)}{n_{\hat{y}}};\ onde\ \hat{y} = \max_{y \in U', U' \subset U_{y2}} \left[\mu_{B_i'}(y)\right] \tag{8}$$

Cada método fornece respostas diferentes, sendo necessário então, que se escolha um método mais adequado visando a aplicação. Outro fator que acarreta em diferenças na resposta do sistema é a inclinação das retas das funções e as áreas onde há superposição destas funções.

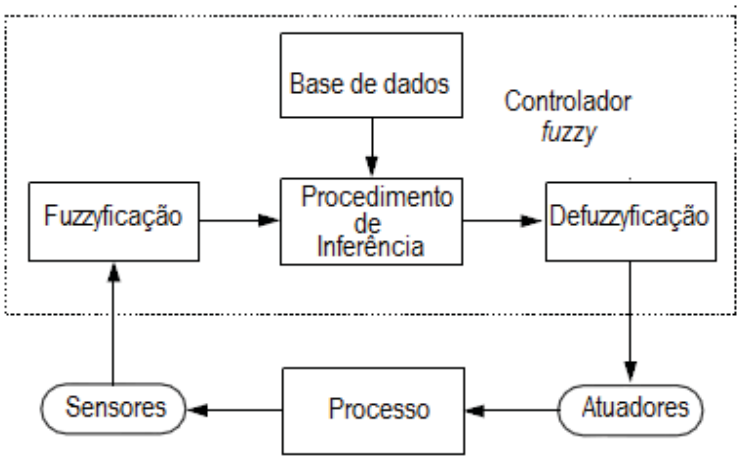
Conceitos como níveis de velocidade (lento, rápido), de temperatura (quente, morno, frio), de distância (perto, longe), podem parecer de fácil classificação para nós humanos, porém dentro dessa classificação existem incertezas, como por exemplo, a

temperatura classificada como quente para uma pessoa em uma região fria, pode ser uma temperatura normal para uma pessoa que mora em uma região que é normalmente quente. Por isso, podem-se estabelecer valores fixos para esses níveis, por exemplo, a partir de 19º C seria considerado como estado quente.

3.8 Estruturas básicas para um controlador *fuzzy*

A estrutura básica para um controlador *fuzzy* pode ser visualizada pelo diagrama de blocos da Figura 5.

Figura 5 - Estrutura básica de um controlador tipo *fuzzy*



Fonte : Adaptado de Gomide (1994).

Nesta figura observa-se que o controlador *fuzzy* consiste de quatro principais unidades:

- Fuzificador que converte uma entrada crisp em um conjunto termos *Fuzzy*;
- Regras *fuzzy* (base de dados) baseadas na execução do sistema;
- Inferências *fuzzy* que executam aproximadamente raciocínio associado as variáveis de entrada com regras *fuzzy*;

- Defuzificador o qual converte as saídas do controlador *fuzzy* para um valor crisp para a entrada do sistema real sobre o alvo.

Vaz (2006) apresenta uma análise sobre a influência da escolha de diferentes funções de pertinência no caso de um controle de semáforo. Em sua dissertação, comprova-se que o desempenho do sistema é afetado quando se arbitra por funções de pertinência diferentes. De igual modo, espera-se que a quantidade de variáveis linguísticas, as definições de regras e os mecanismos de inferência e os diferentes métodos de defuzificação também tenham relevância no desempenho do sistema. Assim, sistemas que requerem um controle mais fino necessitam de maior cuidado com as escolhas e até mesmo ajustes finais sobre estes itens.

O Processador *Fuzzy* tem sido implementado em várias plataformas tais como computadores (PC), processadores (processador digital de signal (DSP), dispositivo de memória digital (LUTs- lookuptable), FPGA, etc.

A implementação mais direta consiste no uso do hardware dedicado pela relativamente simples arquitetura e algoritmo de processamento. Muitas indústrias têm desenvolvido chips controladores *Fuzzy* Lógicos usando o semicustom projeto com células padrão CMOS. Neste projeto implementa-se um controlador *Fuzzy* usando um kit da Altera –DE2 contendo um FPGA.

4 METODOLOGIA E MATERIAIS USADOS NO PROJETO

A navegação autônoma de robôs é um problema de muita complexidade, porém de uma utilidade quase imensurável visto a contribuição que um robô dotado desta capacidade pode exercer sobre a sociedade.

Para implementar controladores de robôs móveis autônomos, como já foi visto na revisão de literatura, existem muitas técnicas tais como, EHW- hardware evolutivo, que utiliza algoritmos genéticos e tem gerado muitos trabalhos; Redes Neurais das mais variadas configurações são sugeridas; técnicas mais simples como virar sempre para um dos lados até que o obstáculo não seja mais percebido; Reinforcement Learning (RL), aprendizagem baseada em recompensa, também é uma técnica que está sendo muito explorada e mesmo vários trabalhos integrando RL com a lógica *fuzzy* já foram publicados.

Particularmente para este trabalho, outras técnicas além da *fuzzy* poderiam ser utilizadas. Mesmo os controladores clássicos PID, tem sido usados. A preferência pela lógica *fuzzy*, no entanto, não é uma escolha aleatória, mas sim oriunda da simplicidade de modelagem que ela oferece por se basear na linguagem natural do homem. Sendo muito vantajosa sobre as técnicas de PID ou mesmo Redes Neurais para problemas complexos.

Este sistema tem a função de resolver o problema de um robô capaz de se mover em meio a um ambiente arbitrário evitando colisões com possíveis obstáculos. O robô interage com o ambiente levando em conta as informações que obtém de seus sensores, o que permite uma tomada de decisão, através de comandos aos atuadores (motores).

4.1 Descrição do problema

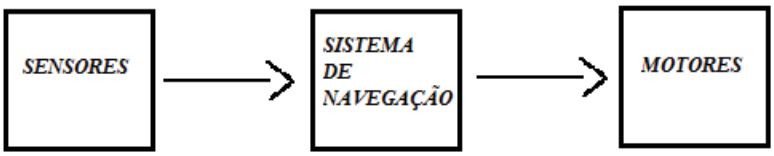
Baseado na literatura estudada propõem-se a implementação de um robô que deve mover-se para frente até que encontre um obstáculo ou uma parede, caso em que deverá desviar-se. Para tal, o robô móvel utiliza três sensores de distância da *Sharp Corporation*

(2011) dispostos na frente e nas laterais. Como não há locomoção para trás, não está sendo usado um sensor traseiro. As regras de tomada de decisão para qual atitude o robô deve tomar se baseiam nas informações fornecidas pelos três sensores a cada instante sendo tratadas essas informações por lógica *fuzzy.*

4.2 O Diagrama de blocos do sistema e o robô

Na Figura 6, mostra-se o esquema geral do trabalho, cuja representação foi resumida em três blocos básicos que são detalhados nos itens a seguir. Na Figura 7 (a) mostra-se o esquema que ilustra a disposição dos três sensores de distância (S1, S2 e S3), motores da esquerda (E) e da direita (D) e as rodas do robô, cujo protótipo, Figura 7 (b), foi confeccionado nas oficinas do laboratório de Engenharia Elétrica- FEIS - UNESP.

Figura 6 - Bloco básico de representação do sistema de navegação do robô.



Fonte: Própria do autor

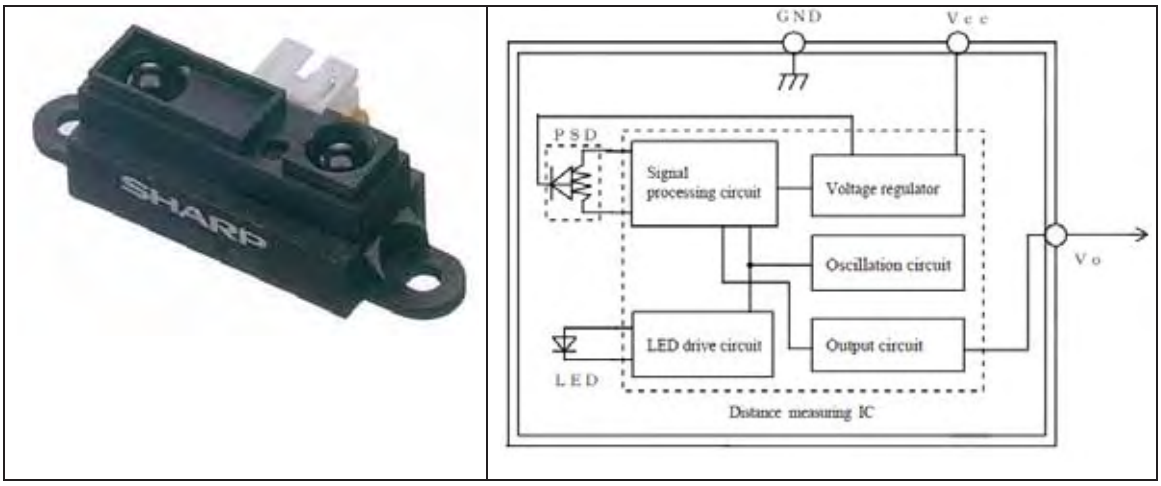
Figura 7 - (a) Ilustração da Disposição das rodas, motores e sensores do robô. (b) O protótipo do robô em uso na pesquisa

Fonte: Própria do autor

4.3 Sensores

Para a implementação da lógica *fuzzy* e a tomada de decisão quanto a posição do robô são necessárias várias medidas ao longo do trajeto do robô. Devido a isto, se optou por sensores que medem a distância para algum obstáculo que possa existir no caminho do robô. Para isso utilizam-se três sensores de distância do tipo - GP2Y0A21YK0F da SHARP, conforme pode ser observado na Figura 8(a), capaz de medir variações entre 10 e 80 centímetros com saída analógica. Este sensor é um dispositivo composto por um circuito integrado com PSD (*Position Sensitive Detector*), IRED (*Infrared Emitting Diode*) e um circuito de processamento de sinal. O diagrama de blocos deste dispositivo é apresentado na Figura 8(b).

Figura 8 -(a) Sensor de distância (10 a 80) cm- GP2Y0A21YK0F – SHARP. (b) Diagrama de Blocos do sensor da SHARP



Fonte: (SHARP, 2011)

Este sensor adota o método de triangulação, que confere ao mesmo pouca influência por conta de variações de refletividade dos objetos, temperatura ambiente e tempo de operação. Conforme consta no manual do fabricante, é recomendado que estes sensores sejam posicionados na vertical de forma a ocasionar menos erros de medição. Sua saída é analógica por isso, para o processamento por dispositivos lógicos programáveis, como é o caso, há necessidade de um conversor analógico digital.

Apresentam-se na tabela 1, os testes realizados para calibração de cada um dos sensores de distância e o escalonamento em binário, para uma resolução de oito bits com referência de tensão de 3 V. Na linha da tabela destacada em negrito observa-se que o funcionamento do sensor em uma região estável é obtido no intervalo entre 10 e 80 cm.

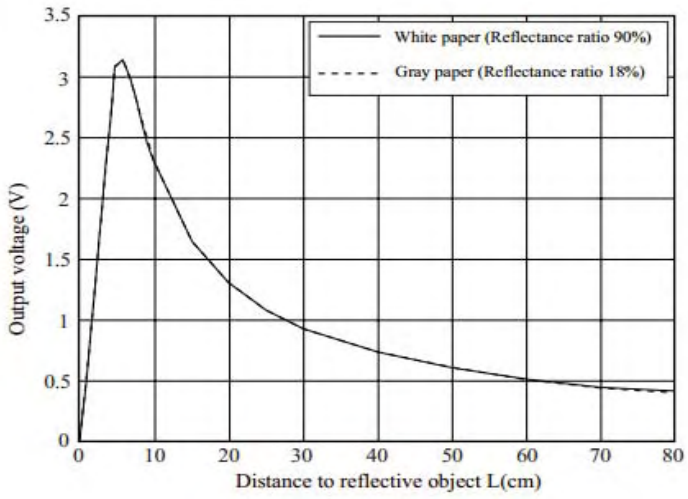
Tabela 1- Testes para os três sensores de distância da SHARP.

	Sensor Direita		**Sensor Esquerda**		**Sensor Central**		**Média dos Sensores**	**Degrau de referência**	**Equivalente Binário (8 bits) (arredondado)**
Distância (cm)	Saida (v)		Saida (v)		Saida (v)		Saida (v)		
0	**0,14**		**0,14**		**0,16**		**0,15**	**12,47**	**00001101**
10	2,33		2,35		2,36		2,35	199,47	11001000
20	1,25		1,29		1,27		1,27	107,95	01101100
30	0,84		0,87		0,84		0,85	72,25	01001001
40	0,61		0,63		0,6		0,61	52,13	00110101
50	0,45		0,47		0,42		0,45	37,97	00100110
60	0,30		0,33		0,30		0,31	26,35	00011011
70	0,18		0,23		0,18		0,20	16,72	00010001
80	0,07		0,11		0,09		0,09	7,65	00001000

Fonte: Própria do autor

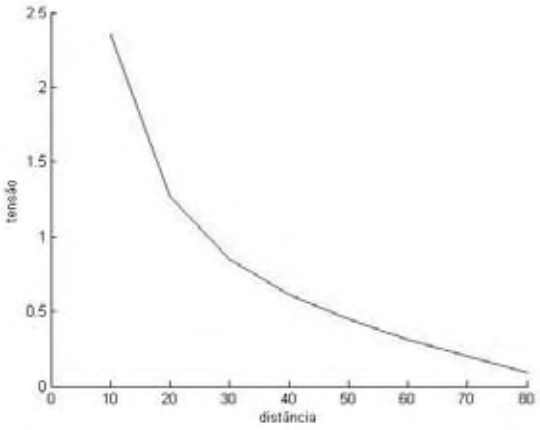
Para os testes, o sensor fixado como mostrado anteriormente (Figura 7) no robô, foi posicionado de a forma a ficar na marca zero da escala graduada e alimentado com uma tensão DC de 5,0 V. Foi verificada a tensão, obtida na saída de cada sensor, em função da distância, de 10 em 10 cm entre eles e um anteparo de cor branca, na faixa de 0 a 80 cm. O gráfico com a relação de tensão por distância ao objeto, dado pelo fabricante, pode ser visto na Figura 9a. Com o auxílio dos valores médios de tensão da tabela 1, e o software MATLAB, obtém-se o gráfico da Figura 9b. Nota-se através da análise desta figura que a curva obtida no intervalo de 10 a 80 cm de distância é semelhante à curva dada no manual do fabricante do sensor.

Figura 9 - (a) Tensão de saída X distância ao objeto dado pelo fabricante. (b) Tensão de saída X distância, obtido experimentalmente para um sensor GP2Y0A21YK0F - SHARP.



(a)

Fonte: Sharp Corporation (2011)



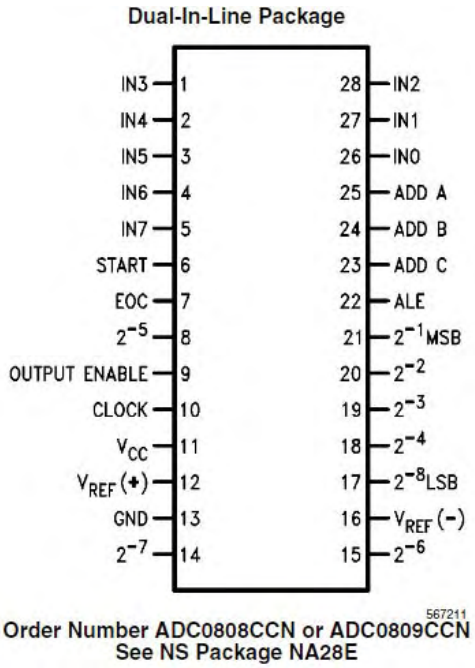
(b)

Fonte: Própria do autor

4.4 Conversor A/D

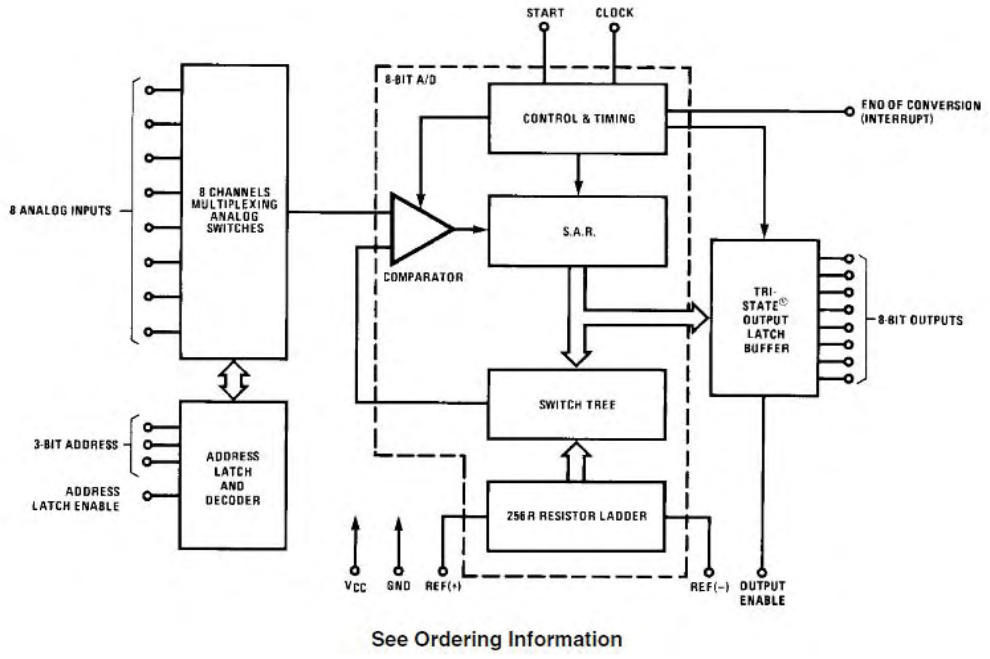
Para o tratamento do sinal dos sensores pela placa DE2 da Altera e usando a interface de expansão paralela de 40 pinos é necessária a conversão dos dados da forma analógica para digital. Optou-se então, por usar o conversor analógico digital ADC0808 da *National Semiconductor.* Este componente é fabricado com a tecnologia CMOS monolítico e possui oito canais para aquisição de dados multiplexados, destes, usam-se três canais, um para cada sensor. Na Figura 10 mostra-se a pinagem ou diagrama de conexões e na Figura 11, o diagrama de blocos, conforme consta do manual do fabricante.

Figura 10 - Diagrama de Conexões fornecido pelo fabricante.



Fonte: Encontrado no Manual da *National Semiconductor* (2011).

Figura 11 - Diagrama de Blocos do ADC 0808



Fonte: Encontrado no Manual da National Semiconductor (2012).

Para o correto funcionamento do conversor A/D deve-se atentar para as entradas de **START**- inicio de conversão, **ALE**- *Adress Latch Enable*, habilitando o latch de endereços para a seleção dos canais e a saída **EOC** – *End Of Conversion*, que indica o final de conversão.

4.5 Motores

Neste trabalho usam-se dois motores de corrente contínua (CC) para o acionamento das duas rodas conforme visto na Figura 7. É um dos motores mais usados em sistemas robóticos por oferecer torque razoável. Por isso, descreve-se a seguir os conceitos e características principais dos motores de corrente contínua e seu acionamento.

4.5.1 Principais Características

Um dos fatores da crescente utilização de motores CC é o fato de que o controle de

velocidade deste motor é bastante simples, necessitando somente variar a tensão de alimentação. Além disso, os motores CC apresentam torque constante em toda a faixa de velocidade.

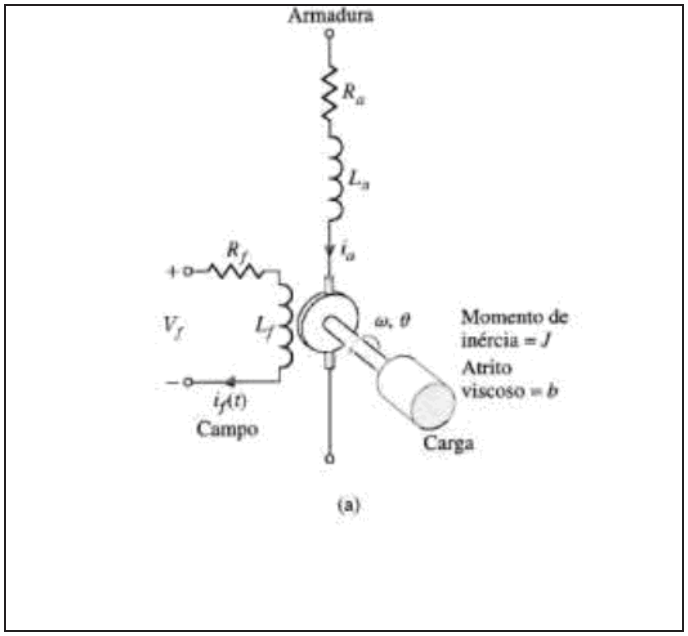
Recebem várias classificações, de acordo com a forma de produção do campo magnético e conforme o projeto de construção da armadura. De acordo com a forma de produção do campo magnético, os motores CC são classificados em Motores de Fluxo Magnético Variável e Motores de Fluxo Magnético Constante (TORRES; HSU; CUNHA, 2004).

O motor de corrente contínua é constituído basicamente de:

- **Rotor** é a porção central girante. O termo pode também ser aplicado ao enrolamento que se encontra no rotor.
- **Estator** é o enrolamento estático (estacionário) ao redor do rotor. Em motores pequenos, o estator pode ser trocado por imã permanente.
- **Comutador** é a **c**onexão através de escovas com o enrolamento do rotor.

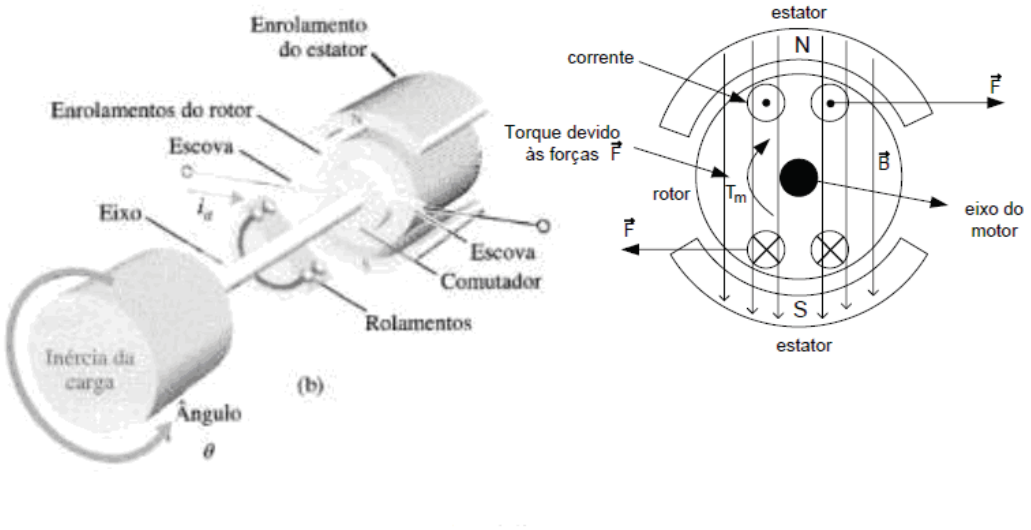
A seguir, tem-se na Figura 12, um exemplo de um circuito elétrico para um motor CC e na Figura 13 a sua vista interna.

Figura 12 - Circuito elétrico -Motor CC.



Fonte: (TORRES, 2004).

Figura 13 - Motor CC e sua vista interna.



Fonte: (TORRES, 2004).

O funcionamento deste motor está associado ao efeito da indução/repulsão magnética. Uma espira é mergulhada em uma região que possui um fluxo magnético provocando a sua rotação (Figura 13); como a espira é fixa no estator, são polarizadas apenas algumas bobinas enquanto ocorre o giro do rotor, possibilitando a rotação de 360°. O processo se repete continuamente gerando uma rotação permanente do rotor. O fluxo magnético pode ser invertido, alterando-se os polos do circuito, fazendo com que a corrente elétrica circule para o outro lado, assim, pode-se controlar a rotação do motor CC tanto no sentido horário quanto no anti-horário. A equação que rege a velocidade linear do motor CC é dada por:

$$V = IR + L\left(\frac{dI}{dt}\right) + k_e\omega \qquad (9)$$

Onde, V - velocidade; I – corrente; R - resistência; L - indutância; ω - velocidade angular;

k_e- constante elétrica.

A equação do torque gerado pelo giro do motor, é dada por :

$\tau = k_t I$	(10)

Onde, k_t- constante de torque.

Neste trabalho foi utilizado um motor CC parametrizado por: redução 1: 40 - 5V - 330 RPM. Mostram-se na Figura 14, o motor e suas características dadas pelo fabricante.

Figura 14 - Motor CC e as especificações do fabricante

Modelo	Tensão		Sem carga		Máximo rendimento				Máxima Potência	
	operação	nominal	rotação	corrente	rotação	corrente	torque	potência	torque	Velocidade
MUL5-R330	3~12V	5V	330rpm	0.33A	280rpm	1.44A	0.63kgf.cm	1.8W	1.48kgf.cm	184 rpm

Fonte: (MULT COMERCIAL, 2012)

4.5.2 Acionamentos de Motores CC

Nos motores em geral, a tensão de partida deve ser grande o suficiente para romper o coeficiente de atrito estático que é maior do que o coeficiente de atrito móvel. Sendo assim, o grande desafio para um circuito elétrico de acionamento é dar a partida no motor CC a partir do seu repouso e manter as condições necessárias para o seu movimento.

Portanto, além do problema da partida, o sistema de controle de um motor de corrente contínua, necessita controlar a sua velocidade e a direção. Tem-se algumas opções para o circuito de acionamento e controle de motores CC, como o acionamento com portas lógicas discretas e ponte H, ou na forma de Circuitos Integrados com ponte H e usando microcontroladores. Neste projeto usa-se o dispositivo integrado L298 que apresentam em sua estrutura duas pontes H, controlados via placa DE2 da Altera e o programa em VHDL.

Ainda, esse acionamento dos motores CC pode ser feito através da variação contínua da tensão aplicada ao motor ou é usada a técnica de Modulação por largura de pulso ou PWM, que diminui as perdas de energia e, por conseguinte reduzem o aquecimento dos componentes do driver de acionamento, reduzindo custo e tamanho.

4.6 Plataforma de desenvolvimento

Usa-se neste trabalho como plataforma de desenvolvimento do projeto, o kit da Altera DE-2 mostrado na Figura 15. Este kit usa como elemento principal, o dispositivo FPGA Cyclone II EP2C35F672C6 e o dispositivo serial de configuração EPCS16. A placa contém memórias, LEDs, chaves e expansores suficientes para o desenvolvimento de trabalhos complexos. Algumas características importantes do kit são listadas a seguir:

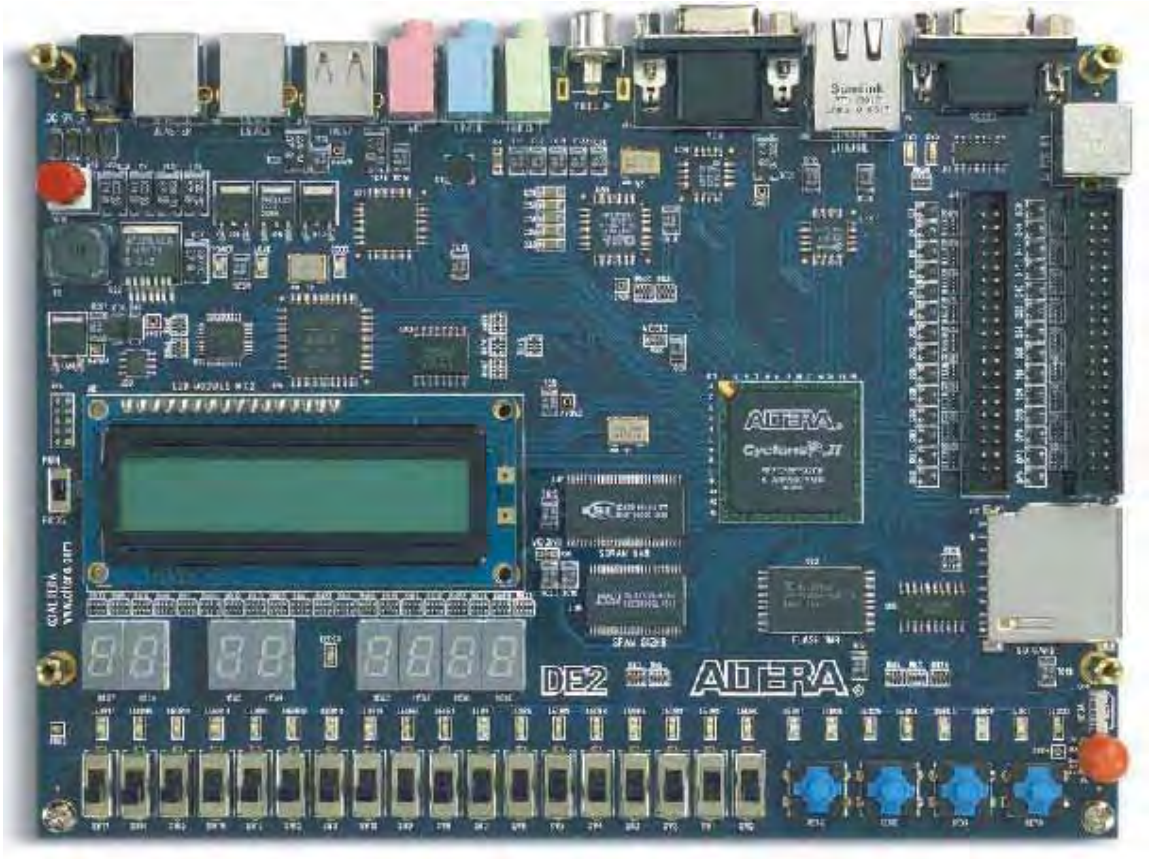
- Dispositivos de I/O;
- USB *Blaster* para configuração do FPGA;
- Entradas RS-232 para conexão com periféricos;
- 4-MB Flash (1 Mbyte on some boards), 512 KB SRAM, 8 MB de SDRAM;
- 18 chaves do tipo liga /desliga e 4 Chaves do tipo push-button;
- Display LCD 16 x 2 e 8 *displays* de 7-segmentos;
- 18 LEDs vermelhos e 9LEDs verdes;
- 1 X Oscilador de 50-MHz e 1 X 27-MHz;
- Controlador USB Host/Slave com USB com conectores do tipo A e B;
- 2expansores de 40 pinos (com diodo e resistor de proteção).

Com esta placa pode-se programar no modo gráfico ou em texto (linguagem de descrição de hardware). O modo texto apresenta-se como um recurso importante que está

sendo usado neste projeto, de forma que o circuito do dispositivo programável seja configurado a partir de uma arquitetura programada na linguagem de descrição de hardware VHDL. Desta maneira, consegue-se tempos menores de execução, facilidade de implementação e testes, com o recurso da simulação, agilizando o processo de realização do trabalho de pesquisa, superando as limitações quanto à quantidade de circuitos que podem ser configurados e a pouca experiência do programador com a linguagem.

A programação do circuito na placa pode ser feita in-sytem, de forma provisória ou definitiva, bastando para isso, ter o software de programação e desenvolvimento do kit, que é o software Quartus II-Altera.

Figura 15 - Placa de desenvolvimento da Altera-DE2



Fonte: (ALTERA, 2010).

Utiliza-se principalmente no projeto, os expansores de 40 pinos, IDE (*Integrated Drive Electronics*) que fornecem a comunicação entre a placa e o meio externo, como por

exemplo, o circuito de acionamento e potência do motor (saídas) e o conversor A/D (entradas dos sensores), além dos sinais de alimentação, VCC e o terra, GND. Outros recursos são utilizados para teste na criação dos blocos para a verificação do funcionamento esperado, como os leds.

Utilizam-se também os dois osciladores disponíveis, 50 e 27 MHz sendo necessária a criação de blocos de divisão de frequência por conta da velocidade da operação dos dispositivos externos – muito inferiores, por exemplo o ADC. Para o conversor e demais blocos, cria-se um divisor de frequência de 500 kHz, para o correto sincronismo de operação do projeto.

5 DESENVOLVIMENTO DO SISTEMA DE NAVEGAÇÃO DO ROBÔ.

Neste item descreve-se o desenvolvimento dos vários blocos realizados e testados. São eles, o bloco de controle da conversão A/D que converte para um sinal de oito bits os dados de cada sensor de distância, e o processo da lógica *fuzzy*, que engloba os blocos do fuzificador, as regras de inferência e o defuzificador, usando para isso a linguagem VHDL e os recursos do software como, as megafunções.

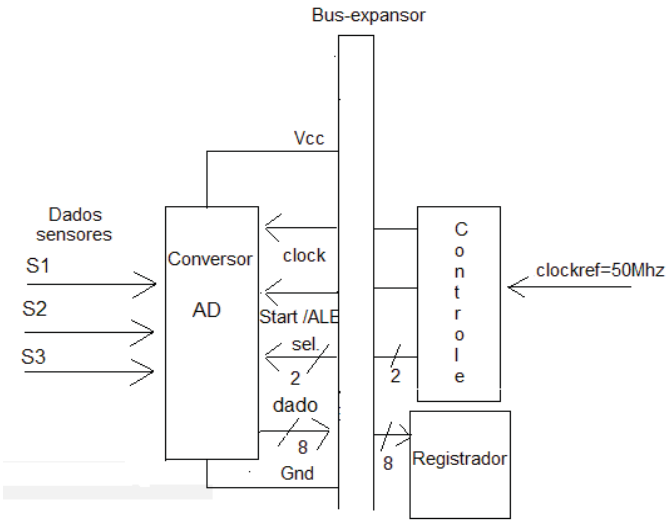
5.1 Bloco de Controle

Este bloco foi implementado para controlar as entradas via conversor A/D e fazer o sincronismo de todo o processo da lógica *fuzzy* programado em VHDL.

O conversor A/D possui oito entradas analógicas multiplexadas, das quais, três estão sendo usados na aquisição de dados para os sensores de distância, com saídas de oito bits. Desta forma, as saídas de 8 bits foram armazenadas de modo provisório, em *latches,* um pra cada sensor, projetados em texto VHDL e são usados para o sincronismo de início do processamento para o controle *fuzzy*.

No controle do conversor Analógico/Digital, ADC0808 considera-se as temporizações do dispositivo apresentada pelo fabricante em sua folha de dados. Por isso, os sinais de controle deste bloco são disparados utilizando um *clock* com frequência de 500k Hz. Além de controlar o ADC, iniciando a conversão, este bloco faz a varredura dos três canais (um canal para cada sensor) e ativa os *latches*. Estes armazenam as informações de cada sensor de forma temporária até que seja liberado pelo conversor através do sinal fim de conversão, EOC. O diagrama bloco da arquitetura implementado para o controle e aquisição de dados pelo conversor A/D para os três sensores é mostrado na Figura 16.

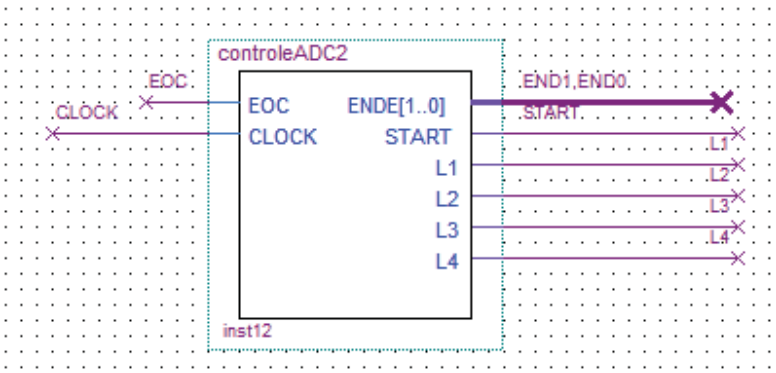
Figura 16 - Arquitetura de controle e aquisição de dados do conversor A/D.



Fonte: Própria do autor

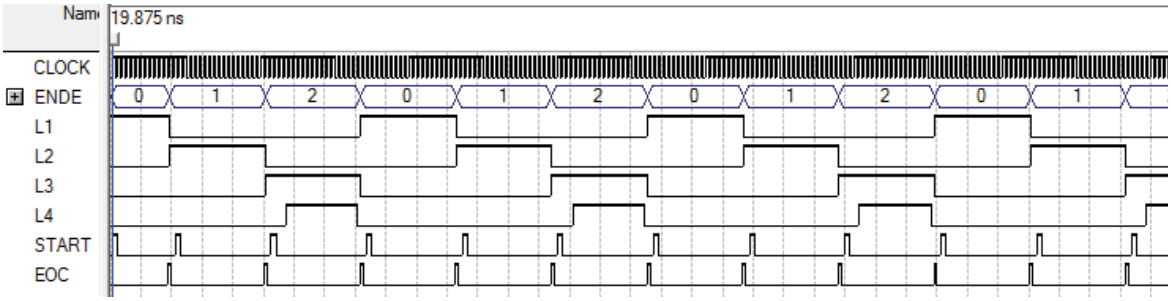
Na Figura 17 apresenta-se o símbolo do bloco de controle do ADC. Os sinais de entrada deste bloco são o CLOCK e o EOC (final de conversão) e os sinais de saída são o vetor de endereçamento ENDE[1..0], o de partida para a conversão “START” e o vetor de acionamento dos *latches*, L[3..0] que fazem parte do próximo bloco. Na Figura 18, apresenta-se uma simulação para este bloco, realizada no software Quartus II ver. 9.1.2 da Altera (ALTERA, 2011).

Figura 17 - Esquema da pinagem ou símbolo para o bloco de controle do A/D, desenvolvido em VHDL.



Fonte: Própria do autor

Figura 18 - Uma simulação de teste do bloco de controle e aquisição de dados.



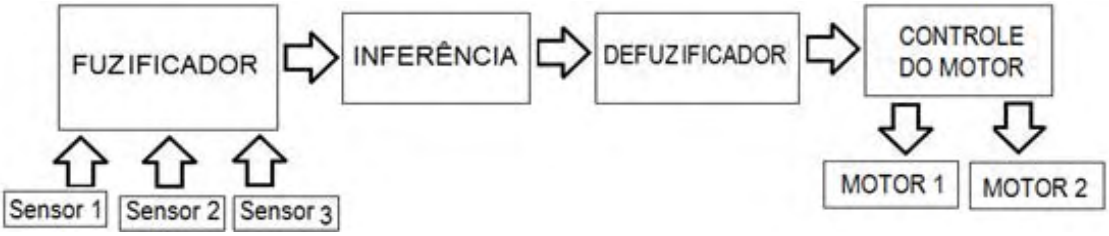
Fonte: Própria do autor.

Mostra-se nesta simulação a aquisição de dados com a seleção dos canais zero, um e dois pela entrada *ENDE*. No início da aquisição, para cada canal, acontece quando há um pulso START. Os sinais de saída, L_4 a L_1dependem, do sinal de EOC. Assim, quando o ADC está pronto, ativa o sinal de saída EOC e este sinal dispara um dos latch, que armazena o resultado da medida do sensor (um latch para cada sensor), incrementa o endereço e em seguida dispara o START novamente, para uma nova conversão, evitando assim uma leitura incorreta de dados. Ressalta-se na simulação que a base de *clock* é igual a 500 kHz, obtido pela divisão de freqüência do oscilador de 50 MHz.

5.2 Controlador *Fuzzy*

O controlador envolvendo todo o processo *fuzzy* é mostrado em um diagrama de blocos esquemático mostrado na Figura 19. Observa-se neste diagrama do controlador, desde às entradas dos sinais dos sensores, até o bloco de acionamento de saída para os dois motores. Os blocos em sua maioria foram implementados na linguagem VHDL ou através de funções paramétricas – LPMs, da biblioteca do software Quartus II. Detalha-se a seguir alguns dos principais blocos.

Figura 19 - Diagrama de blocos do Sistema Controlador *Fuzzy*

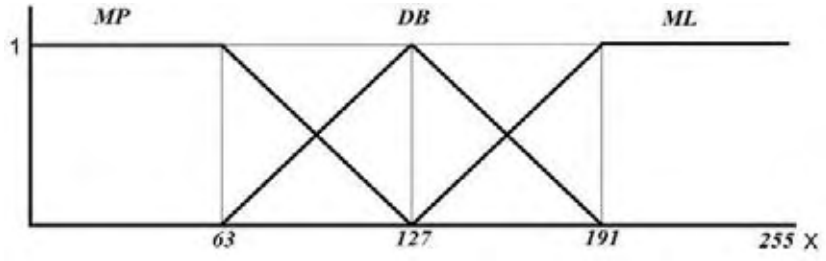


Fonte: Própria do autor

5.3 Fuzificador

Usando a teoria da lógica *fuzzy* para implementar o *fuzificador* foram definidos os conjuntos *fuzzy* e as formas das funções de pertinência. Portanto, o conjunto *fuzzy* para os sensores de distância é a distância, cujas funções de pertinência podem ser definidas por: MP – Muito Perto, DB – Distância Boa e ML – Muito Longe, nas formas de funções trapezoidal e triangular, conforme é mostrado na Figura 20.

Figura 20 - Função de pertinência



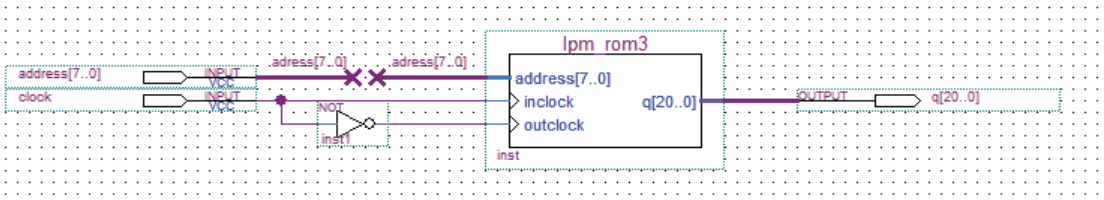
Fonte: Própria do autor

Por exemplo, *x* representa os valores da distância do robô a um obstáculo, em Volts, escalonada para a resolução de 8 bits equivalentes aos valores 0 a 5 volts ou em distância, de 0 a 80 cm. O subconjunto MP de *x* que representa a distância MUITO PERTO, $\mu MP(x)$ é o conjunto *fuzzy* com a função de pertinência mostrada na Figura 20. Observa-se nesta

figura que $\mu MP(x) = 1$ para distâncias até 63, a partir disso começa a decrescer. No caso de considerarmos distância Muito Longe, $\mu ML(x) = 1$ para valores de x no intervalo de 191 a 255.

Implementa-se o *fuzificador* através de uma memória ROM de 256X21 bits (figura 21) usando funções paramétricas contidas na biblioteca do software "Quartus II" devido ao pequeno tempo de resposta destas funções e a facilidade de configuração, bastando endereçar para se ter o resultado na saída. Cada localização da memória contém o grau de pertinência para cada gráfico de um sensor. Portanto, como são três sensores, tem-se três memórias, uma para cada sensor, com conteúdos idênticos, programados previamente. O endereço é a tensão equivalente à distância, obtida da medição no sensor.

Figura 21 - *Fuzificador* para um sensor



Fonte: Própria do autor.

Com a resolução do conversor A/D em oito bits, têm-se 256 valores de tensão fornecidos pelos sensores de distância na faixa de 10 a 80 cm. Porém, para facilitar o controle da arquitetura do projeto, a resolução das informações (grau de pertinência) da memória, faz-se a decodificação com sete bits, portanto isso explica o *fuzificador* (memória) com 21 bits de dados e 8 bits para endereço (256 x 21 bits).

Neste bloco de ROM, o sinal de sincronismo de entrada INCLOCK, é defasado para o *clock* do sinal de saída, OUTCLOCK, por uma porta lógica NOT, atraso suficiente para a resposta da memória.

Mostra-se na Figura 22, um trecho do conteúdo de uma das memórias, que representa um dos três *fuzificador*. Observa-se na figura, que para o endereço 57 (56 + 1), por exemplo, o dado é codificado em três partes "000000000000001000000" sendo os sete bits mais significativos referentes à função de pertinência MP, os sete bits do meio, à

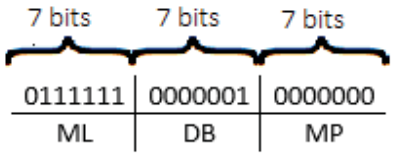
função DB e por fim, os sete bits menos significativos referentes à função ML. Na Figura 23, mostra-se a ordem como é decodificada a informação na memória.

Figura 22 - Trecho do conteúdo da memória ROM, em binário, para um *fuzificador*.

Addr	+0	+1	+2	+3	+4
0	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
8	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
16	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
24	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
32	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
40	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
48	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
56	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000	0000000000000001000000
64	0000000000000010111111	0000000000000100111110	0000000000000110111101	0000000000001000111100	0000000000001010111011
72	0000000000010010110111	0000000000010100110110	0000000000010110110101	0000000000011000110100	0000000000011010110011
80	0000000000100010101111	0000000000100100101110	0000000000100110101101	0000000000101000101100	0000000000101010101011
88	0000000000110010100111	0000000000110100100110	0000000000110110100101	0000000000111000100100	0000000000111010100011
96	0000000001000010011111	0000000001000100011110	0000000001000110011101	0000000001001000011100	0000000001001010011011

Fonte: Própria do autor.

Figura 23 - A ordem da decodificação da informação na memória ROM, das funções de pertinência (em binário), para um f*uzificador*.



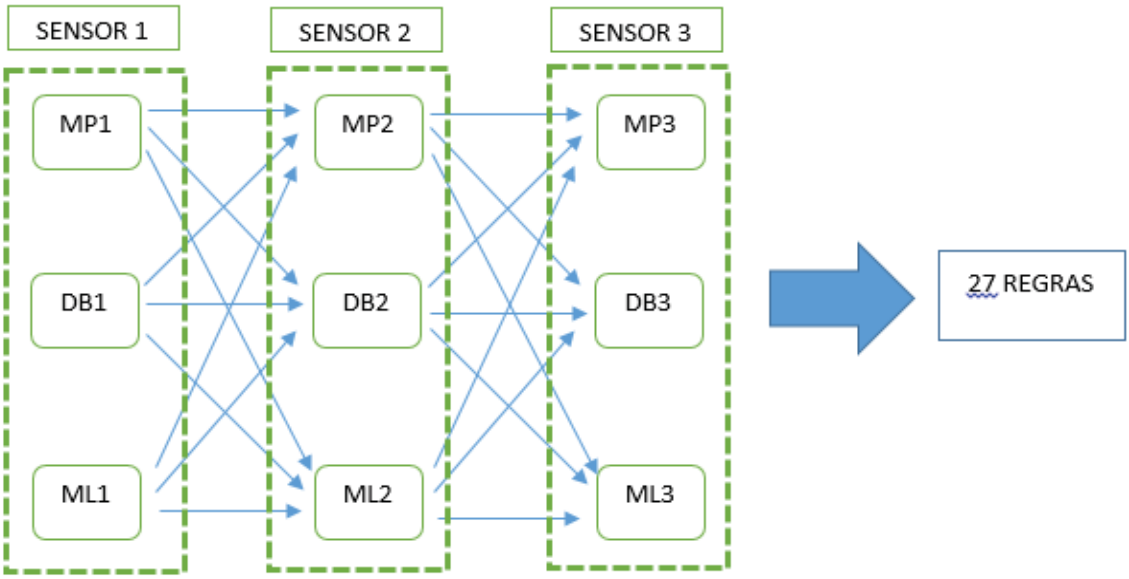
Fonte: Própria do autor.

5.4 Regras de Inferência

Neste trabalho têm-se três entradas originadas uma de cada sensor e para cada entrada têm-se três funções de pertinência (MP, DB e ML). Desta forma, o número total de regras para este projeto é de 3^3, ou seja, vinte e sete regras, conforme Eq. 6.

Cada regra relaciona uma função de pertinência de cada entrada. Cobrindo-se o universo de possibilidades escolhe-se a ação de controle para cada uma delas. Nesta etapa pode-se recorrer ao conhecimento de um especialista, por exemplo. Na Figura 24 mostra-se como é feito o arranjo para a obtenção do universo de regras.

Figura 24- Arranjo entre as funções de pertinência para obtenção do conjunto de regras.



Fonte: Própria do autor

Para este projeto, são estabelecidas as ações "Velocidade Positiva" – VP –, "Velocidade Nula" – VN – e "Velocidade Negativa" – VN. Assim, quando se imagina que o robô encontra obstáculos MP ("Muito Perto") na frente dos seus três sensores, ele deve adotar movimentos de rotação em torno de si mesmo. O bom senso prevalece nesta etapa, o que é uma grande vantagem dos controladores *fuzzy* sobre os controladores clássicos como já foi dito anteriormente.

Exemplificando, se o evento aponta para a regra 15, na Tabela 2, observa-se que esta regra é ativada quando os sensores 1, 2 e 3 detectam distâncias que correspondem às funções "DB1", "DB2" e "ML3" respectivamente. As ações de controle são: a roda esquerda deve adotar a velocidade VP e a roda direita a velocidade V0.

Tabela 2 - Regras de Inferência

REGRA		S2	S1	S3	**RODA ESQ**	**RODA DIR**
1		MP1	MP2	MP3	VN	VP
2		MP1	MP2	DB3	VP	V0
3		MP1	MP2	ML3	VP	VN
4		MP1	DB2	MP3	VN	VP
5		MP1	DB2	DB3	VP	VN
6		MP1	DB2	ML3	VP	V0
7		MP1	ML2	MP3	VP	VP
8		MP1	ML2	DB3	VP	VP
9		MP1	ML2	ML3	VP	VP
10		DB1	MP2	MP3	VN	VP
11		DB1	MP2	DB3	VN	VP
12		DB1	MP2	ML3	VP	VN
13		DB1	DB2	MP3	V0	VP
14		DB1	DB2	DB3	VN	VP
15		DB1	DB2	ML3	VP	V0
16		DB1	ML2	MP3	VP	VP
17		DB1	ML2	DB3	VP	VP
18		DB1	ML2	ML3	VP	VP
19		ML1	MP2	MP3	VN	VP
20		ML1	MP2	DB3	VN	VP
21		ML1	MP2	ML3	VN	VP
22		ML1	DB2	MP3	V0	VP
23		ML1	DB2	DB3	VN	VP
24		ML1	DB2	ML3	VP	VN
25		ML1	ML2	MP3	VP	VP
26		ML1	ML2	DB3	VP	VP
27		ML1	ML2	ML3	VP	VP

Fonte: Própria do autor

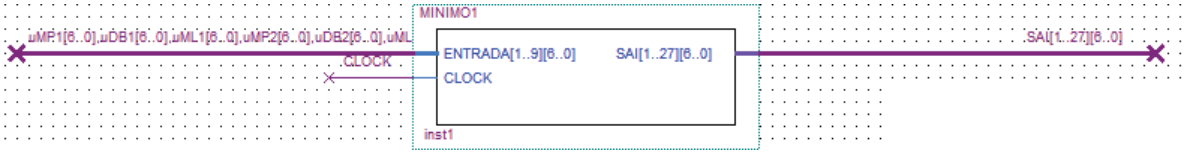
5.5 Blocos de Mínimo

Com as regras de inferência estabelecidas o próximo passo é achar o menor valor entre os graus de pertinência oriundos das funções de pertinência a que se refere cada regra ou seja, este bloco apresentará vinte e sete valores correspondentes aos menores valores de cada grupo de três graus de pertinência. Estes valores mínimos são agora imputados às funções de pertinência do *defuzificador* (ações: VP, V0 e VN) relacionadas à regra que a

referencia. Para esclarecer, ainda utilizando-se da regra 15, vemos que os valores mínimos entre “DB1”, “DB2” e “ML3” são imputados às funções VP e V0 das rodas esquerda e direita, respectivamente.

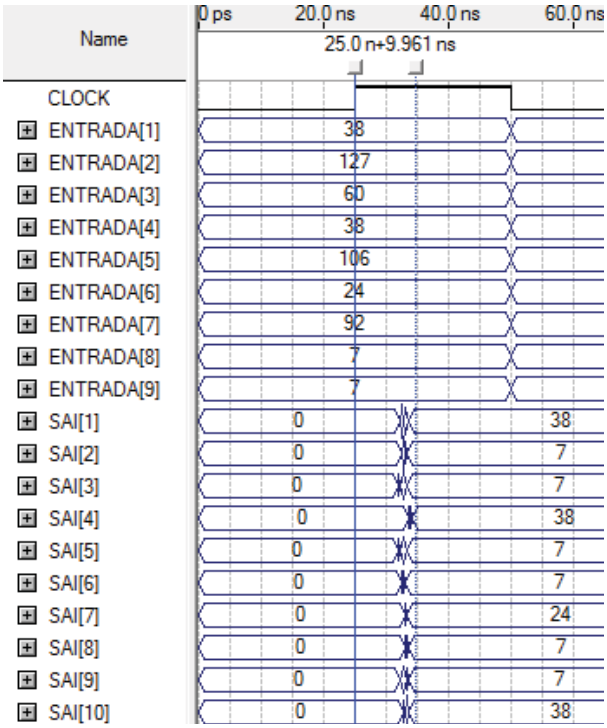
O bloco de Mínimo apresenta em sua saída o menor valor das funções de pertinências que compõe uma regra. Este bloco possui um CLOCK para o sincronismo da saída conforme pode ser visto na Figura 25, e a simulação na Figura 26.

Figura 25 - Bloco de Mínimo



Fonte: Software Quartus II (ALTERA, 2011).

Figura 26 - Simulação do bloco Mínimo com *clock* de período, 50ns



Fonte: Software Quartus II (ALTERA, 2011).

Este bloco coloca em sua saída SAI[n] o menor valor dentre os valores das três funções de pertinências que compõem a regra n. As entradas deste bloco são os graus de pertinência vindos das memórias (*fuzificadores*). Na Tabela 3 apresenta-se a correspondência entre os nomes das entradas do bloco, como pode ser visto na simulação (Figura 26), com as funções de pertinência.

Tabela 3 - Correspondência do nome das entradas com as funções de pertinência

Entradas (1 a 9)	Função de Pertinência
ENTRADA 1	uMP1
ENTRADA 2	uDB1
ENTRADA 3	uML1
ENTRADA 4	uMP2
ENTRADA 5	uDB2
ENTRADA 6	uML2
ENTRADA 7	uMP3
ENTRADA 8	uDB3
ENTRADA 9	uML3

Fonte do próprio autor

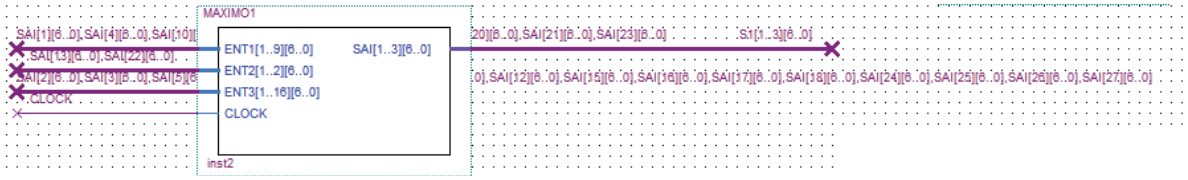
O resultado do bloco "MINIMO" pode ser verificado através da simulação escolhendo-se a regra 10, por exemplo. Para esta regra o valor mínimo entre os valores dos graus de pertinência correspondentes à "DB" para o Sensor 1, "MP" para o Sensor 2 e "MP" para o Sensor 3. Transformando no vetor entrada do bloco "MÍNIMO", tem-se ENTRADA2 (DB1), ENTRADA4 (MP2) e ENTRADA7 (MP3). Através da simulação feita para teste do referido bloco, observa-se os valores para estas entradas: "127", "38" e "92". Como esperado, a saída foi o menor destes três valores – SAI[10] = 38.

5.6 Blocos Máximos

Através deste bloco (Figura 27), faz-se a operação de UNIÃO, isto é, seleciona-se o valor maior entre os valores mínimos sobre uma mesma função de pertinência do *defuzificador* (VP, V0 e VN) que se sobrepõem, isto é, quando uma ação é chamada por mais de uma regra, têm-se sobreposições de mínimos numa mesma função de pertinência,

uma vez que ela é a ação. Da Tabela 2, pode-se observar que a função de pertinência (ou ação) “V0” da roda direita é chamada por três regras (2, 6 e 15), portanto, haverá sobreposição nesta função. O bloco Máximo faz então, a união desses valores, isto é, o maior grau de pertinência.

Figura 27 - Bloco Máximo - responsável por encontrar o maior valor.



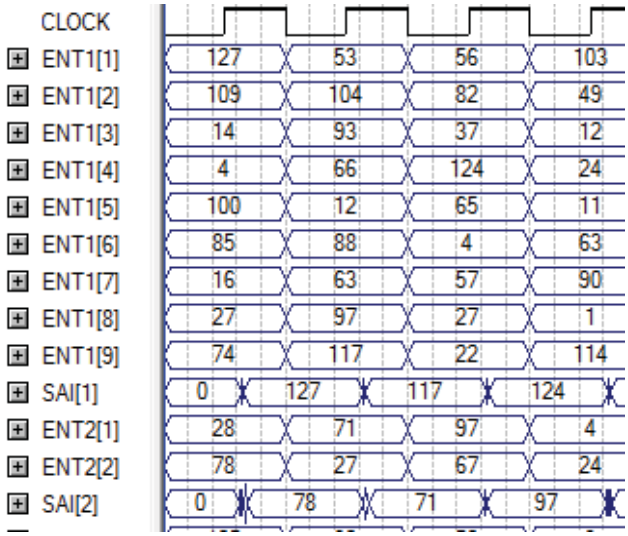
Fonte: Própria do autor

As entradas deste bloco são referentes às regras para cada função (VN – velocidade negativa, V0 – velocidade nula, VP – velocidade positiva) do *defuzificador*, isto é, nove regras levam à velocidade VN, duas regras levam à velocidade “V0” e 16 regras levam à velocidade VP para a roda esquerda. Observe que para a roda direita é diferente, como pode ser visto pela tabela de regras.

As saídas são “VN” (SAI[1]), V0 (SAI[2]) e VP (SAI[3]) e seus valores são os máximos entre os valores das regras que se sobrepõe para cada uma destas velocidades. Exemplificando, para a roda esquerda, as regras 8 e 9 se sobrepõe, a saída será o valor da regra que tem valor maior.

Na Figura 28 apresenta-se uma simulação deste bloco para alguns valores aleatórios apenas para verificação do seu funcionamento.

Figura 28 - Simulação do BLOCO MÁXIMO para um *clock* de 50ns



Fonte: Própria do autor

Nesta simulação as entradas ENT1[1], ENT1[2] a ENT1[9] são valores mínimos oriundos de nove regras que chamam a velocidade “VN” para a roda esquerda. O valor de “VN” é dado pelo sinal de saída SAI[1], onde verifica-se que depois do valor “0” com um certo atraso tem-se o valor maior, igual a 127, para este conjunto de entradas.

5.7 Defuzificador

Como dito anteriormente, foi escolhido para o *defuzificador* o método do centróide, que é um dos mais usados. A equação que define este método é dada por :

$$cx = \sum \frac{At.ct}{At} \qquad (11)$$

onde Ai e ci representam a i-ésima área e i-ésimo centroide respectivamente.

Entretanto, o centróide de cada função de pertinência foi calculado através de uma

função aproximada disponível em Minussi (2009) dada por:

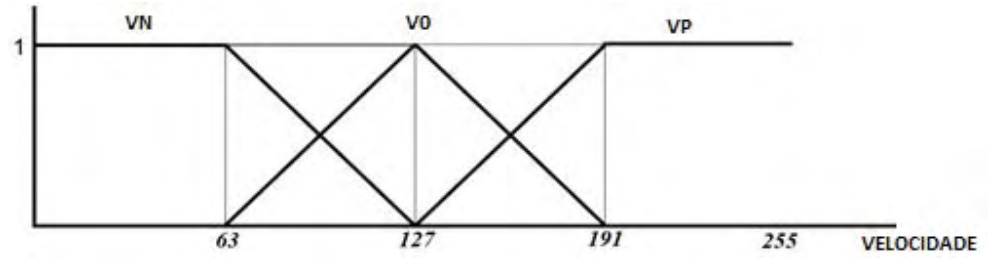
$$\frac{\sum_{i=1}^{n} \mu_i u_i}{\sum_{i=1}^{n} \mu_i} \quad (12)$$

Onde *n* é o número de regras, μ_i o grau de pertinência da regra *i* e u_i é a projeção, no eixo das abscissas, do valor máximo da função de pertinência *i*.

Este cálculo é uma abordagem alternativa muito empregada em controladores *fuzzy.* Ressalta-se que a lógica *fuzzy* tem uma tolerância a valores não exatos sendo, portanto, irrelevante os erros por conta da equação aproximada para o cálculo do centróide.

Na Figura 29 mostram-se as funções de pertinência do *defuzificador*, escolhidas para este projeto.

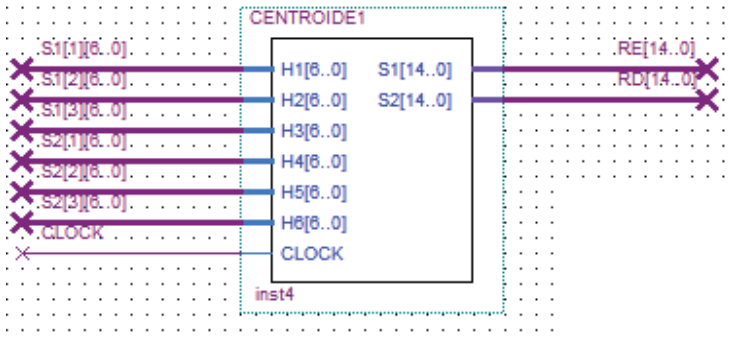
Figura 29 - *Defuzificador*



Fonte: Própria do autor

Da mesma forma como ocorre em outras etapas onde são necessárias réplicas de blocos, nesta decidiu-se pela implementação de vários blocos ao invés de replicá-los, garantindo assim, o uso de elementos lógicos indispensáveis, diminuindo então a área utilizada do chip FPGA. Portanto, têm-se dois blocos centróide (uma para cada roda). O bloco Centróide encontra-se na Figura 30. Vale ressaltar que os valores especificados para os valores de velocidade na figura 29, representados com resolução de 8 bits, devem ser compatíveis com os valores para o acionamento adequado dos motores.

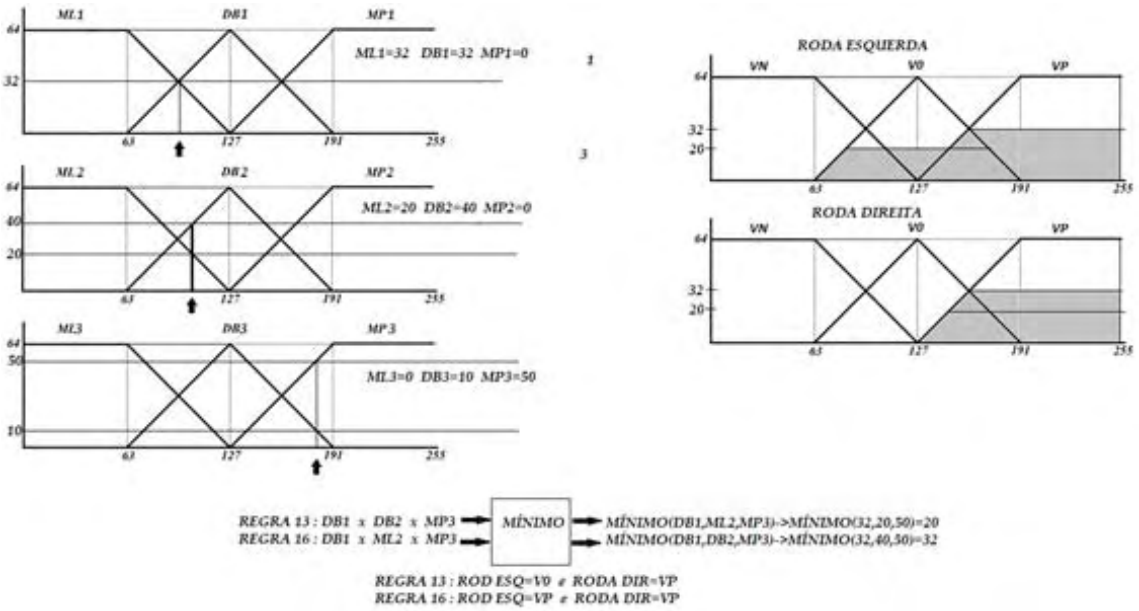
Figura 30 - Bloco Centroide



Fonte: Própria do autor

No bloco do centróide as entradas são os valores máximos de cada velocidade “VN”, “V0” e “VP”. Este bloco apresenta em sua saída o valor do centroide da área formada pelos máximos destas velocidades. Ou seja, “H1”, “H2” e “H3” são os máximos das funções “VN”, “V0” e “VP” da roda esquerda e “H4”, “H5” e “H6”, os máximos das “VN”, “V0” e “VP” da roda direta.

Figura 31 - Caso específico para exemplificação



Fonte: Própria do autor

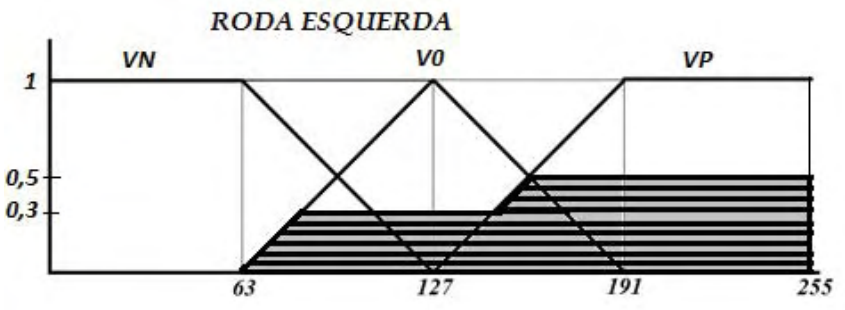
Na Figura 31, mostra-se um exemplo com as etapas de funcionamento dos blocos (*fuzificador*, mínimo, máximo e *defuzificador*). Foram utilizadas as regras 13 e 16, onde a regra 13 relaciona “DB1”, “DB2” e “MP3” e a regra “16” relaciona “DB1”, “ML2” e “MP3”. Quando os sensores detectam obstáculos em distâncias correspondentes aos pontos indicados pelas setas, o *fuzificador* apresenta os graus de pertinência para cada função. Pelo exemplo o *fuzificador* 1 apresentará grau de pertinência 32 para a função “ML1”, 32 para “DB1” e 0 para “MP1”; o *fuzificador* 2, 20 para “ML2”, 40 para “DB2” e 0 para “MP2” e por fim o *fuzif*icador 3, 0 para “ML3”, 10 para “DB3” e 50 para “MP3”.

O Bloco Mínimo, através da tabela de regras, se encarrega de apresentar o menor valor entre os graus de pertinência que cada regra referencia. Para a regra 13, observa-se da tabela de regras que, estão relacionadas às funções “DB1”, “DB2” e “MP3” e para a regra 16, “DB1”, “ML2” e “MP3”. Portanto, o bloco MÍNIMO mostrará em sua saída o valor 20 para a relação referenciada pela regra 13 e o valor 32 para a regra 16. Como cada regra está relacionada também com uma função velocidade da roda esquerda e uma função velocidade da roda direita, pode-se verificar que a regra 13 leva à “V0” e “VP” para a roda esquerda e direita, respectivamente. Já a regra 16 leva à “VP” para a roda esquerda e “VP” também para a roda direita.

O Bloco Máximo faz a união, isto é, o maior valor entre os valores de uma mesma função velocidade. Do exemplo, pode-se verificar que para a roda direita, a função “VP” possui duas linhas horizontais que são os valores mínimos das regras 13 e 16 para esta roda. O bloco faz então a união das áreas, como pode ser visto na Figura 32.

O trabalho final é feito pelo Bloco Centróide, que determina o centro da área formada pela união das áreas de cada função de velocidade. Para a roda esquerda, neste exemplo, o Bloco Centróide tem como saída a projeção do centróide da área maior (após a união) das funções V0 e VP no eixo das abscissas. Este valor corresponde à velocidade com que o motor deve girar.

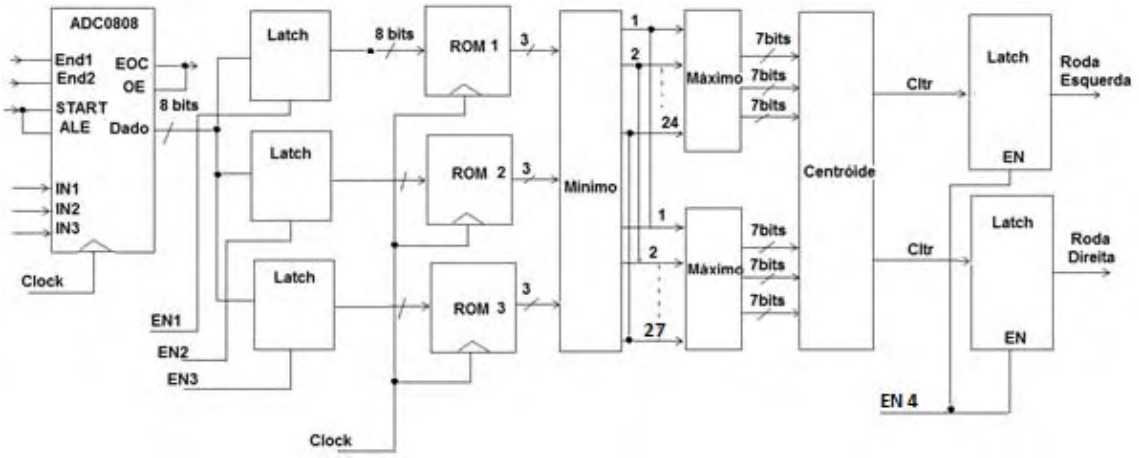
Figura 32 - Gráfico de *defuzificação* para o exemplo.



Fonte: Própria do autor

Resumindo, tem-se na Figura 33 o sistema completo, sem o bloco de controle, Neste, a primeira etapa é feita pelos sensores que convertem a distância entre ele e um objeto em tensão. Por ser um sensor analógico, necessita-se de um conversor A/D para digitalização da tensão. Os valores das distâncias convertidos são demultiplexados e armazenados nos *Latches*, acionados pelo controlador. Estas saídas digitalizadas se tornam endereços para as memórias, que são os *fuzificadores*. Cada fuzificador tem três saídas, que representam cada uma das funções de pertinência. Os valores *fuzificados* são entradas do bloco mínimo para realização da operação de interseção. Em seguida, têm - se dois blocos, um para cada roda, para realização da operação de união. Os resultados da operação de união são inseridos no bloco Centróide, que é a *defuzificação*. As duas saídas do bloco Centróide são valores crisp que correspondem à velocidade que o motor deve girar.

Figura 33 - Processo completo do controlador *fuzzy* sem o controlador

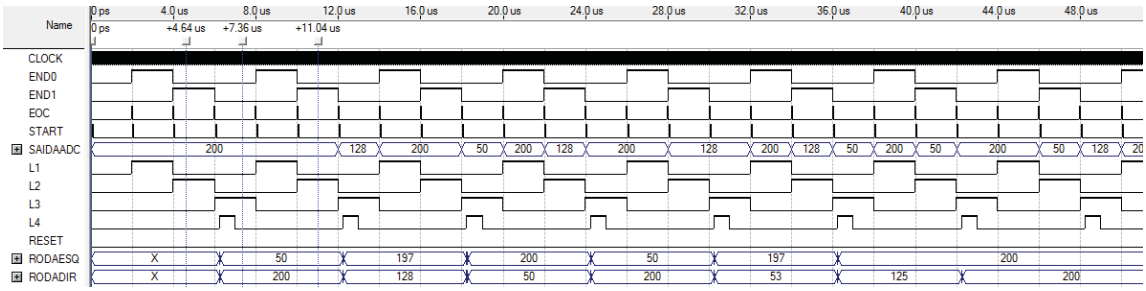


Fonte: Própria do autor

6 ANÁLISE DAS SIMULAÇÕES

Após a verificação do funcionamento de cada bloco através da análise das simulações individuais, faz- se a junção de todos os blocos para obtenção da simulação do sistema completo , mostrado na Figura 34.

Figura 34 – Simulação do sistema completo



Fonte: Própria do autor

Nesta simulação os sinais que aparecem são descritos a seguir:

END0 e END1 – endereçamento, sendo o END0 o bit menos significativo do endereço.

EOC – End Of Conversion – Fim de Conversão.

START – sinal de disparo para o ADC para início de conversão.

SAIADC – Dados obtidos após a conversão pelo ADC.

L1, L2, L3 – controle dos *latches* para cada sensor e o L4 é o controle para o Latch que habilita o último bloco – Centróide.

RODAESQ e RODADIR - valores de*fuzzy*ficados que representam a velocidade dos motores da esquerda e da direita, respectivamente.

Para analisar os resultados obtidos nesta dissertação, utiliza-se a *toolbox "fuzzy"* do Matlab para possibilitar a comparação com os valores obtidos através das simulações realizadas com o software Quatus II.

No Apêndice A apresentam-se os procedimentos de como foram obtidos os resultados para os mesmos valores da simulação da Figura 33 no toolbox *fuzzy* do Matlab. Os resultados da simulação do sistema completo no Quartus II e os obtidos no Matlab constam da Tabela 4 a seguir.

Tabela 4 - Comparação entre os resultados obtidos da simulação no Quartus II e no MatLab

QUARTUS						MATLAB						
REGRA	S1	S2	S3	RE	RD	NS1	NS2	NS3	RESQ	RDIR	CONVERTIDOS	
1	200	200	200	50	200	0,78125	0,78125	0,78125	0,192	0,808	49,152	206,848
2	200	200	128	197	128	0,78125	0,78125	0,5	0,808	0,5	206,848	128
3	200	200	50	200	50	0,78125	0,78125	0,195313	0,808	0,192	206,848	49,152
4	200	128	200	50	200	0,78125	0,5	0,78125	0,192	0,808	49,152	206,848
5	200	128	128	197	53	0,78125	0,5	0,5	0,808	0,192	206,848	49,152
6	200	128	50	200	125	0,78125	0,5	0,195313	0,808	0,5	206,848	128
7	200	50	200	200	200	0,78125	0,195313	0,78125	0,808	0,808	206,848	206,848
8	200	50	128	200	200	0,78125	0,195313	0,5	0,808	0,808	206,848	206,848
9	200	50	50	200	200	0,78125	0,195313	0,195313	0,808	0,808	206,848	206,848
10	128	200	200	50	200	0,5	0,78125	0,78125	0,192	0,808	49,152	206,848
11	128	200	128	52	198	0,5	0,78125	0,5	0,192	0,808	49,152	206,848
12	128	200	50	200	50	0,5	0,78125	0,195313	0,808	0,192	206,848	49,152
13	128	128	200	125	200	0,5	0,5	0,78125	0,5	0,808	128	206,848
14	128	128	128	53	196	0,5	0,5	0,5	0,192	0,808	49,152	206,848
15	128	128	50	200	125	0,5	0,5	0,195313	0,808	0,5	206,848	128
16	128	50	200	200	200	0,5	0,195313	0,78125	0,808	0,808	206,848	206,848
17	128	50	128	200	200	0,5	0,195313	0,5	0,808	0,808	206,848	206,848
18	128	50	50	200	200	0,5	0,195313	0,195313	0,808	0,808	206,848	206,848
19	50	200	200	50	200	0,195313	0,78125	0,78125	0,192	0,808	49,152	206,848
20	50	200	128	50	200	0,195313	0,78125	0,5	0,192	0,808	49,152	206,848
21	50	200	50	50	200	0,195313	0,78125	0,195313	0,192	0,808	49,152	206,848
22	50	128	200	125	200	0,195313	0,5	0,78125	0,5	0,808	128	206,848
23	50	128	128	200	200	0,195313	0,5	0,5	0,192	0,808	49,152	206,848
24	50	128	50	197	52	0,195313	0,5	0,195313	0,808	0,192	206,848	49,152
25	50	50	200	200	200	0,195313	0,195313	0,78125	0,808	0,808	206,848	206,848
26	50	50	128	200	200	0,195313	0,195313	0,5	0,808	0,808	206,848	206,848
27	50	50	50	200	200	0,195313	0,195313	0,195313	0,808	0,808	206,848	206,848

Fonte: Própria do autor

Na tabela 4 têm-se:

- S1, S2 e S3 – SENSOR 1, SENSOR2 e SENSOR 3 respectivamente.
- RE – Roda esquerda e RD – Roda direita
- NS1 – Valor do Sensor 1 normalizado (dividido por 255) utilizado para simulação no Matlab, idem para NS2 e NS3.
- RESQ – Valor final *defuzificado* obtido no Matlab para a roda esquerda e RDIR para a roda direita.

Os valores das colunas CONVERTIDOS são valores obtidos multiplicando-se RESQ e RDIR por 255 e o resultado deve ser comparado com RESQ e RDIR apresentado na simulação do Quartus II. Pode-se observar que os valores estão muito coerentes e a pequena diferença é explicada pelos erros de arredondamento e resolução da digitalização.

Uma análise intuitiva da resposta da simulação nos indica que o resultado está coerente com o que se espera e a comparação com o Matlab confirma que o projeto responde de forma satisfatória. Por exemplo, analisando o caso em que todos os sensores estejam muito próximos de um obstáculo, caso da **Regra 1** observa-se que, para a roda esquerda o controlador envia o valor 50 e para a roda direita o valor 200. Isto significa que a roda esquerda gira anti-horária (para trás) enquanto que a direita gira no modo horário, para frente fazendo com que o robô gire em torno de seu eixo evitando colidir com os obstáculos.

7 CONCLUSÕES

O objetivo deste trabalho foi desenvolver um sistema de navegação utilizando técnicas de inteligência artificial para um robô móvel autônomo capaz de desviar de obstáculos. Para isso foi feito um algoritmo inteligente usando lógica *Fuzzy* em uma linguagem de descrição de hardware usando a placa de desenvolvimento da Altera - DE2 visando ser embarcada ao protótipo de um robô móvel.

Os subsídios e a formulação de soluções encontradas no processo do levantamento bibliográfico foram imprescindíveis para a realização da pesquisa. Nesta revisão bibliográfica foi visto que, diferentes técnicas de resolução, como Redes Neurais, Lógica *Fuzzy*, Algoritmo Genético, dentre outros para a tomada de decisão, programação e arquiteturas de projetos de robôs são usadas atualmente, visando o domínio e o aperfeiçoamento dessas técnicas.

Com os estudos realizados, uma arquitetura de projeto foi proposta para que os objetivos fossem alcançados. A metodologia de realização da programação na linguagem VHDL segue a mesma orientação observada na maioria dos artigos analisados, ou seja, o uso de módulos ou blocos de programas que permitem uma análise isolada dos mesmos e a verificação de erros mais facilmente.

Os blocos foram criados isoladamente viabilizando a necessidade de criação de *PACKAGES*, que são recursos da linguagem e permitem a recorrência várias vezes em programas. Outros recursos do software foram usados como as bibliotecas paramétricas que oferecem vários blocos de funções, entre eles às de memória.

Para validação de cada bloco realiza-se algumas simulações que fornecem a análise do funcionamento e temporização, mostrando o controle e o sincronismo de um *clock*, visando obter a arquitetura completa do sistema de navegação do robô e otimizada, antes de iniciar-se a fase de teste com um protótipo. Os resultados dessas simulações se mostram coerentes quando comparados com os resultados do processo *fuzzy* obtido pelo *toolbox*

disponível no software MATLAB.

Problemas encontrados tais como, saídas instáveis e quantidade de ciclos maior que o desejado provocou a reconstrução de alguns blocos que muitas vezes foram reformulados de forma completamente diferente da planejada inicialmente. Algumas dessas reformulações tiveram como solução a realização de blocos de controle e sincronismo por *clock*, adequados.

A habilidade do programador no domínio da linguagem de programação em VHDL, adquirida com a prática no projeto, facilita o desenvolvimento e minimiza o tempo de busca de soluções. As dificuldades na implementação de um projeto em VHDL, perpassa também, pelo cuidado que deve haver no uso de comandos ou funções que funcionam apenas para simulações, isto é, não são sintetizáveis (realizados em hardware), o que demanda um pouco mais de habilidade do programador e tempo, para a realização do algoritmo. Isto mostra como esta tecnologia, a de hardware reconfigurável, está em um processo de crescente desenvolvimento.

Outro aspecto importante de ressaltar quando se implementa hardware em dispositivos lógicos programáveis é a ocupação da área do hardware do projeto - neste projeto foi de 6% do total (1934/33.216). Outra questão importante é a resolução aritmética que se pode obter e o erro de arredondamento, o que implica em se dispor de mais memória.

A avaliação dos resultados demonstra que a aplicação do controlador Mamdani usado no projeto, para um sistema de navegação robótica é viável e apresenta resposta coerente. Das simulações do sistema resumidas na Tabela 4, observa-se que nos vinte e sete casos simulados, os resultados são intuitivamente corretos indicando que o robô pode se desviar dos obstáculos.

Apesar da facilidade de elaboração dos sistemas Mamdani por não necessitarem de um modelo matemático, mas apenas da percepção do projetista é importante notar que para sistemas que exigem um controle mais refinado é necessário avaliar sua resposta sistemas com controladores Sugenos, que para muitos projetos apresenta resposta melhores.

7.1 Trabalhos futuros

Sugere-se para continuação do trabalho, inicialmente obter o resultado com o sistema implementado no protótipo do robô de forma a fazer os ajustes necessários no programa, para que o robô consiga desviar dos obstáculos. Outra sugestão seria fazer a análise sobre os efeitos de se escolher diferentes formas de funções de pertinência e também a variação na quantidade destas funções. Aplicar um algoritmo heurístico ou Rede Neural com o objetivo de refinar a resposta de controle através de ajustes sobre as funções de pertinência, seria uma outra idéia importante. Finalmente, fazer um controlador do tipo Sugeno e comparar o desempenho com este que foi implementado.

REFERÊNCIAS

ALTERA Corporation – DE2 **Development and Education Board**. Disponível em: http://www.Altera.com/education/univ/materials/boards/de2/unv-de2-board.html. Acesso em: 03 out. 2010.

ALTERA Corporation. **Introduction to quartus II.** Disponível em: http://www.altera.com/literature/manual/intro_to_quartus2.pdf. Acesso em: 13 jun. 2011.

CABRERA, S.; SÁNCHEZ-SOLANO, C. J.; JIMÉNEZ, A. B.; BATURONE, I. Arquitectura eficiente para la implementación de hardware de sistemas de inferência difusosingenieria electrónica. **Automática y Comunicaciones**, v. 23, n. 1, p.59-66, 2003.

COSTA, E. R.; GOMES, M. L.; BIANCHI, R. A. C. Um mini robô móvel seguidor de pistas guiado por visão local. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, SBAI, 6, 2003,. Bauru. **Anais…** Bauru: SBA, 2003. p. 710-715. Disponível em: < http://fei.edu.br/~rbianchi/publications/sbai2003.pdf>. Acesso em: 12 jan. 2012.

DAIJIN, K. An implementation of *fuzzy* logic controller on the reconfigurable FPGA system. **IEEE Transactions on Industrial Electronics,** New York, v. 47, n. 3, p. 703-715, 2000. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=847911>. Acesso em: 12 jan. 2013.

FARIA, L. T. **Sistema inteligente híbrido intercomunicativo para detecção de perdas comerciais.** 2012. 112 f. Dissertação (Mestrado em Automação) – Faculdade de Engenharia, Universidade Estadual Paulista, Ilha Solteira, 2012.

GOMIDE, F. A. C.; GUDWIN, R. R. Modelagem, controle, sistemas e lógica *fuzzy*. **Revista Controle & Automação**, Campinas, v. 4, n. 3, p. 97-115, 1994.

MAMDANI, E. H. Applications of *fuzzy* algorithm for control of a simple dinamic plant. **Proceedings of the Institution of Electrical Engineers,** New York, v. 121, n. 12, p. 1585 – 1588, 1973. Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5250910>. Acesso em: 12 jan. 2013.

MINUSSI, C. R. **Lógica Nebulosa (Lógica *Fuzzy*)**. Ilha Solteira: Unesp/FE/DEEE, 2009. 119 p.

MUL COMERCIAL. Disponível em <http://multcomercial.com.br/pdf/motores/motor11.pdf>. Acesso em: 03 jan.2012.

NATIONAL. **Semiconductor**. Disponível em: <http://pdf1.alldatasheet.com/datasheet-pdf/view/8097/NSC/ADC0808.html> Acesso em: 10 out. 2011.

REZENDE, S. O. et al. **Sistemas inteligentes**: fundamentos e aplicações. Barueri: Manole, 2005. 525 p.

SANDI L.; F. A.; HEMERLY, E. M.; LAGES, W. F. Sistema para navegação e guiagem de robôs móveis autônomos. **Revista SBA Controle e Automação,** Campinas, v. 9, n. 3, p. 107-118, 1998. Disponível em: <>. Acesso em: 12 abr. 2013.

SHARP Corporation. **Manual do fabricante.** Disponível em: <http://sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf>. Acesso em: 10 ago. 2011.

SHINDO, T.; TAKITA, K.; YOKOI, H.; KAKAZU, Y. Generation of robot control circuit using EHW approach. In: PROCEEDINGS OF THE SOCIETY INSTRUMENT AND CONTROL ENGINEERS ANNUAL CONFERENCE – SICE, 38, 1999, Morioka. **Proceeding of the…** Iwate: IEEE, 1999. p. 963-966.

SUGENO, M. An introductory survey of *fuzzy* control information. **Science**, London, v. 36, p. 59-83, 1974.

SHHEIBIA, T. A. A. **Controle de um braço robótico utilizando uma abordagem de agente inteligente.** 2001. Dissertação (Mestrado) - Universidade Federal da Paraíba, Campina Grande, 2001.

TORRES, A. E.; HSU, ; CUNHA, J. P. V. S. **Introdução ao funcionamento e ao acionamento de motores DC**. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2004. 14 p. (Relatório técnico científico). Disponível em: < http://www.coep.ufrj.br/~jpaulo/MOTOR-DC-Euler.pdf>. Acesso em: 12 jan. 2013.

VAZ, A. M. **Estudo das funções de pertinência para conjuntos fuzzy utilizados em controladores semafóricos fuzzy.** 2006. 170 f. Dissertação (Mestrado em Transportes) - Faculdade de Tecnologia, Universidade de Brasília. Brasília, DF, 2006.

VUONG, P. T.; MADNI, A. M.; VOUNG, J. B. VHDL implementation for a *Fuzzy* logic controller. In: WORLD AUTOMATION CONGRESS – WAC, 2006, Budapest. **Proceedings of the…** Budapest: IEEE, 2006. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4259884>. Acesso em: 05 jan. 2013.

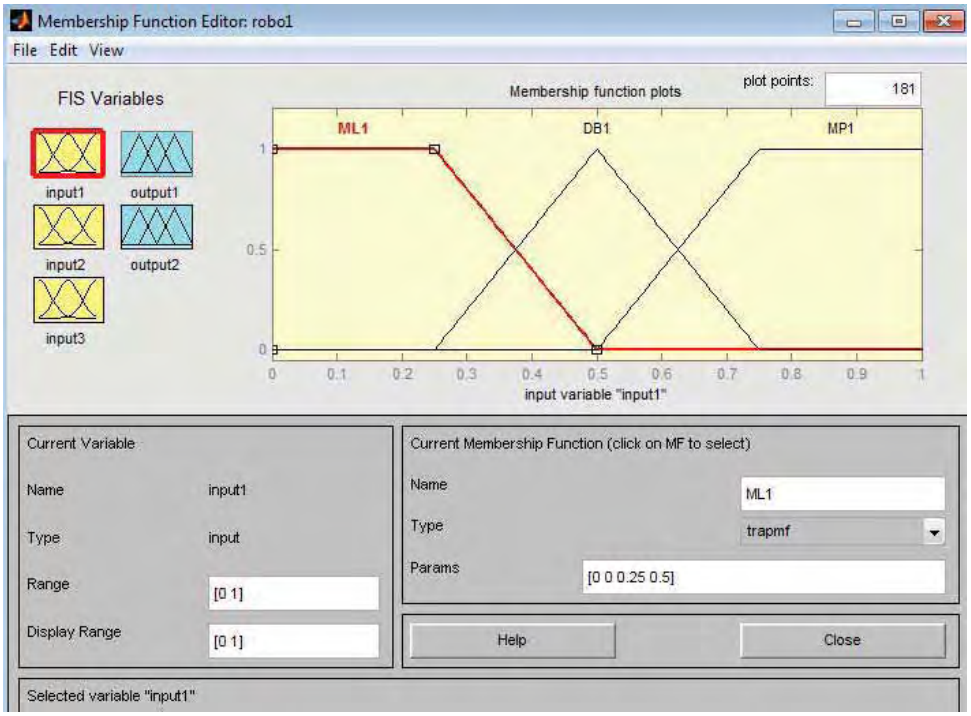
ZADEH, L. A. Outline of a new approach to the analysis of complex system and decision processes. **IEEE Transactions on Systems, Man anda Cybernetics,** New York, v. 3, n. 1, p. 28-44, 1965.

8 APÊNDICE A - TOOLBOX FUZZY – MATLAB

Os procedimentos que foram usados para validar os resultados do sistema implementado via toolbox fuzzy do Matlab , estão descritos neste apêndice.

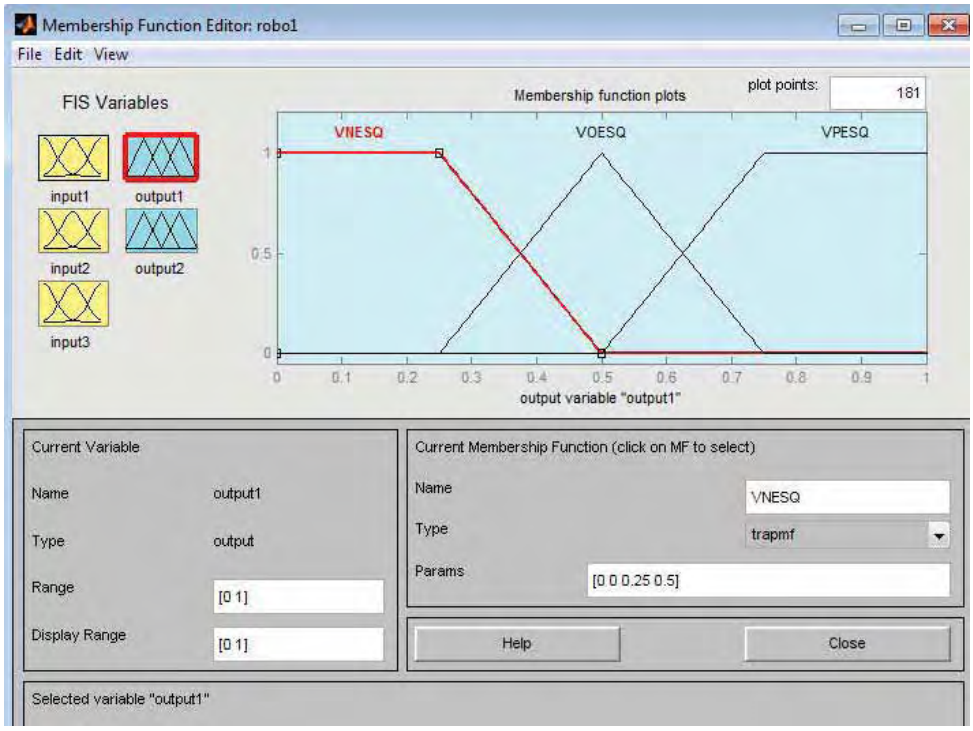
Primeiramente faz-se a definição do formato geométrico das funções de pertinência para o *fuzificador*, Figura A.1 e o *defuzificador*, Figura A.2.

Figura A.1 – Funções de pertinência- toolbox Fuzzy-Matlab para o fuzificador



Fonte: Própria do autor

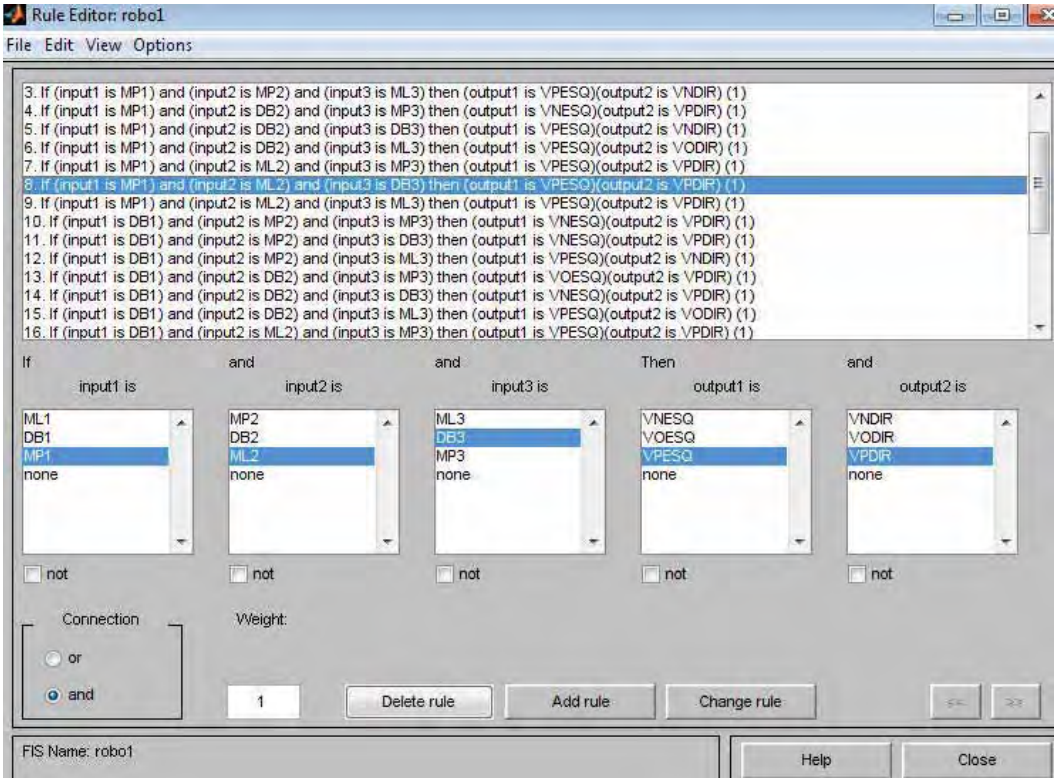
Figura A.2 – Funções de pertinência- toolbox Fuzzy-Matlab para o Defuzificador



Fonte: Própria do autor

Em seguida, são inseridas as regras, mostradas na Figura A.3 e por fim, na Figura A.4 apresenta-se o resultado do sistema inserido no *toolbox Fuzzy* do Matlab, para o mesmo caso de estudo com entradas arbitrárias, dos valores obtidos com o sistema *Fuzzy* implementado no Quartus II.

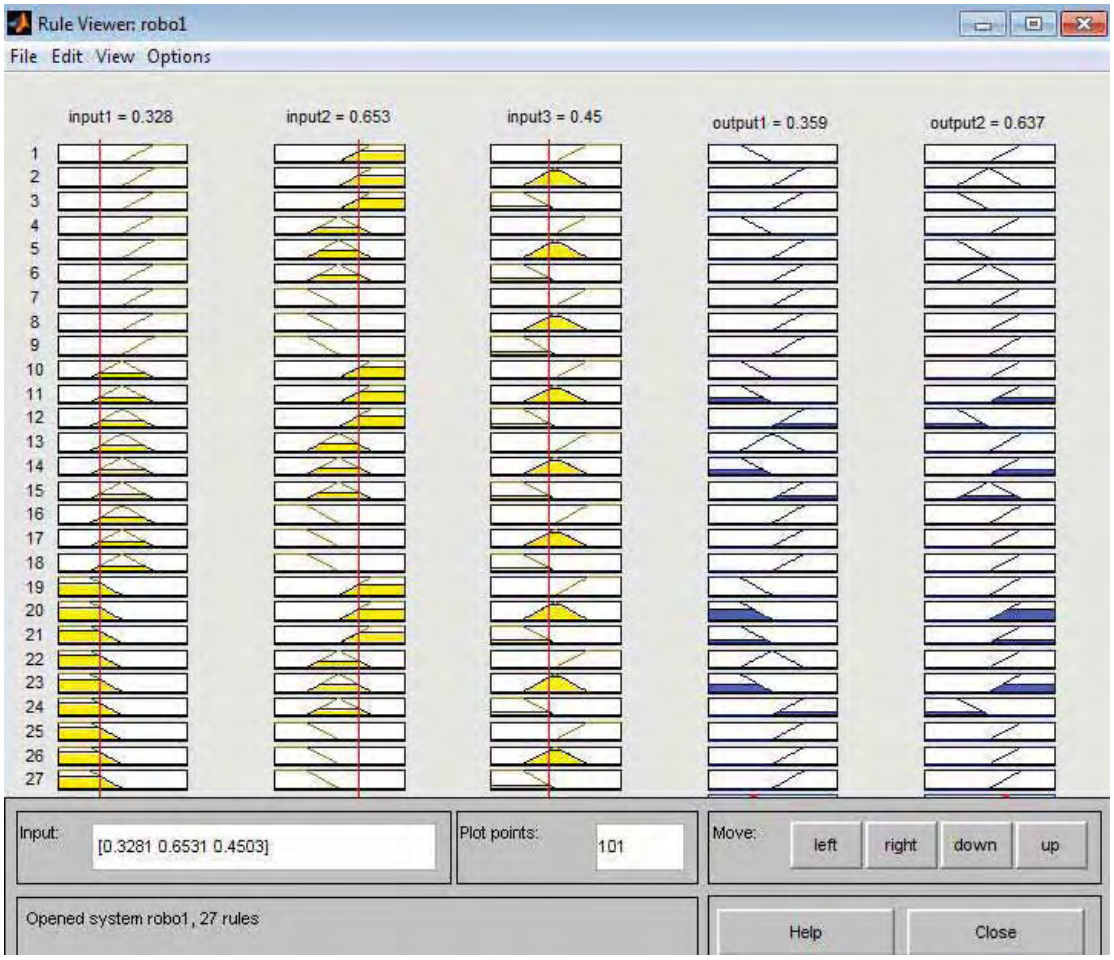
Figura A.3 - Regras - toolbox Fuzzy - Matlab



Fonte: Própria do autor

Para a construção das regras no MatLab, insere-se o vetor de entradas no Visualizador de Regras (*Rules Viewer* – Figura A.3) obtendo o resultado defuzificado. A comparação é realizada para valores normalizados dos dados, isto é, multiplicam-se os valores do Matlab por 255, valor máximo de representação das velocidades. Com os resultados obtidos na Figura A.3 alimenta-se a planilha (tabela 3) para fins de comparação com os resultados obtidos pelo protótipo implementado.

Figura A.4 - Resposta do sistema- toolbox Fuzzy para entradas arbitrárias



Fonte: Própria do autor

9 APÊNDICE B – PROGRAMAS EM VHDL

Neste apêndice são colocados os programas desenvolvidos em VHDL, usando o software de desenvolvimento da Altera –Quartus II - ver.9.1.2, encontrado no site do fabricante e disponibilizado para o público.

Bloco atraso

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
---------------------------------
ENTITY ATRASO IS
PORT(
        CLOCK,ENABLE:           IN STD_LOGIC;
        PRONTO:             OUT STD_LOGIC);
END ATRASO;
ARCHITECTURE FUNC OF ATRASO IS
BEGIN
        PROCESS(CLOCK,ENABLE)
                VARIABLE AUX:           INTEGER RANGE 0 TO 16;
                VARIABLE AUXPRONTO:  STD_LOGIC:='0';
                BEGIN
                        IF ENABLE='1' THEN
                                IF CLOCK'EVENT AND CLOCK='1' THEN
                                        IF AUX<20 THEN
                                                IF AUX=4 THEN
                                                        AUXPRONTO:='1';
                                                ELSIF AUX=19 THEN
                                                        AUXPRONTO:='0';
                                                END IF;
                                                AUX:=AUX+1;
                                        END IF;
                                END IF;
                        ELSE
                                AUX:=0;
                        END IF;
        PRONTO<=AUXPRONTO;
        END PROCESS;
END FUNC;
```

Bloco centróide

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH;
---------------------------------
ENTITY CENTROIDE1 IS
PORT(
        H1,H2,H3,H4,H5,H6:          IN INTEGER RANGE 0 TO 127;
        CLOCK:                                  IN STD_LOGIC;
        S1,S2:                              OUT   INTEGER   RANGE   0   TO
32767);
END CENTROIDE1;
---------------------------------
ARCHITECTURE FUNC OF CENTROIDE1 IS
BEGIN
PROCESS(CLOCK)
BEGIN
IF CLOCK'EVENT AND CLOCK='1' THEN
        S1<=(H1*50 + H2*127 + H3*200)/(H1 + H2 + H3);
        S2<=(H4*50 + H5*127 + H6*200)/(H4 + H5 + H6);
END IF;
END PROCESS;
END FUNC;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
---------------------------------
ENTITY CONTROLE1 IS
PORT(
        CLOCK:                      IN STD_LOGIC;
        HABILITA:           OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END CONTROLE1;
---------------------------------
ARCHITECTURE FUNC OF CONTROLE1 IS
BEGIN
PROCESS(CLOCK)
VARIABLE CONTA:         INTEGER RANGE 0 TO 7;
BEGIN
        IF CLOCK'EVENT AND CLOCK='0' THEN
                IF CONTA<=2 THEN
                        HABILITA<="000";
                ELSIF CONTA<=3 THEN
                        HABILITA<="001";
                ELSIF CONTA<=4 THEN
                        HABILITA<="010";
```

```
                ELSIF CONTA<=5 THEN
                        HABILITA<="100";
                END IF;
                IF CONTA=5 THEN
                        CONTA:=0;
                ELSE
                        CONTA:=CONTA+1;
                END IF;
        END IF;
END PROCESS;
END FUNC;
```

Package Defmatriz

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
---------------------------
PACKAGE DEFMATRIZ IS
TYPE MATRIZ1 IS ARRAY (1 TO 9) OF STD_LOGIC_VECTOR(6 DOWNTO
0);
TYPE MATRIZ2 IS ARRAY (1 TO 27) OF STD_LOGIC_VECTOR(6 DOWNTO
0);
END DEFMATRIZ;
```

Bloco controlecompleto1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.ROTACIONA2.ALL;
------------------------------
ENTITY controlecompleto1 IS
PORT(
                CLOCKIN:        IN STD_LOGIC;
                SAIDAS:         OUT  STD_LOGIC_VECTOR(6  DOWNTO
0));
END controlecompleto1;
------------------------------
ARCHITECTURE FUNC OF controlecompleto1 IS
BEGIN
PROCESS(CLOCKIN)
VARIABLE AUX1:          STD_LOGIC_VECTOR(6              DOWNTO
0):="0000000";-- Aux(0) e Aux(1) = End1 e End0; Aux(2) = Start; Aux(3)= Latch1;
Aux(4)= Latch2; Aux(5) = Latch3; Aux(6) = L4
VARIABLE CONT:          INTEGER RANGE 0 TO 500:=0;
```

```
BEGIN
        IF CLOCKIN'EVENT AND CLOCKIN='1' THEN
                IF CONT=2 THEN
                        AUX1(2):='1';
                ELSIF CONT=5 THEN
                        AUX1(2):='0';
                ELSIF CONT=105 THEN
                        AUX1(3):='1';
                ELSIF CONT=109 THEN
                        AUX1(3):='0';
                ELSIF CONT=110 THEN
                        AUX1(1):='1';
                ELSIF CONT=112 THEN
                        AUX1(2):='1';
                ELSIF CONT=115 THEN
                        AUX1(2):='0';
                ELSIF CONT=225 THEN
                        AUX1(4):='1';
                ELSIF CONT=229 THEN
                        AUX1(4):='0';
                ELSIF CONT=230 THEN
                        AUX1(1):='0';
                        AUX1(0):='1';
                ELSIF CONT=232 THEN
                        AUX1(2):='1';
                ELSIF CONT=235 THEN
                        AUX1(2):='0';
                ELSIF CONT=335 THEN
                        AUX1(5):='1';
                ELSIF CONT=339 THEN
                        AUX1(5):='0';
                ELSIF CONT=342 THEN
                        AUX1(6):='1';
                        AUX1(0):='0';
                        AUX1(1):='0';
                ELSIF CONT=345 THEN
                        AUX1(6):='0';
                        CONT:=0;
                END IF;
                CONT:=CONT+1;
                --IF CONT=14 THEN
                --      CONT:=0;
                --      AUX1:="000000";
                --END IF;
END IF;
SAIDAS<=AUX1;
END PROCESS;
```

```
END FUNC;
```

Package defmatriz2

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
----------------------------
PACKAGE DEFMATRIZ2 IS
TYPE MAT9 IS ARRAY (1 TO 9) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
TYPE MAT2 IS ARRAY (1 TO 2) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
TYPE MAT16 IS ARRAY (1 TO 16) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
TYPE MAT3 IS ARRAY (1 TO 3) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
TYPE MAT4 IS ARRAY (1 TO 4) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
TYPE MAT20 IS ARRAY (1 TO 20) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
TYPE MAT27 IS ARRAY (1 TO 27) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
END DEFMATRIZ2;
```

Block endelatches

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------
ENTITY ENDELATCHES IS
PORT(
        PROXIMO:                IN STD_LOGIC;
        ENDE:                   OUT  STD_LOGIC_VECTOR(3  DOWNTO
0));
END ENDELATCHES;
ARCHITECTURE FUNC OF ENDELATCHES IS
BEGIN
        PROCESS(PROXIMO)
                VARIABLE AUX:           INTEGER RANGE 0 TO 4;
                VARIABLE AUXEND:        STD_LOGIC_VECTOR(3  DOWNTO
0):="0001";
                BEGIN
                        IF PROXIMO'EVENT AND PROXIMO='1' THEN
                                IF AUX=0 THEN
                                        AUXEND:="0010";
                                ELSIF AUX=1 THEN
                                        AUXEND:="1100";
                                ELSIF AUX=2 THEN
                                        AUXEND:="0001";
                                --ELSIF AUX=3 THEN
                                --      AUXEND:="1000";
```

```
                    END IF;
                AUX:=AUX+1;
                IF AUX=3 THEN
                    AUX:=0;
                END IF;
                END IF;
        ENDE<=AUXEND;
        END PROCESS;
    END FUNC;
```

Bloco endereco

```
    LIBRARY IEEE;
    USE IEEE.STD_LOGIC_1164.ALL;
    USE IEEE.STD_LOGIC_UNSIGNED.ALL;
    ---------------------------------
    ENTITY ENDERECO IS
    PORT(
        PROXIMO:            IN STD_LOGIC;
        ENDE:           OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
    END ENDERECO;
    ARCHITECTURE FUNC OF ENDERECO IS
    BEGIN
        PROCESS(PROXIMO)
            VARIABLE AUX:       INTEGER RANGE 0 TO 4;
            VARIABLE AUXEND:    STD_LOGIC_VECTOR(1  DOWNTO
0);
            BEGIN
                IF PROXIMO'EVENT AND PROXIMO='1' THEN
                    IF AUX=0 THEN
                        AUXEND:="01";
                    ELSIF AUX=1 THEN
                        AUXEND:="10";
                    ELSIF AUX=2 THEN
                        AUXEND:="00";
                    END IF;
                AUX:=AUX+1;
                IF AUX=3 THEN
                    AUX:=0;
                END IF;
                END IF;
        ENDE<=AUXEND;
        END PROCESS;
    END FUNC;
```

Bloco maximo2

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.MAIOR4.ALL;
USE WORK.MAIOR5.ALL;
USE WORK.MAIOR6.ALL;
USE WORK.DEFMATRIZ2.ALL;
---------------------------
ENTITY MAXIMO2 IS
PORT(
        ENT1:       IN MAT4;
        ENT2:       IN MAT3;
        ENT3:       IN MAT20;
        CLOCK:              IN STD_LOGIC;
        SAI:        OUT MAT3);
END MAXIMO2;
-----------------------------
ARCHITECTURE FUNC OF MAXIMO2 IS
BEGIN
PROCESS(CLOCK)
BEGIN
IF CLOCK'EVENT AND CLOCK='1' THEN
        SAI(1)<=MAX4(ENT1);
        SAI(2)<=MAX5(ENT2);
        SAI(3)<=MAX6(ENT3);
END IF;
END PROCESS;
END FUNC;
```

Bloco inicia

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------
ENTITY INICIA IS
PORT(
        CLOCK,RESET:                    IN STD_LOGIC;
        START:                                  OUT STD_LOGIC);
END INICIA;
ARCHITECTURE FUNC OF INICIA IS
BEGIN
        PROCESS(CLOCK,RESET)
                VARIABLE AUX:           INTEGER RANGE 0 TO 4;
```

```
                VARIABLE AUXSTART:     STD_LOGIC;
                BEGIN
                        IF RESET='1' THEN
                                IF CLOCK'EVENT AND CLOCK='1' THEN
                                        IF AUX<4 THEN
                                                IF AUX=2 THEN
                                                        AUXSTART:='1';
                                                ELSIF AUX=3 THEN
                                                        AUXSTART:='0';
                                                END IF;
                                                AUX:=AUX+1;
                                        END IF;
                                END IF;
                        ELSE
                                AUX:=2;
                        END IF;
                START<=AUXSTART;
                END PROCESS;
        END FUNC;
```

Package maior1

```
        LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        USE WORK.DEFMATRIZ2.ALL;
        ----------------------------
        PACKAGE MAIOR1 IS
        FUNCTION        MAX1(SIGNAL        ENTRA:        MAT9)        RETURN
STD_LOGIC_VECTOR;
        END MAIOR1;
        ----------------------------
        PACKAGE BODY MAIOR1 IS
        FUNCTION MAX1(SIGNAL ENTRA: MAT9)
                RETURN STD_LOGIC_VECTOR IS
        VARIABLE AUX:           MAT9;
        BEGIN
        AUX(1):=ENTRA(1);
        AUX(2):=ENTRA(2);
        AUX(3):=ENTRA(3);
        AUX(4):=ENTRA(4);
        AUX(5):=ENTRA(5);
        AUX(6):=ENTRA(6);
        AUX(7):=ENTRA(7);
        AUX(8):=ENTRA(8);
        AUX(9):=ENTRA(9);
```

```
IF AUX(2)>AUX(1) THEN
        AUX(1):=AUX(2);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(4)>AUX(3) THEN
        AUX(3):=AUX(4);
ELSE AUX(3):=AUX(3);
END IF;
IF AUX(6)>AUX(5) THEN
        AUX(5):=AUX(6);
ELSE AUX(5):=AUX(5);
END IF;
IF AUX(8)>AUX(7) THEN
        AUX(7):=AUX(8);
ELSE AUX(7):=AUX(7);
END IF;

IF AUX(3)>AUX(1) THEN
        AUX(1):=AUX(3);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(7)>AUX(5) THEN
        AUX(5):=AUX(7);
ELSE AUX(5):=AUX(5);
END IF;

IF AUX(5)>AUX(1) THEN
        AUX(1):=AUX(5);
ELSE AUX(1):=AUX(1);
END IF;

IF AUX(9)>AUX(1) THEN
        AUX(1):=AUX(9);
ELSE AUX(1):=AUX(1);
END IF;

RETURN (AUX(1));
END MAX1;
END MAIOR1;
```

Package maior2

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```
USE WORK.DEFMATRIZ2.ALL;
----------------------------
PACKAGE MAIOR2 IS
FUNCTION MAX2(SIGNAL ENTRA: MAT2) RETURN
STD_LOGIC_VECTOR;
END MAIOR2;
----------------------------
PACKAGE BODY MAIOR2 IS
FUNCTION MAX2(SIGNAL ENTRA: MAT2)
        RETURN STD_LOGIC_VECTOR IS
VARIABLE AUX:           MAT2;
BEGIN
AUX(1):=ENTRA(1);
AUX(2):=ENTRA(2);
IF AUX(2)>AUX(1) THEN
        AUX(1):=AUX(2);
ELSE AUX(1):=AUX(1);
END IF;
RETURN (AUX(1));
END MAX2;
END MAIOR2;
```

Package maior3

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.DEFMATRIZ2.ALL;
----------------------------
PACKAGE MAIOR3 IS
FUNCTION MAX3(SIGNAL ENTRA: MAT16) RETURN
STD_LOGIC_VECTOR;
END MAIOR3;
----------------------------
PACKAGE BODY MAIOR3 IS
FUNCTION MAX3(SIGNAL ENTRA: MAT16)
        RETURN STD_LOGIC_VECTOR IS
VARIABLE AUX:           MAT16;
BEGIN
AUX(1):=ENTRA(1);
AUX(2):=ENTRA(2);
AUX(3):=ENTRA(3);
AUX(4):=ENTRA(4);
AUX(5):=ENTRA(5);
AUX(6):=ENTRA(6);
AUX(7):=ENTRA(7);
AUX(8):=ENTRA(8);
```

```
AUX(9):=ENTRA(9);
AUX(10):=ENTRA(10);
AUX(11):=ENTRA(11);
AUX(12):=ENTRA(12);
AUX(13):=ENTRA(13);
AUX(14):=ENTRA(14);
AUX(15):=ENTRA(15);
AUX(16):=ENTRA(16);

IF AUX(2)>AUX(1) THEN
        AUX(1):=AUX(2);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(4)>AUX(3) THEN
        AUX(3):=AUX(4);
ELSE AUX(3):=AUX(3);
END IF;
IF AUX(6)>AUX(5) THEN
        AUX(5):=AUX(6);
ELSE AUX(5):=AUX(5);
END IF;
IF AUX(8)>AUX(7) THEN
        AUX(7):=AUX(8);
ELSE AUX(7):=AUX(7);
END IF;
IF AUX(10)>AUX(9) THEN
        AUX(9):=AUX(10);
ELSE AUX(9):=AUX(9);
END IF;
IF AUX(12)>AUX(11) THEN
        AUX(11):=AUX(12);
ELSE AUX(11):=AUX(11);
END IF;
IF AUX(14)>AUX(13) THEN
        AUX(13):=AUX(14);
ELSE AUX(13):=AUX(13);
END IF;
IF AUX(16)>AUX(15) THEN
        AUX(15):=AUX(16);
ELSE AUX(15):=AUX(15);
END IF;

IF AUX(3)>AUX(1) THEN
        AUX(1):=AUX(3);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(7)>AUX(5) THEN
```

```
            AUX(5):=AUX(7);
        ELSE AUX(5):=AUX(5);
        END IF;
        IF AUX(11)>AUX(9) THEN
            AUX(9):=AUX(11);
        ELSE AUX(9):=AUX(9);
        END IF;
        IF AUX(15)>AUX(13) THEN
            AUX(13):=AUX(15);
        ELSE AUX(13):=AUX(13);
        END IF;

        IF AUX(5)>AUX(1) THEN
            AUX(1):=AUX(5);
        ELSE AUX(1):=AUX(1);
        END IF;
        IF AUX(13)>AUX(9) THEN
            AUX(9):=AUX(13);
        ELSE AUX(9):=AUX(9);
        END IF;

        IF AUX(9)>AUX(1) THEN
            AUX(1):=AUX(9);
        ELSE AUX(1):=AUX(1);
        END IF;
        RETURN (AUX(1));
        END MAX3;
        END MAIOR3;
```

Package maior4

```
        LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        USE WORK.DEFMATRIZ2.ALL;
        ----------------------------
        PACKAGE MAIOR4 IS
        FUNCTION MAX4(SIGNAL ENTRA: MAT4) RETURN
STD_LOGIC_VECTOR;
        END MAIOR4;
        ----------------------------
        PACKAGE BODY MAIOR4 IS
        FUNCTION MAX4(SIGNAL ENTRA: MAT4)
            RETURN STD_LOGIC_VECTOR IS
        VARIABLE AUX:       MAT4;
        BEGIN
        AUX(1):=ENTRA(1);
```

```
        AUX(2):=ENTRA(2);
        AUX(3):=ENTRA(3);
        AUX(4):=ENTRA(4);

        IF AUX(2)>AUX(1) THEN
                AUX(1):=AUX(2);
        ELSE AUX(1):=AUX(1);
        END IF;
        IF AUX(4)>AUX(3) THEN
                AUX(3):=AUX(4);
        ELSE AUX(3):=AUX(3);
        END IF;

        IF AUX(3)>AUX(1) THEN
                AUX(1):=AUX(3);
        ELSE AUX(1):=AUX(1);
        END IF;

        RETURN (AUX(1));
        END MAX4;
        END MAIOR4;
```

Package maior5

```
        LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        USE WORK.DEFMATRIZ2.ALL;
        ----------------------------
        PACKAGE MAIOR5 IS
        FUNCTION MAX5(SIGNAL ENTRA: MAT3) RETURN
STD_LOGIC_VECTOR;
        END MAIOR5;
        ----------------------------
        PACKAGE BODY MAIOR5 IS
        FUNCTION MAX5(SIGNAL ENTRA: MAT3)
                RETURN STD_LOGIC_VECTOR IS
        VARIABLE AUX:           MAT3;
        BEGIN
        AUX(1):=ENTRA(1);
        AUX(2):=ENTRA(2);
        AUX(3):=ENTRA(3);
        IF AUX(2)>AUX(1) THEN
                AUX(1):=AUX(2);
        ELSE AUX(1):=AUX(1);
        END IF;
```

```
        IF AUX(3)>AUX(1) THEN
                AUX(1):=AUX(3);
        ELSE AUX(1):=AUX(1);
        END IF;
        RETURN (AUX(1));
        END MAX5;
        END MAIOR5;
```

Package maior6

```
        LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        USE WORK.DEFMATRIZ2.ALL;
        ----------------------------
        PACKAGE MAIOR6 IS
        FUNCTION MAX6(SIGNAL ENTRA: MAT20) RETURN
STD_LOGIC_VECTOR;
        END MAIOR6;
        ----------------------------
        PACKAGE BODY MAIOR6 IS
        FUNCTION MAX6(SIGNAL ENTRA: MAT20)
                RETURN STD_LOGIC_VECTOR IS
        VARIABLE AUX:           MAT20;
        BEGIN
        AUX(1):=ENTRA(1);
        AUX(2):=ENTRA(2);
        AUX(3):=ENTRA(3);
        AUX(4):=ENTRA(4);
        AUX(5):=ENTRA(5);
        AUX(6):=ENTRA(6);
        AUX(7):=ENTRA(7);
        AUX(8):=ENTRA(8);
        AUX(9):=ENTRA(9);
        AUX(10):=ENTRA(10);
        AUX(11):=ENTRA(11);
        AUX(12):=ENTRA(12);
        AUX(13):=ENTRA(13);
        AUX(14):=ENTRA(14);
        AUX(15):=ENTRA(15);
        AUX(16):=ENTRA(16);
        AUX(17):=ENTRA(17);
        AUX(18):=ENTRA(18);
        AUX(19):=ENTRA(19);
        AUX(20):=ENTRA(20);
```

```
IF AUX(2)>AUX(1) THEN
        AUX(1):=AUX(2);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(4)>AUX(3) THEN
        AUX(3):=AUX(4);
ELSE AUX(3):=AUX(3);
END IF;
IF AUX(6)>AUX(5) THEN
        AUX(5):=AUX(6);
ELSE AUX(5):=AUX(5);
END IF;
IF AUX(8)>AUX(7) THEN
        AUX(7):=AUX(8);
ELSE AUX(7):=AUX(7);
END IF;
IF AUX(10)>AUX(9) THEN
        AUX(9):=AUX(10);
ELSE AUX(9):=AUX(9);
END IF;
IF AUX(12)>AUX(11) THEN
        AUX(11):=AUX(12);
ELSE AUX(11):=AUX(11);
END IF;
IF AUX(14)>AUX(13) THEN
        AUX(13):=AUX(14);
ELSE AUX(13):=AUX(13);
END IF;
IF AUX(16)>AUX(15) THEN
        AUX(15):=AUX(16);
ELSE AUX(15):=AUX(15);
END IF;
IF AUX(18)>AUX(17) THEN
        AUX(17):=AUX(18);
ELSE AUX(17):=AUX(17);
END IF;
IF AUX(20)>AUX(19) THEN
        AUX(19):=AUX(20);
ELSE AUX(19):=AUX(19);
END IF;

IF AUX(3)>AUX(1) THEN
        AUX(1):=AUX(3);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(7)>AUX(5) THEN
        AUX(5):=AUX(7);
```

```
ELSE AUX(5):=AUX(5);
END IF;
IF AUX(11)>AUX(9) THEN
        AUX(9):=AUX(11);
ELSE AUX(9):=AUX(9);
END IF;
IF AUX(15)>AUX(13) THEN
        AUX(13):=AUX(15);
ELSE AUX(13):=AUX(13);
END IF;
IF AUX(19)>AUX(17) THEN
        AUX(17):=AUX(19);
ELSE AUX(17):=AUX(17);
END IF;

IF AUX(5)>AUX(1) THEN
        AUX(1):=AUX(5);
ELSE AUX(1):=AUX(1);
END IF;
IF AUX(13)>AUX(9) THEN
        AUX(9):=AUX(13);
ELSE AUX(9):=AUX(9);
END IF;


IF AUX(9)>AUX(1) THEN
        AUX(1):=AUX(9);
ELSE AUX(1):=AUX(1);
END IF;

IF AUX(17)>AUX(1) THEN
        AUX(1):=AUX(17);
ELSE AUX(1):=AUX(1);
END IF;
RETURN (AUX(1));
END MAX6;
END MAIOR6;
```

Bloco maximo1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.MAIOR1.ALL;
USE WORK.MAIOR2.ALL;
USE WORK.MAIOR3.ALL;
USE WORK.DEFMATRIZ2.ALL;
```

```
----------------------------
ENTITY MAXIMO1 IS
PORT(
        ENT1:           IN MAT9;
        ENT2:           IN MAT2;
        ENT3:           IN MAT16;
        CLOCK:                  IN STD_LOGIC;
        SAI:            OUT MAT3);
END MAXIMO1;
-----------------------------
ARCHITECTURE FUNC OF MAXIMO1 IS
BEGIN
PROCESS(CLOCK)
BEGIN
IF CLOCK'EVENT AND CLOCK='1' THEN
        SAI(1)<=MAX1(ENT1);
        SAI(2)<=MAX2(ENT2);
        SAI(3)<=MAX3(ENT3);
END IF;
END PROCESS;
END FUNC;
```

Bloco timing

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
---------------------------------
ENTITY TIMING IS
PORT(
        RESET,CLOCK:                    IN STD_LOGIC;
        PROX,INICIA,PRONTO:             OUT STD_LOGIC);
END TIMING;
---------------------------------
ARCHITECTURE FUNC OF TIMING IS
BEGIN
PROCESS(CLOCK,RESET)
        VARIABLE CONTA: INTEGER RANGE 0 TO 200;
        BEGIN
                IF RESET='1' THEN
                        CONTA:=0;PRONTO<='0';
                        ELSIF CLOCK'EVENT AND CLOCK='1' THEN
                                IF CONTA<2 THEN
                                        PROX<='0';INICIA<='0';PRONTO<='0';
```

```
                                    ELSIF CONTA<4 THEN
                                            INICIA<='1';
                                    ELSIF CONTA<5 THEN
                                            INICIA<='0';
                                    --ELSIF CONTA<7 THEN
                                            --PROX<='1';
                                    --ELSIF CONTA<9 THEN
                                            --PROX<='0';
                                    --ELSIF CONTA<35 THEN
                                    --      PRONTO<='1';
                                    ELSIF CONTA>39 THEN
                                            PROX<='0';
                                    ELSIF CONTA>37 THEN
                                            PROX<='1';
                                    ELSIF CONTA>35 THEN
                                            PRONTO<='1';
                                    ELSIF CONTA=200 THEN
                                            CONTA:=0;
                                    END IF;
                                    CONTA:=CONTA+1;
                            END IF;
                    END PROCESS;
            END FUNC;
```

Package menor1

```
        LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        ----------------------------
        PACKAGE MENOR1 IS
        FUNCTION MENOR(SIGNAL IN1,IN2,IN3: STD_LOGIC_VECTOR) RETURN
STD_LOGIC_VECTOR;
        END MENOR1;
        ----------------------------
        PACKAGE BODY MENOR1 IS
        FUNCTION MENOR(SIGNAL IN1,IN2,IN3: STD_LOGIC_VECTOR)
                RETURN STD_LOGIC_VECTOR IS
        VARIABLE        VAR1,VAR2,VAR3,VMENOR:        STD_LOGIC_VECTOR(6
DOWNTO 0);
        BEGIN
        VAR1:=IN1;VAR2:=IN2;VAR3:=IN3;
                IF VAR2<VAR3 THEN
                        IF VAR1<VAR2 THEN
                                VMENOR:=VAR1;
                        ELSE VMENOR:=VAR2;
```

```
                END IF;
        ELSE
                IF VAR1<VAR3 THEN
                        VMENOR:=VAR1;
                ELSE VMENOR:=VAR3;
                END IF;
        END IF;
RETURN (VMENOR);
END MENOR;
END MENOR1;
```

Bloco minimo1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.DEFMATRIZ.ALL;
USE WORK.MENOR1.ALL;
--------------------------------
ENTITY MINIMO1 IS
PORT(
        ENTRADA:            IN MATRIZ1;
        CLOCK:                      IN STD_LOGIC;
        SAI:                OUT MATRIZ2);
END MINIMO1;
--------------------------------
ARCHITECTURE FUNC OF MINIMO1 IS
BEGIN
PROCESS(CLOCK)
BEGIN
        IF CLOCK'EVENT AND CLOCK='1' THEN
                SAI(1)<=MENOR(ENTRADA(1),ENTRADA(4),ENTRADA(7));
                SAI(2)<=MENOR(ENTRADA(1),ENTRADA(4),ENTRADA(8));
                SAI(3)<=MENOR(ENTRADA(1),ENTRADA(4),ENTRADA(9));
                SAI(4)<=MENOR(ENTRADA(1),ENTRADA(5),ENTRADA(7));
                SAI(5)<=MENOR(ENTRADA(1),ENTRADA(5),ENTRADA(8));
                SAI(6)<=MENOR(ENTRADA(1),ENTRADA(5),ENTRADA(9));
                SAI(7)<=MENOR(ENTRADA(1),ENTRADA(6),ENTRADA(7));
                SAI(8)<=MENOR(ENTRADA(1),ENTRADA(6),ENTRADA(8));
                SAI(9)<=MENOR(ENTRADA(1),ENTRADA(6),ENTRADA(9));
                SAI(10)<=MENOR(ENTRADA(2),ENTRADA(4),ENTRADA(7));
                SAI(11)<=MENOR(ENTRADA(2),ENTRADA(4),ENTRADA(8));
                SAI(12)<=MENOR(ENTRADA(2),ENTRADA(4),ENTRADA(9));
                SAI(13)<=MENOR(ENTRADA(2),ENTRADA(5),ENTRADA(7));
                SAI(14)<=MENOR(ENTRADA(2),ENTRADA(5),ENTRADA(8));
```

```
                SAI(15)<=MENOR(ENTRADA(2),ENTRADA(5),ENTRADA(9));
                SAI(16)<=MENOR(ENTRADA(2),ENTRADA(6),ENTRADA(7));
                SAI(17)<=MENOR(ENTRADA(2),ENTRADA(6),ENTRADA(8));
                SAI(18)<=MENOR(ENTRADA(2),ENTRADA(6),ENTRADA(9));
                SAI(19)<=MENOR(ENTRADA(3),ENTRADA(4),ENTRADA(7));
                SAI(20)<=MENOR(ENTRADA(3),ENTRADA(4),ENTRADA(8));
                SAI(21)<=MENOR(ENTRADA(3),ENTRADA(4),ENTRADA(9));
                SAI(22)<=MENOR(ENTRADA(3),ENTRADA(5),ENTRADA(7));
                SAI(23)<=MENOR(ENTRADA(3),ENTRADA(5),ENTRADA(8));
                SAI(24)<=MENOR(ENTRADA(3),ENTRADA(5),ENTRADA(9));
                SAI(25)<=MENOR(ENTRADA(3),ENTRADA(6),ENTRADA(7));
                SAI(26)<=MENOR(ENTRADA(3),ENTRADA(6),ENTRADA(8));
                SAI(27)<=MENOR(ENTRADA(3),ENTRADA(6),ENTRADA(9));
        END IF;
    END PROCESS;
    END FUNC;
```

Bloco testelatch

```
    LIBRARY IEEE;
    USE IEEE.STD_LOGIC_1164.ALL;
    USE IEEE.STD_LOGIC_UNSIGNED.ALL;
    ----------------------------
    ENTITY TESTELATCH IS
    PORT(
        PRONTO,RESET:                                   IN STD_LOGIC;
        OLATCHES:                           OUT         STD_LOGIC_VECTOR(3
DOWNTO 0));
    END TESTELATCH;
    ----------------------------
    ARCHITECTURE FUNC OF TESTELATCH IS
    BEGIN
    PROCESS(PRONTO,RESET)
    VARIABLE CONTA: INTEGER RANGE 0 TO 4;
    VARIABLE AUX:   STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
    BEGIN
        IF RESET='1' THEN
            OLATCHES<="0000";
            ELSIF PRONTO'EVENT AND PRONTO='1' THEN
                IF CONTA=1 THEN
                    AUX:="0001";
                ELSIF CONTA=2 THEN
                    AUX:="0010";
                ELSIF CONTA=3 THEN
```

```
                    AUX:="0100";
                ELSIF CONTA=4 THEN
                    AUX:="1000";
                    CONTA:=0;
                END IF;
                CONTA:=CONTA+1;
        END IF;
OLATCHES<=AUX;
END PROCESS;
END FUNC;
```