ALLAN MASSAHUD DE CARVALHO

ROBÔ EXPLORADOR

Monografia da disciplina Projeto Orientado apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de bacharel em Ciência da Computação

Orientador

Prof. Wilian Soares Lacerda

LAVRAS
MINAS GERAIS - BRASIL
2001

ALLAN MASSAHUD DE CARVALHO

ROBÔ EXPLORADOR

Monografia da disciplina Projeto Orientado apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação

APROVADA em 28 de junho de 2001.

Prof. _____
(João Carlos Giacomin)

Prof. _____
(Nadiel Massahud)

Prof. _____
UFLA
(Orientador - Wilian Soares Lacerda)

LAVRAS
MINAS GERAIS - BRASIL

Dedicatória

Aos meus pais, meus irmãos e meus amados sobrinhos, Emily, Matheus e Gabrielle.

Agradecimentos

A parte mais agradável em qualquer trabalho é esta, a qual temos oportunidade de poder agradecer às pessoas que cordialmente tanto nos ajudaram.

Aos Professores Wilian Soares Lacerda e João Carlos Giacomin, pela competente orientação durante este trabalho.

Ao meu amigo Nestor Ignácio Serafim, pela ajuda na montagem da parte mecânica do robô.

Aos funcionários da serralheria da Ufla, pelo apoio na confecção da plataforma do robô.

Resumo

Um problema que tem segurado os avanços da robótica é a tarefa de criar mecanismos que possam duplicar precisamente os sentidos em uma máquina. Quanto melhor um robô puder sentir e avaliar os eventos no seu meio ambiente, mais autônomo ele se tornará. A interação de sentido necessária para realizar até mesmo a mais simples das ações é bem mais complicada do que a maioria das pessoas pensa. Atualmente os sensores dos robôs estão limitados a cinco tipos: táctil, distância, proximidade, acústica e visual.

Este trabalho descreve um sistema autônomo para controle de robôs exploradores, que deverá locomover-se e desviar-se de obstáculos automaticamente.

O sistema completo consiste em uma unidade de comando, localizada no robô, que decidirá, a partir dos sinais dos sensores do robô, qual o caminho a ser seguido, e uma unidade de acionamento dos dois motores, que os comandará de modo a movimentar o robô a qualquer direção.

SUMÁRIO

Introdução

Desde o começo dos tempos, o homem foi aproveitando tudo o que a natureza colocou ao seu redor. Assim, para facilitar seu trabalho empregou diversas espécies animais, tais como bois, cavalos, burros, cães, gatos e outros.

A competição feroz dos nossos tempos exige o aumento da produtividade, o que obriga a substituir os animais por máquinas e comandos. Os robôs atuais são resultado da reunião de necessidades, idéias e trabalhos, cada qual de procedência muito variada, podendo-se agrupá-los em três grandes blocos:

- Imaginação;
- A necessidade de automação;
- A experiência científica;

A imaginação tem um componente lúdico especial referente à idéia, sempre latente no homem, de criar, imitando a natureza, em geral, e reproduzindo o comportamento do homem, em particular. Assim, os autômatos, alguns dos quais podem ser encontrados nos museus, podem ser considerados antepassados dos Robôs. Quase todos construídos por "mecânicos de precisão" da época, especialmente relojoeiros, que os faziam para satisfação própria ou por diversão da nobreza.

A idéia, a nível popular, sobre os robôs é que eles são máquinas com comportamento humano, o que se deve aos livros e filmes de ficção científica. E até o nome robô tem essa origem; seu criador foi o escritor tcheco Karel Capek, primeiro na sua obra Opilek, e três anos depois, em 1920, na conhecida obra de teatro R.U.R. *Rossum's Universal Robots*. Nela, um homem fabricava máquinas com forma humana, para que servissem como escravos, e denominava-as de *robots* que, no tcheco é *robota* significando escravo.

A necessidade de automação é um segundo fator que deu origem aos robôs. Aqui se apresentam as técnicas fundamentais:

1ª A estrutura mecânica, baseada em vários corpos articulados entre si foi motivada pelos telemanipuladores. Que adquiriram grande relevância na indústria nuclear.

2ª O controle dos movimentos que seguiu os passos das máquinas - ferramenta com controle numérico. Estas integravam, em seu funcionamento, a forma de programação dos teares.

A investigação e a experimentação científica compõe o terceiro bloco que deu origem aos robôs. Sua importância na evolução destes é muito grande, pois as máquinas que se podem chamar de robôs, são o resultado do extraordinário interesse dos cientistas em experimentar suas teorias e reproduzir determinadas características do homem.

Os robôs simplesmente apareceram em 1961, quando a General Motors comprou um robô chamado Unimate para realizar serviços de fundição em molde. Ele podia aprender e realizar 180 passos, mas era pouco mais do que um braço mecânico. Um controlador humano guiava o robô em cada um dos passos, e o robô gravava cada um deles. Depois que os passos estavam gravados, o Unimate iria reproduzi-los através da sua memória.

No meio dos anos 60, cientistas de Stanford, MIT e SRI International começaram a realizar experiências com robôs que incorporavam inteligência artificial e câmeras de TV para guiar os seus braços. Depois de muito refinamento, estes novos robôs eram capazes de montar itens como a bomba de água de um automóvel.

Nos anos 70, os robôs já possuíam alguma credibilidade, mas foi somente nos anos 80 que eles se estabeleceram. A indústria automobilística descobriu que os robôs podiam substituir os humanos em muitas tarefas cansativas e perigosas.

Em 1997 a NASA enviou até Marte o seu Robô Explorador, Mars Path Finder, que tinha o objetivo de fazer investigações da geologia e morfologia da superfície daquele Planeta, e a centenas de metros abaixo do solo, a geo-química e petrologia do solo e das rochas, as propriedades magnéticas do solo, tal como as propriedades magnéticas da poeira.

2. Descrição do hardware

O robô proposto consiste das seguintes partes: A) Uma máquina de estados que decide o caminho a ser seguido, B) um circuito que observa as posições de cada roda do robô a cada instante de tempo T e retorna o deslocamento do robô, C) um gerador de sinais para acionamento das fases do motor D), um circuito que amplifica a corrente dos sinais gerados para o motor e E) dois motores de passo, para a locomoção do robô.

Os itens “A”, “B” e “C” são agrupados em um diagrama de blocos que será chamado de Circuito Comando e o item “D” nomeado de Circuito de Potência. A figura 1 ilustra o esquema do hardware do robô.

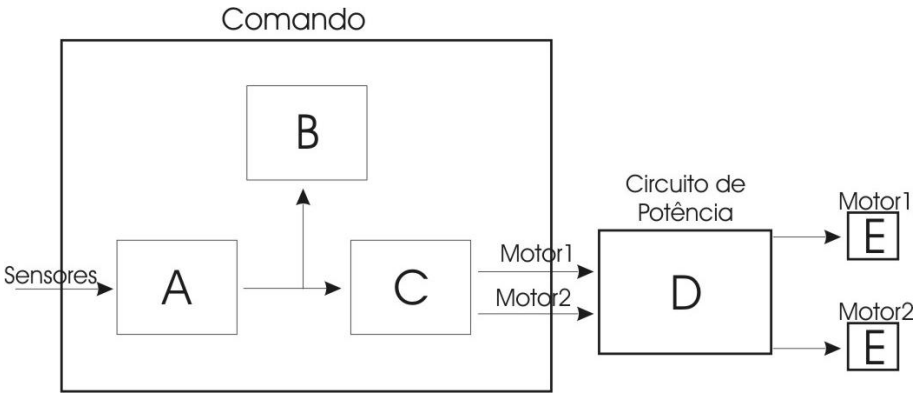


Figura 1 – Hardware do Robô

2.1 Kit Altera

O Kit Altera® compõe se de Dispositivos de Programação Lógica (Programmable Logic Device - PLDs). Os PLDs são compostos por Circuitos Integrados (CIs) configuráveis pelo usuário e são usados para implementar funções lógicas.

No início dos anos 80, simples PLDs eram usados tipicamente para integrar múltiplos dispositivos lógicos discretos e seus projetos eram tipicamente expressos usando equações Booleanas. Hoje em dia, a maioria dos PLDs são empregados para a integração de sistemas e são freqüentemente a alternativa preferida para substituir CIs de aplicações específicas (ASICs) ou aplicações produtos padrões (ASSPs). Um ASIC é um projeto de aplicação individual e um ASSP é um dispositivo que implementa uma função específica.

Os PLDs são todos circuitos para implementação de circuitos de lógica digital, incluindo dispositivos 20-pinos PAL/GAL, field programmable gate arrays (FPGAs) e complexos PLDs (CPLDs). Os PLDs são oferecidos em diferentes arquiteturas, e uma variedade de elementos de memória estão disponíveis para configuração dos dispositivos.

2.1.1 PLD Altera® FLEX 10K

Os dispositivos FLEX 10K são os primeiros PLDs industriais. Baseados em elementos CMOS SRAM. A arquitetura do Flexible Logic Element Matrix (FLEX) incorpora todas as características necessárias para implementar habituais megafunções de vetor de portas. Com mais de 250000 portas, a família FLEX 10K fornece densidade, velocidade e características para integrar sistemas, incluindo barramento de 32 bits, em um simples aparelho.

Os aparelhos FLEX 10K são configuráveis, e previamente testados. Como resultado o projetista não precisa se preocupar em gerar vetores teste, ao invés disso, pode focalizar na simulação e verificação de projeto.

O chip FLEX 10K utilizado encontra-se no UP 1 Education Board [11], que é uma plataforma baseada em dois dispositivos da Altera® das famílias: MAX® 7000 e FLEX® 10K. Seu simples projeto, quando usado com o software MAX+PLUS II, fornece uma excelente plataforma para o aprendizado de lógica digital usando ferramentas de desenvolvimento de alto nível e PLDs.

Os PLDs do UP 1 Education Board, EPM7128S (MAX) e o EPF10K20 (FLEX) podem ser programados com o cabo Byte Blaster, conectando-o no conector de entrada JTAG_IN da Kit Altera e na porta paralela de um microcomputador. O último também pode ser programado através de uma EPROM, chamada EPC1.

O EPF10K20 é baseado em elementos SRAM reconfiguráveis. Juntamente com o EPF10K20, foram utilizados os seguintes componentes do UP 1 Education Board:

- Oscilador de cristal de 25.175MHz, utilizado para gerar o clock ;
- O botão FLEX_PB1, utilizado para dar Reset no sistema ;
- As três barras de terminais nomeados FLEX_EXPAN_A, FLEX_EXPAN_B e FLEX_EXPAN_C utilizados para interfacear as entradas dos sensores e as saídas dos sinais para acionar as bobinas dos motores.

A figura 2 ilustra o diagrama de blocos do UP1 Education Board.

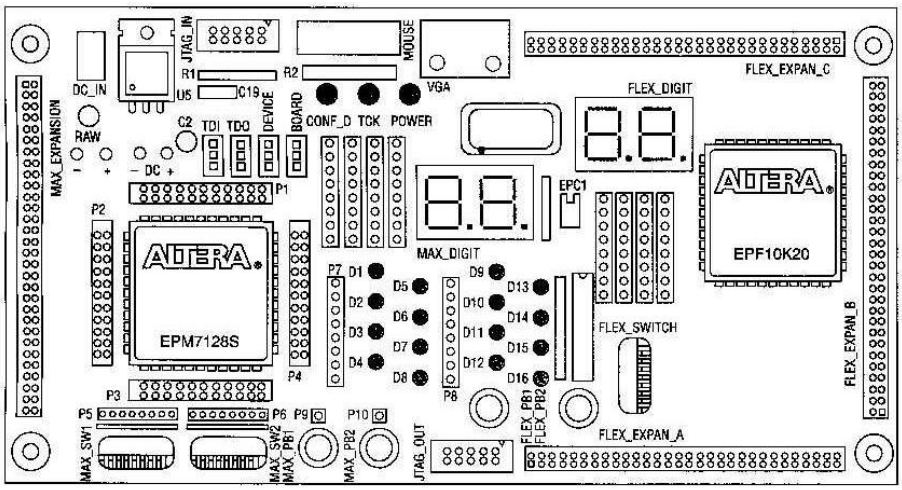


Figura 2 - UP1 Education Board

2.2 Motor de Passo

2.2.1 Introdução

Motores de passo podem ser vistos como motores elétricos sem comutadores. Tipicamente, todos os bobinados no motor são parte do estator, e o rotor geralmente é um imã permanente ou de indutância variável com um bloco dentado de algum material magnético suave.

Toda a comutação deve ser dirigida externamente pelo controlador do motor, e tipicamente, os motores e controladores são projetados de forma que o motor possa ser mantido em qualquer posição fixa como também ser movido com meios passo ou passos completos para ambos os lados.

Para algumas aplicações, deve ser feita uma escolha entre usar servos motores e motores de passo. Ambos os motores oferecem oportunidades semelhantes para posicionamento preciso, porém diferem em vários modos.

Servo motores requerem sistemas de controle de realimentação analógicas de algum tipo. Tipicamente, isto envolve um sensor de posições para prover a realimentação sobre a posição do rotor, e alguma mistura de circuitos para controlar uma corrente pelo motor inversamente proporcional à diferença entre a posição desejada e a posição atual.

Para se fazer uma escolha entre motores de passo e servomotores, devem ser considerados vários aspectos, os quais dependem da aplicação.

Por exemplo, a precisão de posicionamento feita com um motor de passo depende da geometria do rotor, enquanto a precisão de posicionamento feito com um servo motor depende da estabilidade e qualidade do circuito de realimentação.

Motores de passo podem ser usados em simples sistemas de controle sem realimentação, onde estes são geralmente adequados para sistemas que operam com baixas acelerações e com cargas estáticas, mas a realimentação de controle pode ser essencial para acelerações altas, particularmente se elas envolvem cargas variáveis. Se um motor de passo é sobrecarregado em um sistema de controle sem realimentação, todos os dados de posição do rotor são perdidos e o sistema deve ser reinicializado, enquanto servo motores não estão sujeitos a este problema.

Motores de passo entram em duas variedades, imã permanente e indutância variável (também há motores híbridos que são indistinguíveis de motores de imã permanentes do ponto de vista do controlador). Faltando uma especificação no motor, pode-se geralmente pode separar os dois quando nenhuma alimentação é aplicada.

Motores de passo existem em uma larga escala de resolução angular, onde os motores mais grosseiros tipicamente giram 90 graus por passo, enquanto motores de resoluções maiores com imãs permanentes podem comumente girar 1.8 ou até mesmo 0.72 graus por passo.

Com um controlador apropriado, motores com imã permanente e híbridos podem ser ligados em meio passo, e alguns controladores podem rodar passo fracionários menores ou micropasso.

Motores de passo Unipolares com imã Permanente ou híbridos com 5 ou 6 fios são normalmente montados com um dreno no centro de cada dois bobinados. Na prática, o centro dos bobinados é ligado tipicamente à alimentação positiva, e as duas terminações de cada bobinado é aterrada alternadamente para inverter a direção do campo provida pelo bobinado.

2.2.2 O Motor utilizado

Motor de Passo KP4M4-001 tem as seguintes características**:** alimentação+12V dc, Quatro fases, Unipolar, Imã Permanente, 3,6º por passo.

O eixo com um imã permanente acoplado é chamado rotor. O suporte estacionário contendo os pólos da bobina é chamado de estator. Com um motor unipolar, a corrente corre somente em uma direção nas espirais das bobinas, isto é, os pólos do estator podem somente ser polarizados de uma maneira.

O rotor é encaixado em duas engrenagens, que contém 25 dentes cada uma., conforme a figura 3.

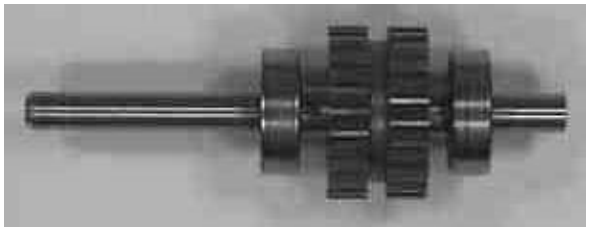


Figura 3 – Rotor do motor de passo

Com 25 dentes na borda do rotor e 4 bobinas excitadas individualmente por vez, o motor KP4M4-001 leva 100 passos para completar uma revolução. O espaço entre os dentes é dado pela equação: 360º / 25 = 14,4 º. Quando os dentes

estão alinhados com os dentes no polo do estator, eles estão desalinhados por um quarto do ângulo dos dentes do próximo polo do estator. Então quando a bobina é energizada , o rotor é puxado em um quarto de 14,4º, produzindo um passo de 3,6º, conforme a figura 4.

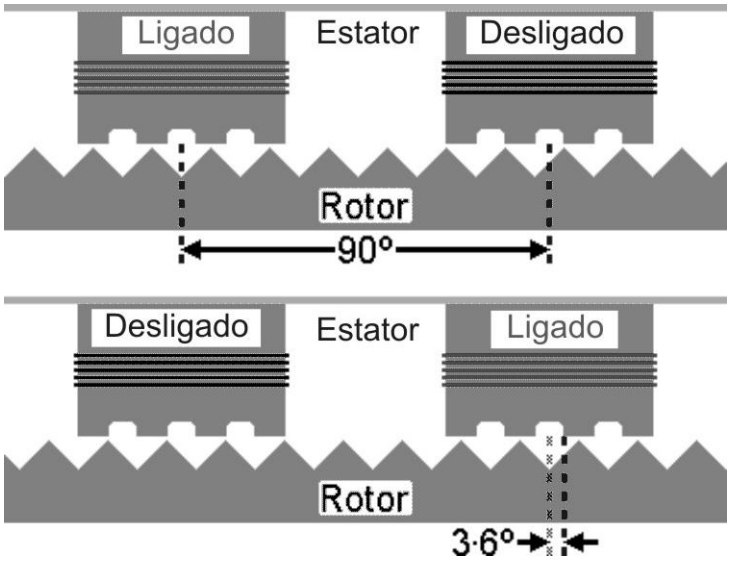


Figura 4 – Mudança de fase no motor de passo

A figura 5 ilustra o motor utilizado neste projeto e a figura 6 ilustra o conector PCB e o fios de conexão do motor.

Figura 5 –Motor de Passo (KP4M4-001) utilizado no Projeto

Figura 6 – Conector do KP4M4-001

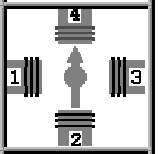
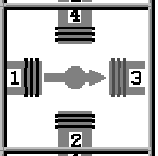
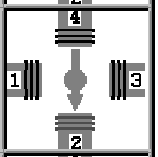
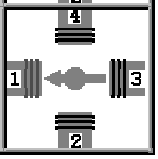
2.2.3 Seqüência de Passos para um Motor de Passo Unipolar de Imã Permanente de quatro fases

Os motores de passo de quatro fases possuem basicamente 2 modos de operação: (a) Single Coil Excitation (energização de uma bobina), (b) Two Coil Excitation (energização de duas bobinas).

Uma outra seqüência pode ser obtida intercalando-se as duas seqüências produzindo a seqüência (c) Half-Step.

a) Single-Coil Excitation – Uma bobina é energizada por vez.

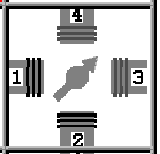
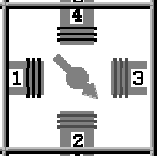
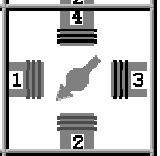
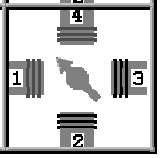
Tabela - 1 Fases do Single-Coil Excitation

Passo	Bobina 4	Bobina 3	Bobina 2	Bobina 1	
a.1	**Ligado**	Desligado	Desligado	Desligado	
a.2	Desligado	**Ligado**	Desligado	Desligado	
a.3	Desligado	Desligado	**Ligado**	Desligado	
a.4	Desligado	Desligado	Desligado	**Ligado**	

Esta seqüência produz o movimento mais suave e consome menor quantidade de energia.

b) Two-Coil Excitation – Duas bobinas sucessivas são energizadas por vez.

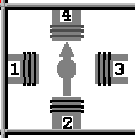
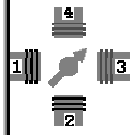
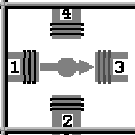
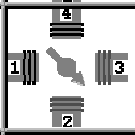
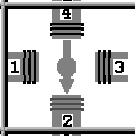
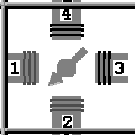
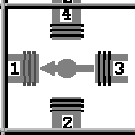
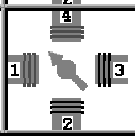
Tabela 2 - Fases do Two-Coil Excitation

Passo	**Bobina 4**	**Bobina 3**	**Bobina 2**	**Bobina 1**	
b.1	**Ligado**	**Ligado**	Desligado	Desligado	
b.2	Desligado	**Ligado**	**Ligado**	Desligado	
b.3	Desligado	Desligado	**Ligado**	**Ligado**	
b.4	**Ligado**	Desligado	Desligado	**Ligado**	

Esta seqüência não é tão suave como o Single Coil Excitation e consome mais energia, porém produz um torque maior.

c) Half Step – Intercalamento do Two-Coil Excitation com o Single Coil Excitation

Tabela 3 – Fases do Half Step

Passo	Bobina 4	Bobina 3	Bobina 2	Bobina 1	
a.1	**Ligado**	Desligado	Desligado	Desligado	
b.1	**Ligado**	**Ligado**	Desligado	Desligado	
a.2	Desligado	**Ligado**	Desligado	Desligado	
b.2	Desligado	**Ligado**	**Ligado**	Desligado	
a.3	Desligado	Desligado	**Ligado**	Desligado	
b.3	Desligado	Desligado	**Ligado**	**Ligado**	
a.4	Desligado	Desligado	Desligado	**Ligado**	
b.4	**Ligado**	Desligado	Desligado	**Ligado**	

Este seqüência produz mais posições estacionárias entre os passos. No caso do motor KP4M4-001, são requeridos 200 passos por revolução e 1.8º por passo.

2.3 Circuito de Potência

Este circuito tem a função de amplificar a corrente gerada pelo Circuito Comando, onde foi implementada a seqüência de passos para energizar as bobinas do motor. O esquema deste circuito é mostrado na figura 7.

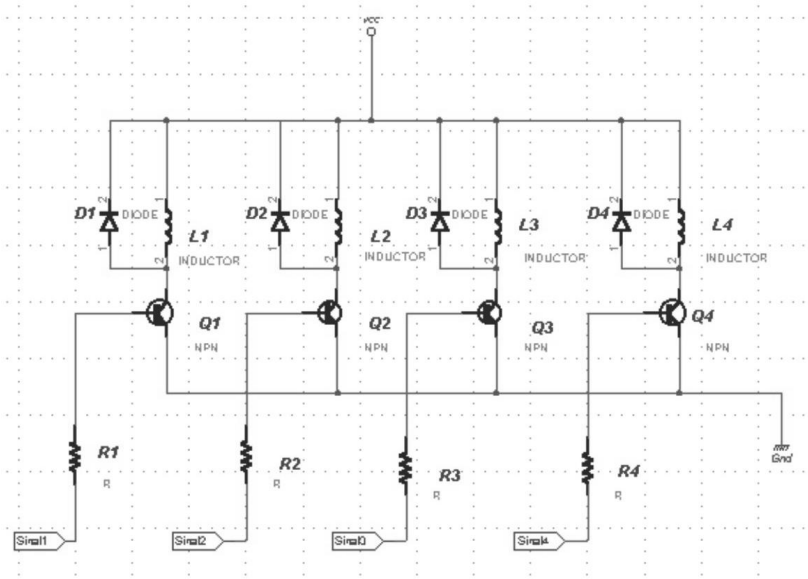


Figura 7 – Esquema do amplificador de Corrente

O sinal gerado pelo Circuito de comando para energizar uma bobina n de um motor entra na base do transistor. Se este sinal estiver em nível alto (1), e no caso aproximadamente 3.5 V, ele vai saturar o transistor e fazer com que ele funcione como uma chave fechada entre a bobina e o ponto de referência (Gnd).

O diodo foi utilizado para proteger o transistor da alta tensão que é gerada pela bobina após ela ser desernegizada.

2.4 Comando

O comando é composto pelos módulos Circuito de Decisão, Circuito Posição, e Gerador de Sinais, como descrito na figura 8.

Ele foi implementado com circuito digital, através da configuração do Chip Flex 10K do Kit Altera®

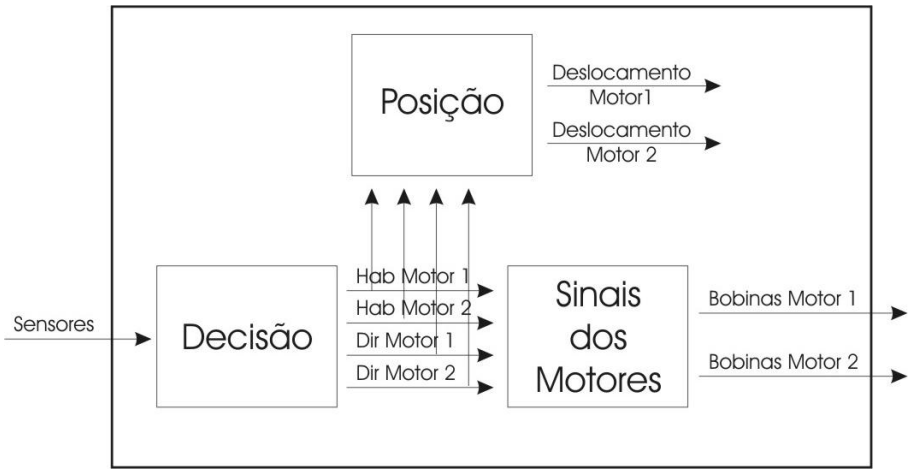


Figura 8 – Diagrama em Blocos do Circuito Comando

A função de cada bloco será descrita nas próximas sessões.

2.4.1 O gerador de sinais dos motores

A seqüência de sinais utilizada para energizar as bobinas do motor é a Two Coil Excitation. Devido ao fato de ter sido implementada em hardware, há a necessidade da construção de um máquina de estados, conforme a figura 9:

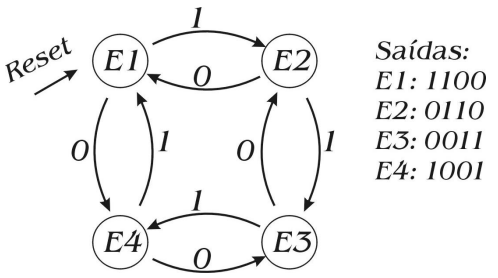


Figura 9 – Máquina de estados do Two Coil Excitation

A máquina quando estiver no estado E1 gera as saídas 1100 que irão energizar as bobinas 4 e 3 do motor. Na próxima transição do clock, se a entrada estiver alta (1) o próximo estado será o estado E2, cujas saídas são 0110, e caso a entrada no estado E1 for baixa (0), o próximo estado será o estado E4, cujas saídas são 1001. O mesmo processo se repete para os estados 2, 3 e 4.

Se energizadas na seqüência correta (E1 -> E2 -> E3 -> E4, ou E4 -> E3 -> E2 -> E1), o motor fará com que o Robô mova para frente ou para trás.

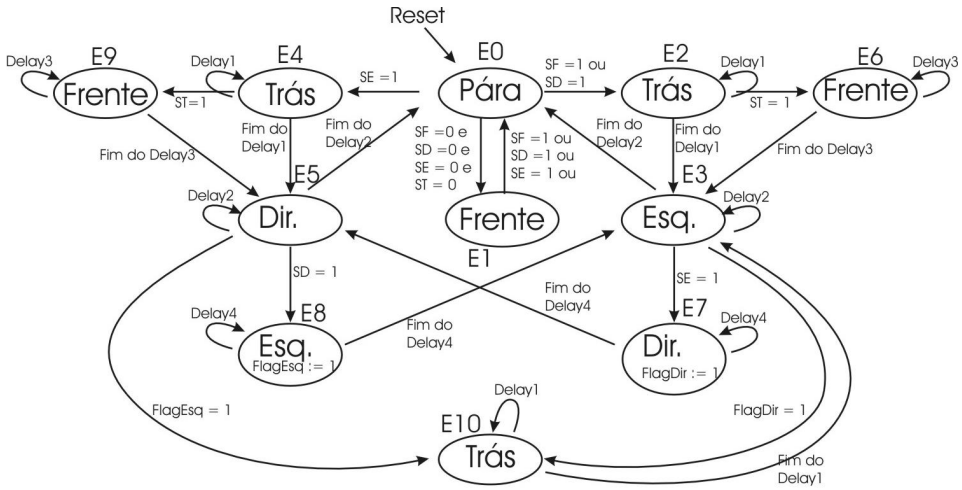
A implementação desta máquina de Estados foi realizada com Flip-Flop's e portas lógicas, conforme Anexo A.

2.4.2 O Circuito de Decisão

Este circuito tem a função de receber as entradas dos sensores e decidir qual será a ação que deverá ser feita. Para a construção do circuito de decisão, utilizou-se uma máquina de estados conforme a figura 10 que foi implementada em linguagem de descrição de sistemas digitais (VHDL), [9] [10]. Para detalhes da implementação, consultar Anexo B.

A idéia do funcionamento deste circuito é que se o robô encontre um obstáculo na sua frente ou na sua direita ele ande para trás um tempo pré determinado e depois tente girar 90º para a esquerda. Caso ele não consiga girar para a esquerda ele tentará girar para a direita. Caso ele não consiga girar para a direita ele irá mover-se para trás um tempo pré determinado e tentará girar para a esquerda, e caso não consiga girar para a esquerda. Este processo se repete até

que ele consiga girar para a direita ou para a esquerda. O mesmo processo de decisão para o sensor da Frente ou o da Direita é feito também para o sensor da Esquerda.



Legenda:
Os rótulos dos estados (Frente, trás, Esq = Esquerda, Dir. = Direita) indicam que as saídas para alimentação dos motores estarão configuradas para fazer o movimento do rótulo.
Delay 1 = 200 passos
Delay 2 = 100 passos
Delay 3 = 50 passos
Delay 4 = Delay 2 atual (ou seja, o tanto que ele desalinhou quando tentou girar)

Figura 10 – Máquina de estados do circuito de Decisão do robô

Agora descrevemos detalhadamente o funcionamento da máquina de estados da figura 10.

Ao iniciar o sistema com um Reset, serão gerados sinais que farão com que o motor pare (estado E0), na próxima transição do Clock, caso o Botão reset não esteja pressionado, as entradas do sistema (sensores) serão analisados. Caso nenhum sensor esteja ativado as saídas estarão configuradas para fazerem o robô andar para frente (estado E1).

Se ao andar para frente, o robô encontre algum obstáculo, as saídas serão configuradas para fazer o robô parar (estado E0). Agora será verificado qual

caminho na máquina de estado a ser seguido, a partir do sensor que foi ativado. No caso temos duas possibilidades: caminho em que o Sensor da Frente (SF) ou o da Direita (SD) foi ativado e o outro caminho (estado E2), e o caminho em que o sensor da Esquerda (SE) foi ativado (estado E4). No caso desta máquina, a ativação ocorre em nível alto (1).

Caso SF seja ativado ou SE for ativado (SF = 1 ou SE = 1), as saídas farão com que o robô ande para trás (estado E2) num tempo Delay 1 e depois gire para a esquerda (estado E3) num tempo Delay 2. Mas se durante o estado E2 deste caminho, estado em que o robô movimenta para Trás, se o sensor de trás for ativado (ST = 1), o robô irá andar para frente (estado E6) Delay 3 e depois irá girar para esquerda (estado E2) Delay 2. Se no estado em que o robô tentar girar para esquerda (estado E3), o sensor da esquerda for ativado, ele voltará para a Direita (estado E7) o tempo Delay 2 corrente (Delay 4) para que ele volte para a direita o tanto que ele tentou girar para a esquerda e depois disso ele irá tentar desviar para a direita (estado E5). Mas caso o robô encontre um obstáculo ao tentar girar para direita num tempo Delay2, ele voltará para esquerda (estado E8) o tempo Delay2 corrente. No grafo, a próxima ação do robô seria tentar girar para esquerda num tempo Delay2 novamente, mas como o robô já tentou fazer este movimento anteriormente, uma variável de controle (maiores detalhes consultar ANEXO B) foi implementada para que a máquina de estados vá para o estado E10, cujas saídas estão configuradas para o robô andar de ré num tempo Delay 1. Depois o robô vai tentar novamente virar para a esquerda (estado E3) e a partir daí o processo se repete até que ele consiga desviar.

O mesmo processo será repetido caso o Sensor da Esquerda seja ativado.

2.4.3 O Circuito Posição

Este circuito recebe como entrada, o sinal de habilitação e a direção de motor. Com esses dados ele é capaz de calcular o deslocamento de cada roda. O funcionamento deste circuito é descrito a seguir, implementado em linguagem VHDL:

Para cada motor tem-se uma variável que armazena (maiores detalhes ver ANEXO C) a quantidade de passos que o motor andou para frente e outra que armazena a quantidade de passos que o motor andou para trás. A cada cinqüenta clocks o valor dessas variáveis são comparados, se o valor delas forem iguais indica que o robô não se locomoveu, se a variável que guarda a quantidade de passos que o robô se locomoveu para frente for maior que a variável que conta a quantidade de passos que o robô se locomoveu para trás, então a diferença entre elas é positiva, caso contrário, negativa. Este valor colocado na saída do circuito.

3. Conclusões

Após montado a plataforma juntamente com os motores, a bateria e o kit Altera, o protótipo não realizou curvas, devido ao baixo torque dos motores de passo. O robô é capaz de realizar conversões somente se a bateria não estiver na plataforma.

A bateria utilizada não permitiu uma autonomia suficiente de funcionamento do robô, porque o consumo de corrente dos motores é muito alto, além da alimentação do circuito de comando.

Apesar de ser extremamente simples, as dificuldades mecânicas encontradas impediram o sucesso do projeto.

Nestas condições, o robô move para frente e quando encontrado obstáculo, ele move para trás, mas ao tentar realizar uma curva para desviar do obstáculo, ele não consegue devido ao peso da bateria.

4. Propostas Futuras

- Como o motor de passo KP4M4-001 não forneceu torque suficiente para o funcionamento, este deverá ser substituído visando o melhoramento do sistema de tração do robô.
- Construção de uma estação remota, que poderia ser um microcomputador, para receber os dados de deslocamento do robô, através de ondas de rádio, e em seguida mapeamento da trajetória realizada pelo robô.
- Melhoramento do circuito de comando de modo a resolver situações de decisão ainda não previstas.

5. Referências Bibliográficas

[1] VERLAB. Laboratório de Visão Computacional e Robótica do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais. http://www.verlab.dcc.ufmg.br/ .2001

[2] LAI - Laboratório de Automação Inteligente da Universidade Federal do Espírito Santo. http://www.ele.ufes.br/pesq/robotica.htm . 2001

[3] USATEGUI, J. M. Â.; LEON, J. N. S. de. **Manual Prático de Robótica**. Tradução de José Alberto Fazano et al. São Paulo: Hemus Editora Ltda. 1988

[4] HARRIES, Ian. **Ian Harries home page.** http://www.doc.ic.ac.uk/~ih/. 2001

[5] NASA. **Mars Path Finder Mission**. **Home page** http://www.mpf.jpl.nasa. gov/MPF/index0.html. 1997

[6] WALNUM, Clayton. **Aventuras em realidade virtual**. Rio de Janeiro: Berkeley, 1993.

[7] ALTERA - **Manual:** Data Book, 1998. San Jose.

[8] RHEDA TECHNOLOGIE. **Projetos Eletrônicos Personalizados: Tipos de motores de passo.** http://www.rheda.com.br. 2001

[9] ALTERA. **Manual:** MAX+PLUS® II VHDL, 1998. San Jose

[10] PERRY, Douglas L.. **VHDL (Computer hardware descripton language)**. 3. ed. New York: McGrawHill, 1998.

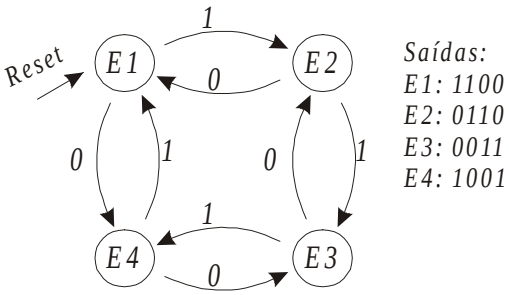
[11] ALTERA. **Manual:** University Program Design Laboratory Package, 1999, San Jose

[12] BIGNELL, James W. & DONOVAN, Robert L. Eletrônica Digital. São Paulo: Editora Makron Books, vol. 1 e 2, 1995.

[13] MORAES, Fernando. **VHDL - Linguagem de Descrição de Hardware.** 1998. Apostila

ANEXOS

ANEXO A – Gerador de Sinais



1) Número de Flip Flop's necessários para implementar a máquina é dado pela fórmula:

Número de Flip Flop's >= Log_2 Número de Estados

Número de Flip Flop's > = Log_2 4

Número de Flip Flop's >= 2

2) Tabela Verdade

Rótulos dos estados associados aos valores das saídas dos Flip Flop's

E1 = 00; E2 = 01; E3 = 10; E4 = 11

Estado Atual		**Entradas**	**Estado Futuro**	
Q1	**Q0**	**Direção**	**Q1**	**Q0**
0	0	0	1	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

3) Mapa de Karnaugh [12] para obter as equações lógicas das entradas dos FF's

D1

		Entrada	
Q1	**Q0**	**0**	**1**
0	0	1	0
0	1	0	1
1	0	1	0
1	1	0	1

D1 = (/Q1 /Q0 /E) + (/Q1 Q0 E) + (Q1 Q0 /E) + (Q1 /Q0 E)

D0

		Entrada	
Q1	**Q0**	**0**	**1**
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

D0 = /Q0

4) Configuração das saídas (S4, S3, S2, S1) para cada estado

	S4	**S3**	**S2**	**S1**
E1	1	1	0	0
E2	0	1	1	0
E3	0	0	1	1
E4	1	0	0	1

5) Tabela verdade para as saídas

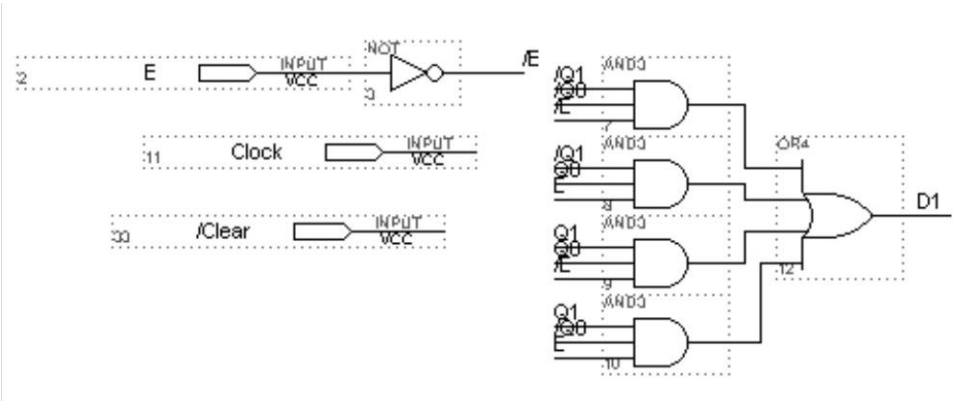
Estado		Saídas			
Q1	**Q0**	**S4**	**S3**	**S2**	**S1**
0	0	1	1	0	0
0	1	0	1	1	0
1	0	0	0	1	1
1	1	1	0	0	1

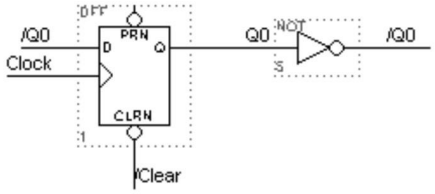
S4 = (/Q1 /Q0) + (Q1 Q0)

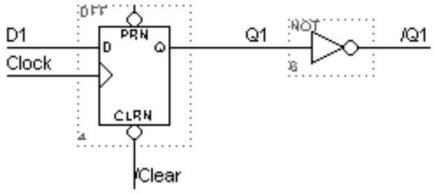
S3 = (/Q1 /Q0) + (/Q1 Q0)

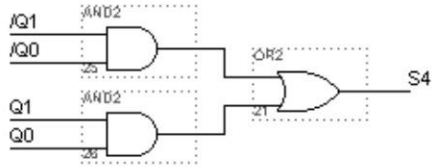
S2 = (/Q1 Q0) + (Q1 /Q0)

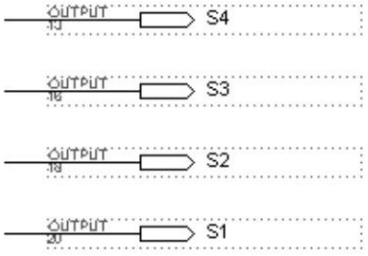
S1 = (Q1 /Q0) + (Q1 Q0)

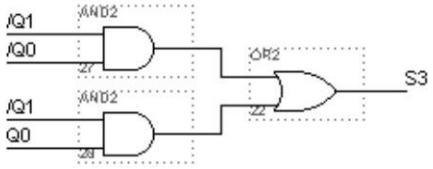


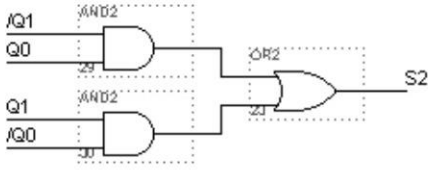

/Q0
Clock
D
PRN
Q
CLRN
/Clear
Q0
NOT
/Q0

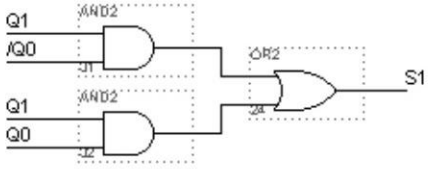

D1
Clock
D
PRN
Q
CLRN
/Clear
Q1
NOT
/Q1


/Q1
/Q0
Q1
Q0
AND2
OR2
S4


OUTPUT
S4
OUTPUT
S3
OUTPUT
S2
OUTPUT
S1


/Q1
/Q0
Q1
Q0
AND2
OR2
S3


/Q1
Q0
Q1
/Q0
AND2
OR2
S2


Q1
/Q0
Q1
Q0
AND2
OR2
S1

ANEXO B – Circuito de Decisão

Arquivo Decisao.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity Decisao is
port(
   SensorF     : in bit;
   SensorE     : in bit;
   SensorD     : in bit;
   SensorT     : in bit;
   Clock       : in bit;
   Reset       : in bit;
   SinalDelay1 : in bit;
   SinalDelay2 : in bit;
   SinalDelay3 : in bit;
   SinalDelay4 : in bit;
   HabDelay1   : out bit;
   HabDelay2   : out bit;
   HabDelay3   : out bit;
   HabDelay4   : out bit;
   HabM1       : out bit;
   HabM2       : out bit;
   DirM1       : out bit;
   DirM2       : out bit
);
end;
```

```
architecture a of Decisao is
   type STATES is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10);
   signal EstAtual, ProxEst : STATES;
   signal FlagDir,FlagEsq  : bit;

begin

   -- processo que diz quem é o estado atual
   controle: process
   begin
      wait until Clock'  event and Clock = '  0'  ;
      EstAtual <= ProxEst;
      if Reset = '  0then
         EstAtual <= S0;
      end if;
      if (EstAtual = S8) then
         FlagEsq <= '  1'  ;
         elsif (EstAtual = S7) then
            FlagDir <= '  1'  ;
         elsif (EstAtual = S10) then
            FlagEsq <= '  0'  ;
            FlagDir <= '  0'  ;
      end if;
   end process;

   -- processo verifica o estado atual e atribui o próximo estado
```

```
combinacional: process (EstAtual, Reset)
begin
   case EstAtual is
      when S0 =>
         HabM1 <= '0';
         HabM2 <= '0';
         DirM1 <= '0';
         DirM2 <= '0';
         HabDelay1 <= '0';
         HabDelay2 <= '0';
         HabDelay3 <= '0';
         HabDelay4 <= '0';

         if (SensorF = '0' and SensorD = '0' and SensorE = '0' and SensorT =
'0') or (SensorT = '1') then
            ProxEst <= S1;
            elsif (SensorF = '1' or SensorD = '1') then
               ProxEst <= S2;
               HabDelay1 <= '1';
            elsif (SensorE = '1') then
               ProxEst <= S4;
               HabDelay1 <= '1';
         end if;
      when S1 =>
         HabM1 <= '1';
         HabM2 <= '1';
         DirM1 <= '1';
         DirM2 <= '1';
```

```
            if (SensorF = '0' and SensorD = '0' and SensorE = '0' and SensorT =
'0') then
               ProxEst <= S1;
               else
                  ProxEst <= S0;
               end if;
            when S2 =>
               HabM1 <= '1';
               HabM2 <= '1';
               DirM1 <= '0';
               DirM2 <= '0';
               if (SinalDelay1 = '0' and SensorT = '1') then
                  ProxEst <= S6;
                  HabDelay1 <= '0';
                  HabDelay3 <= '1';
                  elsif (SinalDelay1 = '0') then
                     ProxEst <= S2;
                     HabDelay1 <= '1';
                  elsif (SinalDelay1 = '1') then
                     ProxEst <= S3;
                     HabDelay1 <= '0';
                     HabDelay2 <= '1';
               end if;
            when S3 =>
               HabM1 <= '1';
               HabM2 <= '1';
               DirM1 <= '1';
               DirM2 <= '0';
```

```
    if (FlagDir = '1') then
      ProxEst <= S10;
      elsif (SinalDelay2 = '0' and SensorE = '1') then
        ProxEst <= S7;
        HabDelay2 <= '0';
        HabDelay4 <= '1';
      elsif (SinalDelay2 = '0') then
        ProxEst <= S3;
      elsif (SinalDelay2 = '1') then
        ProxEst <= S0;
        HabDelay2 <= '0';
    end if;
  when S4 =>
    HabM1 <= '1';
    HabM2 <= '1';
    DirM1 <= '0';
    DirM2 <= '0';
    if (SinalDelay1 = '0' and SensorT = '1') then
      ProxEst <= S9;
      HabDelay1 <= '0';
      HabDelay3 <= '1';
    elsif (SinalDelay1 = '0') then
      ProxEst <= S4;
    elsif (SinalDelay1 = '1') then
        ProxEst <= S5;
        HabDelay1 <= '0';
        HabDelay2 <= '1';
    end if;
```

```
when S5 =>
   HabM1 <= '1';
   HabM2 <= '1';
   DirM1 <= '0';
   DirM2 <= '1';
   if (FlagEsq = '1') then
      ProxEst <= S10;
      elsif (SinalDelay2 = '0' and SensorD = '1') then
         ProxEst <= S8;
         HabDelay2 <= '0';
         HabDelay4 <= '1';
      elsif (SinalDelay2 = '0') then
         ProxEst <= S5;
      elsif(SinalDelay2 = '1') then
         ProxEst <= S0;
         HabDelay2 <= '0';
   end if;
when S6 =>
   HabM1 <= '1';
   HabM2 <= '1';
   DirM1 <= '1';
   DirM2 <= '1';
   if (SinalDelay3 = '0') then
      ProxEst <= S6;
      else
         ProxEst <= S3;
         HabDelay2 <= '1';
         HabDelay3 <= '0';
```

```
    end if;
when S7 =>
    HabM1 <= '  1'  ;
    HabM2 <= '  1'  ;
    DirM1 <= '  0'  ;
    DirM2 <= '  1'  ;
    if (SinalDelay4 = '0  'then
        ProxEst <= S7;
        HabDelay4 <= '  1'  ;
        else
            ProxEst <= S5;
            HabDelay2 <= '  1'  ;
            HabDelay4 <= '  0'  ;
    end if;
when S8 =>
    HabM1 <= '  1'  ;
    HabM2 <= '  1'  ;
    DirM1 <= '  1'  ;
    DirM2 <= '  0'  ;
    if (SinalDelay4 = '0  'then
        ProxEst <= S8;
        else
            ProxEst <= S3;
            HabDelay2 <= '  1'  ;
            HabDelay4 <= '  0'  ;
    end if;
when S9 =>
    HabM1 <= '  1'  ;
```

```
            HabM2 <= '1';
            DirM1 <= '1';
            DirM2 <= '1';
            if (SinalDelay3 = '0') then
               ProxEst <= S9;
               else
                  ProxEst <= S5;
                  HabDelay2 <= '1';
                  HabDelay3 <= '0';
            end if;
         when S10 =>
            HabM1 <= '1';
            HabM2 <= '1';
            DirM1 <= '0';
            DirM2 <= '0';
            if (SinalDelay1 = '0') then
               ProxEst <= S10;
               else
                  ProxEst <= S3;
            end if;
      end case;
   end process;
end a;
```

Nome do Arquivo: Delay.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity Delay is
   port(
      Clock       : in bit;
      HabDelay1   : in bit;
      HabDelay2   : in bit;
      HabDelay3   : in bit;
      HabDelay4   : in bit;
      SinalDelay1 : out bit;
      SinalDelay2 : out bit;
      SinalDelay3 : out bit;
      SinalDelay4 : out bit
   );
end;

architecture a of Delay is
signal Contador1,Contador2,Contador3,Contador4,ContadorAux : integer range
0 to 200;

begin

   delay: process (Clock)
   begin
      if (Clock'  event and Clock = '  1' ) then
         if (HabDelay1 = '1  ' ) then
            if Contador1 = 200 then
               Contador1 <= 0;
               SinalDelay1 <= '  1'  ;
               else
```

```
        Contador1 <= Contador1 + 1;
        SinalDelay1 <= '0';
    end if;
elsif (HabDelay2 = '1') then
    ContadorAux <= Contador2 + 1;
    if Contador2 = 100 then
        Contador2 <= 0;
        SinalDelay2 <= '1';
        else
            Contador2 <= Contador2 + 1;
            SinalDelay2 <= '0';
    end if;
elsif (HabDelay3 = '1') then
    if Contador3 = 30 then
        Contador3 <= 0;
        SinalDelay3 <= '1';
        else
            Contador3 <= Contador3 + 1;
            SinalDelay3 <= '0';
    end if;
elsif (HabDelay4 = '1') then
    Contador2 <= 0;
    if Contador4 = ContadorAux then
        Contador4 <= 0;
        ContadorAux <= 0;
        SinalDelay4 <= '1';
        else
            Contador4 <= Contador4 + 1;
```

```
                    SinalDelay4 <= '  0'  ;
                end if;
            end if;
        end if;
    end process;
end a;
```

ANEXO C – Circuito Posição

Arquivo Posicao.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;


entity Posicao is
   port(
      Clock       : in bit;
      Reset       : in bit;
      DirM1       : in bit;
      DirM2       : in bit;
      HabM1       : in bit;
      HabM2       : in bit;
      PassoM1     : out integer;
      PassoM2     : out integer

   );
end Posicao ;

architecture a of Posicao is
signal ContaClock : integer range 0 to 50;
signal PassoFrenteM1,PassoFrenteM2,PassoTrasM1,PassoTrasM2 : integer;
begin
   process (Clock)
```

```
begin
   if Clock'event and Clock = '1' then
      ContaClock <= ContaClock + 1;
      if Reset = '0' then
         ContaClock <= 0;
         PassoFrenteM1 <= 0;
         PassoFrenteM2 <= 0;
         PassoTrasM1 <= 0;
         PassoTrasM2 <= 0;
      end if;
      if HabM1 = '1' then
         if DirM1 = '1' then
            PassoFrenteM1 <= PassoFrenteM1 + 1;
            else PassoTrasM1 <= PassoTrasM1 - 1;
         end if;
      end if;
      if HabM2 = '1' then
         if DirM2 = '1' then
            PassoFrenteM2 <= PassoFrenteM2 + 1;
            else PassoTrasM2 <= PassoTrasM2 - 1;
         end if;
      end if;
      if ContaClock = 50 then
         PassoFrenteM1 <= 0;
         PassoFrenteM2 <= 0;
         PassoTrasM1 <= 0;
         PassoTrasM2 <= 0;
         ContaClock <= 0;
```

```
        end if;
    end if;
end process;

process
begin
    if ContaClock = 50 then
        if (PassoFrenteM1 > (-PassoTrasM1)) then
            PassoM1 <= PassoFrenteM1 - (-PassoTrasM1);
            else PassoM1 <= PassoTrasM1 + PassoFrenteM1;
        end if;
        if (PassoFrenteM1 = (-PassoTrasM1)) then
            PassoM1 <= 0;
        end if;
        if PassoFrenteM2 >= (-PassoTrasM2) then
            PassoM2 <= PassoFrenteM2 - (-PassoTrasM2);
            else PassoM2 <= PassoTrasM2 + PassoFrenteM2;
        end if;
        if PassoFrenteM2 = (-PassoTrasM2) then
            PassoM2 <= 0;
        end if;
        else -- else do ContaClock = 50
            PassoM1 <= 0;
            PassoM2 <= 0;
    end if;
end process;

end a;
```