TITO PAGOTO SALLES

PROJETO DO CONTROLE DE VARREDURA AUTOMÁTICA DO TANQUE ACÚSTICO

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em Sistemas de Energia e Automação

ORIENTADOR: Prof. Dr. Carlos Dias Maciel

São Carlos

2009

AGRADECIMENTOS

Agradeço ao professor e amigo Dr. Carlos Dias Maciel, por sua orientação, apoio, ajuda, confiança e todo esforço colocado nesse trabalho de conclusão de curso.

Aos meus pais e irmã, que sempre apostaram em mim, pelo suporte, pelo apoio, pelas orações e principalmente pelo amor, que faz a vida valer à pena e possuir um enorme significado.

Aos meus Avôs, os quais sempre me incentivaram e com todas suas experiências, serviram de modelos de ética, bom senso e responsabilidade.

Aos professores da Universidade de São Paulo, que me prepararam e deram o conhecimento necessário para concluir mais uma etapa importante de minha vida.

Aos eternos e verdadeiros amigos que ganhei no decorrer deste período com os quais compartilhei vários momentos de alegria e tristeza.

ÍNDICE

LISTA DE FIGURAS ..... **I**

LISTA DE TABELAS ..... **III**

RESUMO ..... **IV**

ABSTRACT ..... **V**

1. INTRODUÇÃO **1**

2. REVISÃO DA LITERATURA **2**

2.1. MICROCONTROLADORES 2
2.1.1. Diferença entre microcontrolador e microprocessador 3
2.1.2. Mercado de microcontroladores 4
2.1.3. Escolha de microcontroladores 5
2.1.4. Recursos comuns a todos os microcontroladores 6
2.1.4.1. Unidade central de processamento (CPU) 6
2.1.4.2. Memória 7
2.1.4.3. Circuito de relógio (Clock) 9
2.1.4.4. Temporizadores (Timers) 9
2.1.4.5. Watchdog 9
2.1.4.6. Proteção contra falha de alimentação (Brownout) 10
2.1.4.7. Modo de espera e/ou baixo consumo 10
2.1.4.8. Conversor A/D e Conversor D/A 10
2.1.4.9. Comparador analógico e Modulador de largura de pulsos (PWM) 10
2.1.4.10. Portas de E/S digitais (I/O) e Portas de comunicação 11
2.2. PIC 16F877A 11
2.2.1. Estrutura Interna 12
2.2.2. Arquitetura e Conjunto de Instruções 15
2.2.3. Módulos CCP 16
2.2.4. Temporizadores 16
2.2.4.1. Timer0 16
2.2.4.2. Timer1 17
2.2.4.3. Timer2 17
2.2.5. Conversor A/D 18
2.2.6. Linguagem de Programação 18
2.3. AMBIENTE DE DESENVOLVIMENTO MIKROC PARA PIC 19
2.4. MOTOR DE PASSO 20
2.4.1. Tipos de Motores de Passo 21
2.4.2. Funcionamento do motor de passo 22
2.4.3. Vantagens e desvantagens 24

2.5. DRIVER DE POTÊNCIA 24
3. METODOLOGIA 26
3.1. MATERIAIS E MÉTODOS 26
3.2. FLUXOGRAMA DE CONTROLE 28
3.2.1. Modo manual 29
3.2.2. Modo automático 30
3.3. RESULTADOS E DISCUSSÕES 35
4. CONCLUSÃO 37
REFERÊNCIAS BIBLIOGRÁFICAS 38
ANEXO A 39
A.1 CRIANDO UM PROJETO EM LINGUAGEM C 39
A.2 COMPILANDO UM PROJETO 41
A.3 SIMULANDO A EXECUÇÃO 43
ANEXO B 46
B.1 PROGRAMA EM C 46

LISTA DE FIGURAS

Figura 1 – Sistema baseado em um microprocessador. 4

Figura 2 – Modelo do microcontrolador. 4

Figura 3 – Terminais do PIC 16F877A. 12

Figura 4 – Estrutura interna do PIC 16F877A. 14

Figura 5 – Arquitetura *Harvard*. 15

Figura 6 – Componentes do motor de passo. 20

Figura 7 – Representação dos enrolamentos do motor de passo unipolar. 22

Figura 8 – Representação dos enrolamentos do motor de passo bipolar. 22

Figura 9 – Alinhamento entre as bobinas para motor bipolar com passo inteiro. 23

Figura 10 – Curva característica Torque vs. Velocidade do motor de passo. 24

Figura 11 – Diagrama para a ligação do motor. 25

Figura 12 – Tanque de varredura. 26

Figura 13 – Diagrama de blocos do sistema de movimentação. 27

Figura 14 – Fluxograma geral do programa. 28

Figura 15 – Curva de torque do motor utilizado. 29

Figura 16 – Fluxograma da rotina manual. 30

Figura 17 – Varredura do modo automático. 31

Figura 18 - Gráfico Freqüência vs. Passo 33

Figura 19 – Fluxograma da rotina automática. 34

Figura 20 – Medida do osciloscópio para passos no modo manual. 35

Figura 21 – Medida do osciloscópio para passos no modo automático. 36

Figura 22 – Tela inicial no MikroC. 39

Figura 23 – Ícone para cria novo projeto. 39
Figura 24 – Tela de configurações para novo projeto. 40
Figura 25 – Tela para a programação em C. 41
Figura 26 – Ícone para compilar o projeto. 41
Figura 27 – Tela MikroC após a compilação. 42
Figura 28 – Caixa de mensagens e alertas. 42
Figura 29 – Ícone para visualizar as estatísticas do programa. 43
Figura 30 – Tela para a visualização das estatísticas. 43
Figura 31 – Ícone para iniciar a simulação. 44
Figura 32 – Tela após entrar no modo de simulação. 44
Figura 33 – Barra de controle da simulação. 45

LISTA DE TABELAS

Tabela 1 – Principais microcontroladores.....2

Tabela 2 – Número de ciclos e freqüência no modo automático.....32

RESUMO

Este trabalho propôs o controle de um sistema de varredura automática mediante o uso do microcontrolador PIC 16F877A a fim de avaliar o campo acústico de transdutores de ultra-som. Criou-se uma rotina de controle automática, que faz a varredura em um tanque de 41 planos pré-definidos, espaçados de 1 mm, de área 41 mm x 41 mm.

Palavras-chave: microcontoladores; PIC 16F877A; Controle; Motor de passo; MikroC; programação em C.

ABSTRACT

This paper proposed the control of a system of automatic scanning by using the PIC 16F877A microcontroller in order to evaluate the acoustical field of transducers of ultrasound. Was created a routine of automatic control, which scans in a tank of 41 pre-defined plans, spaced 1 mm and area 41 mm x 41 mm.

Keywords: Microcontrollers; PIC 16F877A; control; step motor; MikroC; C programming.

1. Introdução

As técnicas derivadas do uso do ultra-som são utilizadas em diversas áreas, destacando-se as aplicações na área de saúde e ensaios não destrutivos. Os equipamentos que utilizam ultra-som como princípio físico de funcionamento devem ser avaliados por instrumentos que possam garantir a aferição de tais.

A calibração de transdutores de ultra-som tem importância fundamental no processo de medição para prover a confiabilidade necessária aos equipamentos que utilizam o ultra-som como principio de funcionamento. Em um processo de avaliação metrológica, a manutenção da rastreabilidade, conforme descrita no Vocabulário Internacional de Termos Fundamentais e Gerais de Metrologia (VIM) [1] é fator importante, pois informa quão próximo o resultado de uma medição está em concordância com valores estabelecidos por padrões.

A partir da calibração, é possível identificar a confiabilidade de um equipamento de medição, comparando os resultados da calibração com especificações de normas [10] e, então, expressar a incerteza associada ao resultado numérico declarado, a qual quantifica a qualidade final do resultado da medição. A calibração de transdutores ultra-sônicos é de grande importância, pois, a partir da mesma, é possível calcular a pressão gerada no campo ultra-sônico a partir do nível de tensão ou corrente gerada pelo transdutor.

Neste contexto, métodos para mapeamento do campo acústico confiáveis são necessários para garantir a segurança de funcionamento destes equipamentos. O controle destes métodos pode ser realizado digitalmente com o uso de microcontroladores.

O controle digital tem desempenhado papel fundamental no avanço da engenharia e da ciência. Além da importância em sistemas de veículos espaciais, sistemas de posicionamento, sistemas robóticos e similares, o controle digital tem se tornado de grande importância e parte integrante dos modernos processos industriais e de produção. Grande parte deste avanço deve-se aos microcontroladores, que, a cada dia, possuem maior capacidade de processamento em um tamanho menor. Sua grande versatilidade e relativa simplicidade de programação têm tornado mais fácil o controle de processos com eficiência e confiabilidade.

Neste trabalho, foi desenvolvida uma rotina de varredura automática microprocessada a fim de avaliar o campo acústico de transdutores de ultra-som, através do controle de motores de passo.

2. Revisão da literatura

2.1. Microcontroladores

Microcontroladores se tornaram essenciais nos dias de hoje e são responsáveis pelo poder de processamento para de carros, eletrodomésticos, eletrônicos, etc. Estes dispositivos têm ganhado destaque pela sua velocidade de processamento e miniaturização constante que permitiram produtos não apenas menores, mas mais eficientes e com diversos recursos.

Atualmente, existe uma grande diversidade de produtos para solucionar os mais diferentes problemas de controle. O primeiro microcontrolador foi lançado pela Intel em 1978 e recebeu a sigla 8048, que depois evoluiu dando origem à família 8051, em 1983. Dentre os mais populares, destacam-se os microcontroladores da Tabela 1 [2]:

Tabela 1 – Principais microcontroladores

8051 (ATMEL e outros)	É o microcontrolador mais popular. Possui programação simples e é de fácil aquisição em diversas configurações e encapsulamentos.
PsoC (*Cypress*)	A tecnologia do PsoC agrega em um único chip um microcontrolador embarcado de 8 bits, com um core muito parecido com 8051.
AVR (ATMEL)	O AVR é um microcontrolador de arquitetura Harvard. Tem 32 registros de 8 bits e é extremamente rápido entre os microcontroladores de 8 bits.
PIC (*Microchip*)	Família de microcontroladores de 8 e 16 bit. São facilmente encontrados no mercado e possuem farto material para consulta. Foram os primeiros microcontroladores RISC.
HT8 (*Holtek*)	São microcontroladores de alto desempenho da tecnologia RISC que têm um número reduzido de instruções.
ARM (Philips, ST e outros)	É uma arquitetura de processador de 32 bits e é usado principalmente em sistemas embarcados.
MSP430 (Texas)	Os MSP430 são microcontroladores RISC de 16 bits voltados para aplicações de baixo consumo de energia.

O microcontrolador é um circuito integrado de alta escala de integração que incorpora a maior parte dos elementos que configuram um controlador dispondo normalmente dos seguintes componentes:

- Unidade central de processamento (CPU);
- Memória de programa (ROM, RAM, FLASH);
- Diversos módulos para controle de periféricos (temporizadores, porta serial, conversores A/D e D/A, etc.);
- Circuito gerador de *clock*, para sincronizar o funcionamento do sistema.

Os produtos que incorporam um microcontrolador têm uma maior confiabilidade, pois, diminuindo o número de componentes, diminui o risco de falhas e de ajustes ao circuito. Outra vantagem é a redução do tamanho no produto, que é possível devido à integração do microcontrolador em uma placa [6].

2.1.1. Diferença entre microcontrolador e microprocessador

Um microprocessador é um circuito integrado, que contém uma unidade central de processamento (CPU), também chamado processador. A CPU é formada por uma unidade de controle que interpreta as instruções e as executa. Em um microprocessador temos os barramentos de dados, endereços e controle (figura 1), que tornam possível a conexão da CPU com a memória, com os módulos de entrada e saída, e periféricos em geral. Desta forma, diz-se então, que um microprocessador é um sistema aberto, porque sua configuração é variável de acordo com a aplicação.

O principal aspecto que difere os microcontroladores dos microprocessadores é a sua funcionalidade. Para que um microprocessador possa ser utilizado, outros componentes devem ser adicionados, tais como memória, *chipset* e componentes para enviar e receber dados. Por outro lado, o microcontrolador foi projetado para possuir todas estas funcionalidades em um único circuito integrado, representado pela figura 2 [2].

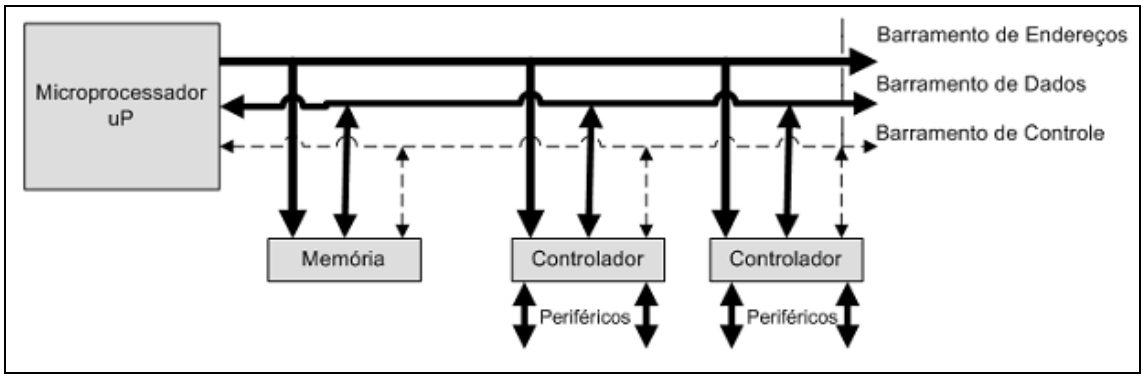


Figura 1 – Sistema baseado em um microprocessador.

Se houvesse somente um tipo de microcontrolador, este precisaria possuir muitos recursos para que se adaptasse a qualquer aplicação criando um dispositivo caro e complexo. Na prática, cada fabricante de microcontroladores oferece diversos modelos, desde os mais simples aos mais complexos. Desta forma, é possível escolher o microcontrolador a ser utilizado segundo diversos critérios: capacidade de memória, quantidade de portas de E/S, velocidade de processamento, conversor A/D e consumo de energia. Por isto, a escolha do microcontrolador é um fator importante na elaboração de um projeto, impactando no desempenho do dispositivo como um todo.

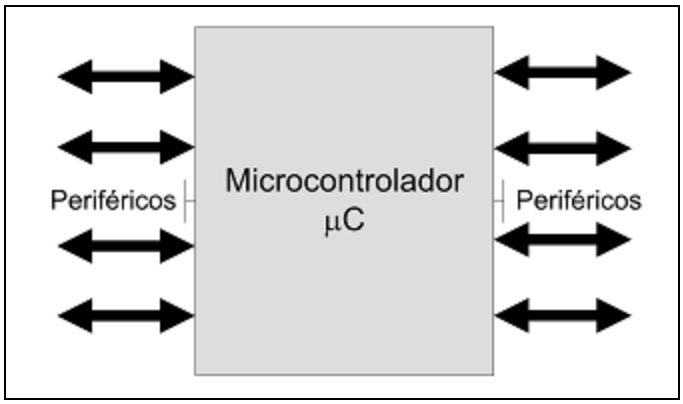


Figura 2 – Modelo do microcontrolador.

2.1.2. Mercado de microcontroladores

Existe uma grande diversidade de microcontroladores e sua classificação mais importante é quanto à sua arquitetura. Atualmente, existem microcontroladores de 2, 4, 8, 16 ou 32 bits, sendo que o desempenho dos microcontroladores de 16 e 32 bits são superiores aos de

4 e 8 bits, porém os microcontroladores de 8 bits dominam o mercado, pois na maioria das aplicações não é necessário utilizar dispositivos com maior capacidade de processamento, que são mais caros devido à sua maior complexidade.

Quanto às técnicas de fabricação destes circuitos, praticamente a totalidade dos microcontroladores utilizam a tecnologia CMOS (*Complementary Metal Oxide Semiconductor*). Esta tecnologia supera as demais por seu baixo consumo e alta imunidade ao ruído. Assim, as vendas de microcontroladores são distribuídas da seguinte forma [2]:

- Um terço do mercado está nas aplicações relacionadas a computadores e seus periféricos;
- Um quarto em aplicações de bens de consumo (eletrodomésticos, jogos, TV, áudio, etc.);
- 16% das vendas mundiais utilizadas em aplicações de comunicação;
- 16% empregados em aplicações industriais;
- Aproximadamente 10% na indústria de automação.

Quanto aos modernos microcontroladores de 32 bits, eles estão se popularizando no mercado em aplicações de processamento de imagens e áudio, comunicações, aplicações militares e dispositivos de armazenamento de dados [2].

2.1.3. Escolha de microcontroladores

Na hora de escolher qual microcontrolador utilizar em um projeto, uma série de fatores deve ser levada em consideração, como a disponibilidade de: documentação, ferramentas de desenvolvimento, preço, quantidade de fabricantes e as características do microcontrolador (tipo de memória de programa, número de temporizadores, interrupções, etc.).

Com relação ao custo, os fabricantes de microcontroladores competem duramente para vender seus produtos, sendo que as vendas de microcontroladores representam 10 vezes mais que as de microprocessadores. Os fabricantes normalmente disponibilizam suas próprias ferramentas de desenvolvimento específicas para cada tipo de microcontrolador, diminuindo assim o custo de sua aquisição [2].

Antes de selecionar um microcontrolador é importante analisar os requisitos da aplicação [6]:

- Processamento de dados: pode ser necessário que o microcontrolador realize cálculos críticos em um tempo limitado;
- Entrada e saída: para determinar as necessidades de Entrada/Saída do sistema é conveniente construir um diagrama de blocos do sistema;
- Consumo de energia: alguns produtos que utilizam microcontroladores são alimentados com baterias e seu funcionamento pode ser tão vital;
- Memória: Deve-se saber a quantidade de memória que será utilizada;
- Projeto da placa: a escolha do microcontrolador irá influir de forma direta neste item. Deve-se levar em conta que o microcontrolador não aumente os custos do projeto da placa de circuito impresso.

2.1.4. Recursos comuns a todos os microcontroladores

Inicialmente todos os microcontroladores adotavam a arquitetura *Von Neumann*, e atualmente alguns fabricantes usam a arquitetura *Harvard*. A arquitetura *Von Neumann* se caracteriza por dispor de uma memória principal que armazena dados e endereços e o acesso a esta memória é feito por meio de um sistema de barramentos único (dados, endereços e controle).

A arquitetura Harvard dispõe de memórias independentes, onde uma contém somente instruções e a outra, somente dados. Ambas dispõem de seus próprios sistemas de acesso e é possível realizar operações de acesso (escrita e leitura), simultaneamente em ambas as memórias [2].

2.1.4.1. Unidade central de processamento (CPU)

Este é o elemento mais importante do microcontrolador e determina suas principais características, tanto em nível de *hardware* com em nível de *software*. O CPU se encarrega de comandar a memória de instruções, receber e decodificar a instrução, assim como buscar os operandos e armazenar o resultado das operações realizadas.

Os processadores atuais, quanto à arquitetura e funcionalidade, podem ser classificados como [2]:

- CISC: um grande número dos processadores atuais, usados nos microcontroladores, é baseado na arquitetura CISC (*Complex Instruction Set Computer* - Computador com um Conjunto Complexo de Instruções). Dispõem em média cerca de 80 instruções de máquina no seu repertório, com algumas sofisticadas e potentes, necessitando muitos ciclos de relógio para sua execução. Uma vantagem dos processadores CISC é que estes oferecem ao programador instruções completas que funcionam como macros;
- RISC: tanto a indústria dos computadores pessoais, como a dos microcontroladores estão adotando a filosofia RISC (*Reduced Instruction Set Complex* – Conjunto reduzido de instruções Complexas). Nestes processadores o repertório de instruções é reduzido, com instruções simples executadas em um só ciclo. A rapidez destas instruções permitem otimizar o *hardware* e *software* do processador.

2.1.4.2. Memória

Nos microcontroladores, a memória de instruções e dados está inclusa no próprio *chip*. A parte não volátil desta memória (ROM) contém as instruções que controlam a aplicação. A outra parte, memória volátil (RAM), guarda as variáveis e dados utilizados no programa.

Algumas particularidades diferenciam os microcontroladores dos computadores pessoais, como a ausência nos microcontroladores de unidades de alta capacidade de armazenamento de dados. Como os microcontroladores executam um único programa a memória ROM necessária é reduzida, a memória RAM pode ser de pouca capacidade por ter que guardar apenas as variáveis e informações trocadas durante a execução do programa.

Também como o microcontrolador executa apenas um programa não é necessário guardar uma cópia do mesmo na RAM, sendo o programa executado diretamente na ROM. Os usuários de computadores pessoais estão acostumados a trabalhar com memória na ordem de *megabytes* ou *gigabytes*, porém, quem trabalha com programação de microcontroladores trabalha com capacidade de Memória ROM entre 512 bytes a 8 Kbytes e RAM entre 20 a 512 *bytes*. De acordo com o tipo de memória ROM que dispõe o microcontrolador, na aplicação, sua utilização pode ser diferente.

Abaixo, são descritos cinco tipos de memórias não voláteis disponíveis nos microcontroladores encontrados no mercado [2]:

- MROM (ROM com máscara): memória não volátil, somente de leitura cujo conteúdo se grava durante a fabricação do *chip*. O elevado custo da máscara de memória só faz seu uso ser recomendado quando se precisam de microcontroladores da ordem de milhões de unidades;
- PROM: tipo de memória não volátil gravada somente uma vez (OTP – *One Time Programmable* - Programável uma única vez). O usuário pode escrever o programa no *chip* com o uso de um gravador conectado ao PC. A versão OTP é recomendável quando o tempo de projeto e construção do produto são muito pequenos. Tanto este tipo de memória, quanto a EPROM, podem ter proteção do código gravado com a queima de fusíveis no próprio *chip*;
- UVE EPROM: os microcontroladores que dispõem de memória EPROM (*Erasable Programmable Read Only Memory*) podem ser gravados e apagados muitas vezes. A gravação é feita assim como nos dispositivos OTP com o uso de um gravador conectado ao PC. Quando é necessário apagar seu conteúdo, por disporem de uma janela de cristal, devem ser expostos a luz ultravioleta durante vários minutos;
- STANDART EEPROM: são memórias de somente leitura, programáveis e apagadas eletricamente EEPROM (*Electrical Erasable Programmable Read Only Memory*). Tanto a programação quanto o apagamento se realizam eletricamente no próprio gravador com um programa rodando em um PC. Esta operação é rápida e estes dispositivos não possuem a janela de cristal. Os microcontroladores com memória EEPROM uma vez instalados no circuito, podem ser gravados e apagados facilmente, quantas vezes forem necessárias, respeitando as limitações de ciclos de leitura e escrita desta memória, sem a necessidade de tirá-los do circuito. Para isto utilizam-se gravadores "*in-circuit*" que conferem rapidez e flexibilidade a operação. Uma tendência dos fabricantes de microcontroladores é incluir uma pequena área de memória EEPROM para que se possa armazenar e modificar uma série de parâmetros do sistema em uso. Neste tipo de memória o processo de escrita é relativamente lento;
- FLASH EEPROM: é uma memória não volátil de baixo consumo, que pode ser escrita e apagada. Funciona como uma ROM e uma RAM, consumindo menos energia elétrica. A memória FLASH é recomendada frente à EEPROM quando é necessária uma grande área de memória não volátil. As memórias FLASH e EEPROM são muito úteis por permitir que os microcontroladores possam ser reprogramados "*in-circuit*", sem a

necessidade de retirá-los das placas. Assim, um dispositivo que utiliza este tipo de memória incorporado ao controle de um motor de automóvel, por exemplo, permite que se possa modificar o programa de funcionamento durante uma manutenção periódica do veículo.

2.1.4.3. Circuito de relógio (Clock)

Todos os microcontroladores dispõem de um circuito oscilador, seja ele interno ou externo. Este gera uma onda quadrada de alta freqüência que comfigura os pulsos de relógio usados na sincronização das operações do sistema. Normalmente, o circuito oscilador está incorporado ao microcontrolador, sendo necessários apenas uns poucos componentes externos para estabilizar a freqüência de operação. Estes componentes consistem em um cristal de quartzo juntamente com elementos passivos ou o conjunto de um ressonador cerâmico com uma rede RC. Aumentar a freqüência do sinal de relógio supõe diminuir o tempo que se executa as instruções, porém isto implica em um aumento do consumo de energia.

2.1.4.4. Temporizadores (*Timers*)

São utilizados para controlar períodos de tempo (temporizadores), e eventos externos (contadores). Para medir tempos é feita a carga de um registro com um valor adequado que será incrementado ou decrementado de acordo com os pulsos de clock do sistema, até que este registro chegue a zero e seja gerado um aviso (interrupção ou *flag*). O contador é utilizado para contar eventos toda vez que é acionado e pode ser lido a qualquer momento pelo *software*.

2.1.4.5. Watchdog

Quando um computador pessoal trava, seja por uma falha de *software* ou por outra causa, se pressiona o botão de *reset* e ocorre a reinicialização do sistema. No caso do microcontrolador, este funciona sem o controle de um operador, 24 horas por dia. O *watchdog* consiste em um temporizador que quando ocorre estouro ou chega à zero provoca um reset automático do sistema, evitando assim que o software permaneça travado.

2.1.4.6. Proteção contra falha de alimentação (*Brownout*)

Trata-se de um circuito que desliga o microcontrolador quando a tensão de alimentação (Vdd) é inferior à tensão mínima (*brownout*). Enquanto a tensão de alimentação for inferior ao de *brownout*, o dispositivo se mantém desligado até a tensão voltar ao seu patamar normal de funcionamento. Esse recurso evita que haja mau funcionamento do dispositivo devido à baixa tensão de alimentação.

2.1.4.7. Modo de espera e/ou baixo consumo

Em muitas situações reais de trabalho, os circuitos microcontrolados devem esperar, sem executar ação nenhuma. Nestes casos para a economia de energia, que é fator chave em aplicações alimentadas por baterias, os microcontroladores dispõem de instruções que os colocam em estado de repouso ou de baixo consumo, com solicitações de consumo mínimas, voltando ao estado normal quando se necessita de processamento.

2.1.4.8. Conversor A/D e Conversor D/A

Os microcontroladores que incorporam um conversor analógico/digital (A/D) podem processar sinais analógicos, nas mais diversas aplicações. Estes dispõem de um multiplexador que permite à entrada de vários sinais analógicos utilizando apenas um conversor A/D.

O conversor D/A transforma os dados digitais obtidos no processamento da aplicação em um sinal analógico. O valor da tensão de saída varia de acordo com os bits enviados ao conversor D/A.

2.1.4.9. Comparador analógico e Modulador de largura de pulsos (PWM)

Alguns microcontroladores possuem internamente um amplificador operacional que atua como comparador de um sinal com uma referência fixa com outro sinal variável que se aplica por um dos pinos de entrada. A saída do comparador provoca um nível lógico 1 ou 0 de acordo com a comparação dos sinais de entrada.

PWM são circuitos que geram pulsos de largura (“*Duty Cicle*”) e freqüência programáveis, disponíveis em um pino de saída do microcontrolador para acionamento de cargas como motores, por exemplo.

2.1.4.10. Portas de E/S digitais (I/O) e Portas de comunicação

As portas de entrada/saída têm como função principal estabelecer a interface do programa contido na memória com o mundo externo, isto é, a aplicação, proporcionando suporte aos sinais de entrada, saída e controle. Geralmente estes pinos são agrupados em portas e podem ser comfigurados como E/S, carregando 1 ou 0 em um registro de comfiguração.

Com o objetivo de possibilitar ao microcontrolador comunicar-se com outros dispositivos externos (outros microcontroladores, redes, etc.) alguns modelos dispõem de portas de comunicação serial que podem ser [2]:

- UART: comunicação serial assíncrona;
- USART: comunicação serial síncrona/assíncrona;
- USB (Universal Serial Bus): moderno barramento serial utilizado nos PC‘s;
- I2C: interface serial a 3 fios desenvolvida pela Philips;
- CAN (*Controller Area Network*): protocolo de comunicação serial desenvolvido pela Bosch e Intel para comunicação em automóveis e adaptado a outros dispositivos;
- SPI (*Serial Peripheral Interface*): protocolo de comunicação que pode operar em modo mestre/escravo a 4 fios.

2.2. PIC 16F877A

O PIC (*Peripherals Integrated Controller* – Controlador Integrado de Periféricos) é um circuito integrado produzido pela Microchip Technology Inc. [4], que pertence à categoria dos microcontroladores, ou seja, um componente integrado que em um único dispositivo contém todos os circuitos necessários para realizar um completo sistema digital programável.

O PIC 16F877A, é um microcontrolador que agrupa em seu interior um número grande de recursos disponíveis, com encapsulamento de quarenta pinos. Cada pino com uma ou mais

funções bem definidas e a cada um é associado um nome que lembra a função correspondente. A definição dos terminais do PIC 16F877A é apresentada na figura 3 [5].

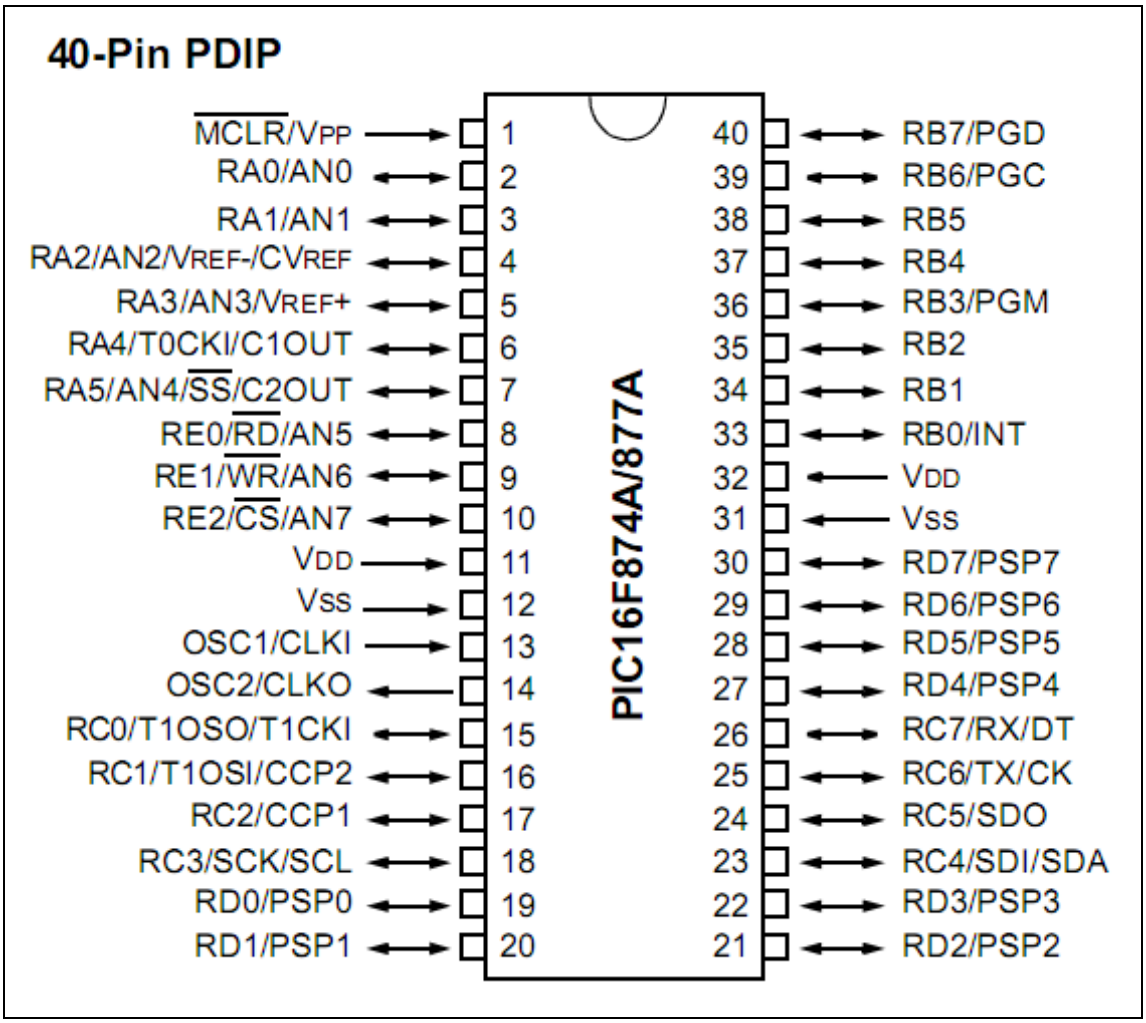
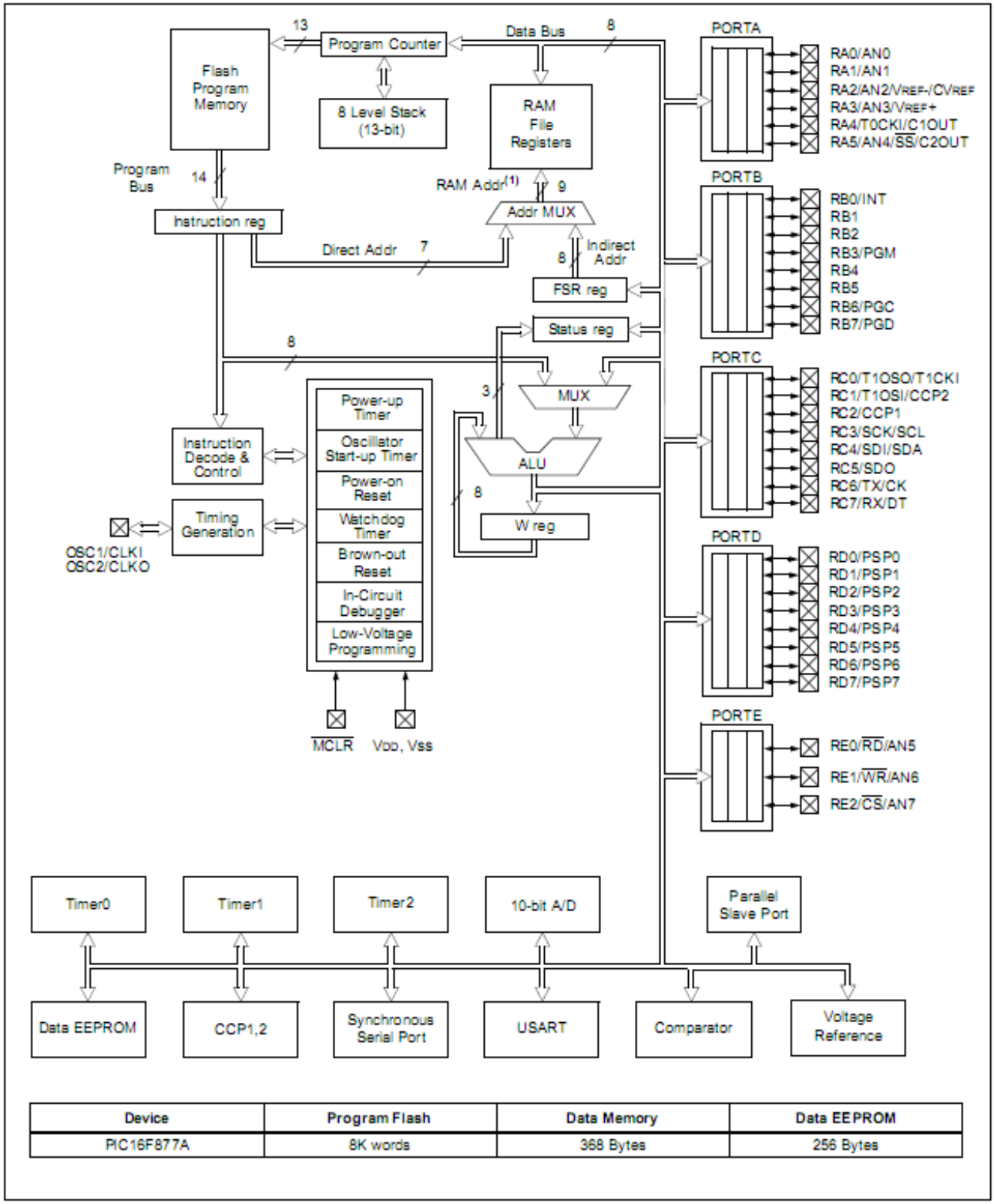


Figura 3 – Terminais do PIC 16F877A.

2.2.1. Estrutura Interna

A estrutura interna do PIC 16F877A é detalhada através do diagrama de blocos apresentado na figura 4 [5], que mostra todos os periféricos e comunicações que compõem o mesmo. Através do diagrama de blocos, pode-se visualizar as diversas partes que compõem o microcontrolador, tais como [5]:

- ULA (Unidade Lógica e Aritmética): é responsável pela execução das operações lógicas e aritméticas;
- Memória de Programa FLASH: é responsável pelo armazenamento do programa que será executado pelo microcontrolador;
- Memória de Dados RAM: utilizada para armazenar as variáveis de programa durante a sua execução;
- PORTs: são as portas de I/O que vão de PORTA a PORTE;
- *Timers*: são contadores que podem ser acessados diretamente na memória o PIC. São eles: TIMER0, TIMER1 e TIMER2;
- A/Ds: conversores A/D (Analógico/Digital) responsáveis pela conversão de sinais analógicos em sinais digitais;
- CCP: é possível acessar três recursos diferentes através dos módulos CCP (*Compare, Capture e PWM*);
- Portas de comunicação serial: SSP (*Synchronous Serial Port*) e USART. Através delas que é possível a comunicação do PIC com um PC via padrão RS-232;



Device	Program Flash	Data Memory	Data EEPROM
PIC16F877A	8K words	368 Bytes	256 Bytes

Figura 4 – Estrutura interna do PIC 16F877A.

2.2.2. Arquitetura e Conjunto de Instruções

O microcontrolador PIC 16F877A é baseado na arquitetura *Harvard*, que possui um barramento para dados e outro para programas e é representada pela figura 5 [6]. O primeiro sendo um barramento de 8 bits e o segundo de 14 bits. Esse tipo de arquitetura permite que, enquanto uma instrução é executada, outra seja "buscada" da memória, trabalhando de forma paralela. Esta arquitetura torna o processamento mais rápido e eficiente. Como o barramento de programa é maior que o de dados, o OPCODE (Código de Operação) da instrução já inclui o dado e o local onde ele vai operar (quando necessário), isto significa que apenas uma posição de memória é utilizada por instrução, economizando muita memória de programa [6].

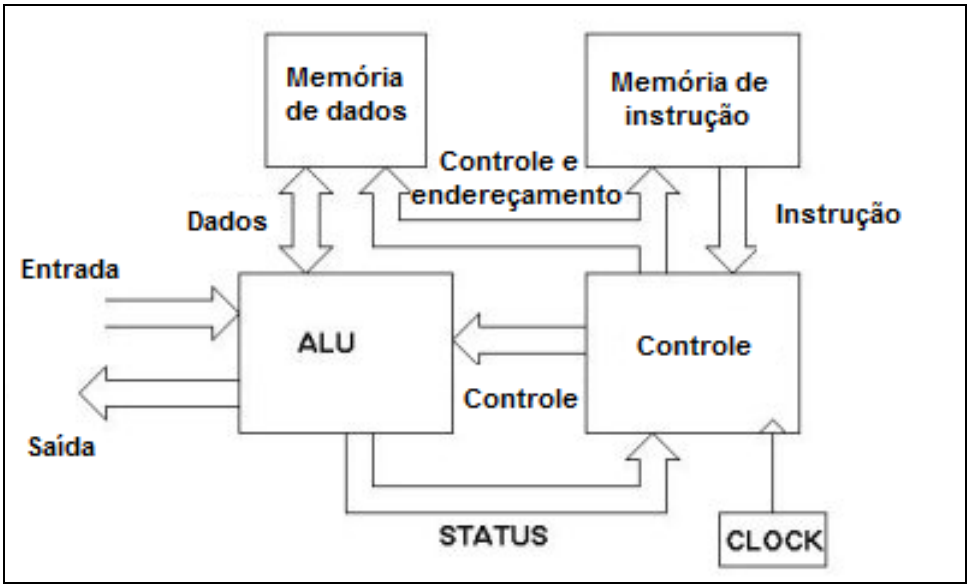


Figura 5 – Arquitetura *Harvard*.

O PIC 16F877A utiliza o conjunto de instruções RISC (*Reduced Instruction Set Computer* – Computador com conjunto de instruções reduzidas). No PIC 16F877A, essa tecnologia RISC possui apenas 35 instruções, o que é considerado pouco. Devido ao fato de conter poucas instruções, o circuito pode ser implementado em uma área menor, com uma conseqüente diminuição no consumo de energia e normalmente mais rápido. Por outro lado, implica no fato de que muitas funções devem ser construídas, pois não possuem a instrução equivalente.

2.2.3. Módulos CCP

Os módulos CCP oferecem três tipos de recursos, são eles: Captura, Comparação e PWM. O PIC 16F877A contém dois módulos de CCP, sendo eles CCP1 e CCP2. Como esses três recursos (*Compare/Capture/PWM*) estão relacionados ao mesmo módulo dentro do PIC, só pode ser utilizado um recurso de cada vez por módulo [5].

O PWM (*Pulse Width Modulation* - Modulação por Largura de Pulso) é o modo de funcionamento mais utilizado do módulo CCP, já que pode ser usado para o controle de motores e possibilita a criação de uma saída analógica após passar por um filtro externo, que varia de 0 a +5 VDC. O PWM é um sinal digital com freqüência fixa, mas com largura de pulso variável.

2.2.4. Temporizadores

Os timers são temporizadores internos que possuem características próprias, como limite de contagem, tipo de incremento, contagem de eventos externos, entre outras. O microcontrolador PIC 16F877A possui três *timers*, cada um com características diferentes de funcionamento, são eles: Timer0, Timer1 e Timer2.

2.2.4.1. Timer0

Timer0 ou TMR0 é um contador de 8 bits incrementado internamente pelo ciclo de máquina ou por um sinal externo. Caso sejam necessários intervalos de tempos maiores para o mesmo ciclo de máquina, utiliza-se o recurso de *prescal*e (PS) [5].

O *prescale* é um divisor de freqüência programável do sinal que incrementa o Timer0. Quando se usa um *prescale* de 1:1, cada ciclo de máquina corresponde a um incremento do Timer0 (unidade de Timer0). Ao se alterar o *prescale* para, por exemplo, 1:4 (os valores possíveis são as potências de dois até 256), o Timer0 será incrementado uma vez a cada quatro ciclos de máquina.

O Timer0 pode ser utilizado para duas funções básicas: contagem de eventos externos e temporização (contagem de tempo) quando a entrada de *clock* é proveniente do *clock* interno.

2.2.4.2. Timer1

Também conhecido como TMR1 é um módulo mais poderoso que o TMR0, pois pode ser configurado para funcionar como contador ou temporizador. É um timer de 16 bits composto de 2 registradores de 8 bits (TMR1H e TMR1L) que podem ser escritos e lidos pelo programador.

De forma semelhante ao TMR0, o TMR1 também pode ser incrementado pelo *clock* de máquina ou por um sinal externo, mas com uma vantagem: existe um circuito interno que possibilita ligar um cristal aos pinos RC0/T1OSO e RC1T1OSI.

O *Timer*1 pode operar de dois modos:

- Como *timer*: neste modo, o *Timer*1 incrementa a cada ciclo de instrução;
- Como contador: neste modo, o *Timer*1 incrementa a cada transição do sinal de *clock* externo.

O modo de operação é determinado pelo bit de seleção de *clock*, TMR1CS (bit 1 do Reg. T1CON). O Timer1 pode ser habilitado setando o bit de controle TMR1ON (bit 0 do registrador). O *Timer* também possui um reset interno que pode ser gerado pelos módulos "*Capture/Compare/PWM* - CCP". Quando o oscilador do Timer1 está habilitado (T1OSCEN setado), os pinos RC1/T1OSI/CCP2 e RC0/T1OSO/T1CKL tornam-se entradas independente do valor dos bits 1 e 0 do registrador TRIS, que são ignorados.

2.2.4.3. Timer2

O Timer2 ou TMR2, por sua vez, é um registrador de 8 bits, porém relacionado somente ao *clock* interno, mas possui duas vantagens sobre os demais:

- Além de possuir um *prescale* próprio, possui também um *postscale*. A diferença é que o *prescale* é utilizado para incrementar o TMR2, enquanto que o *postscale* conta a quantidade de estouros desse *timer* para poder gerar uma interrupção;
- Possui um seguindo registrador, o PR2. O PR2 é utilizado para controlar o estouro do TMR2. Ou seja, ele não conta de zero até o limite imposto pelos 8 bits. Desta forma sempre que TMR2 é igual PR2, o *timer* é "resetado", voltando a zero. Neste mesmo momento, o contador de *postscale* é incrementado. Quando o *postscale* terminar, a

interrupção associada ao Timer2 é gerada. O *prescale* e o *postscale* são zerados sempre que uma operação de escrita envolvendo os registradores TMR2 ou T2CON acontecer ou quando for realizado qualquer tipo de *reset* do PIC.

2.2.5. Conversor A/D

Um conversor A/D (Analógico/Digital) é um circuito que converte um nível (analógico) de tensão ou corrente em um valor numérico (digital) correspondente. O PIC 16F877A, possui um sistema de conversão A/D denominado conversão de aproximação sucessiva. Neste tipo de conversor, a conversão é realizada do bit mais significativo (Msb) para o bit menos significativo (Lsb).

O conversor A/D do PIC 16F877A possui algumas características, como: conversor interno de 10 bits, até oito canais de conversão e quatro tipos de referências: VDD e VSS (internos) e VREF+ e VREF- (externos).

2.2.6. Linguagem de Programação

O microcontrolador PIC16F877A pode ser programado tanto em linguagem de baixo nível (Assembly) como em linguagem de alto nível (Pascal, C, Basic). Como a linguagem de baixo nível não possui nenhum comando, instrução ou função além daquelas definidas no conjunto de instruções do processador utilizado, o desenvolvimento deste trabalho será usando a linguagem de programação C. Evitando assim um trabalho extra de programação que seria necessário no desenvolvimento de rotinas e operações que não fazem parte do conjunto de instruções do processador, e tornando o programa mais “enxuto” e menos complexo, e com isso mais fácil de ser seguido [3].

A utilização de uma linguagem de alto nível como C permite que o programador preocupe-se mais com a programação da aplicação em si, já que o compilador assume para si tarefas como controle e localização das variáveis, operações matemáticas e lógicas, verificação de bancos de memória. Desta forma, é possível dedicar o tempo basicamente à lógica do problema e não aos detalhes internos do *chip*.

2.3. Ambiente de desenvolvimento MikroC para PIC

Um dos fatores de maior importância na hora da escolha de um microcontrolador é o suporte tanto de *software*, quanto de *hardware* disponíveis. Um bom conjunto de ferramentas de desenvolvimento pode ser decisivo na escolha de um microcontrolador. As principais ferramentas de ajuda ao desenvolvimento de sistemas embarcados em microcontroladores são [4]:

- Assemblador (*Assembler*): a programação em linguagem assembly pode ser uma tarefa árdua para um iniciante. Permite desenvolver programas muito eficientes e que ocupam pouca memória, dando ao programador o domínio absoluto do sistema. Os fabricantes fornecem gratuitamente o assemblador para os microcontroladores mais populares;
- Compilador: a programação em alto nível (C, C++) permite diminuir o tempo de desenvolvimento do produto. Mas a programação deve ser bem estruturada, pois caso contrário, a eficiência do programa fica comprometida. Apesar de as versões mais avançadas serem caras, os fabricantes de modelos mais populares disponibilizam compiladores gratuitos com versões limitadas;
- Simulador: são capazes de executar em um PC programas para o microcontrolador. Os simuladores permitem ter controle sobre a execução do programa, sendo ideais para a depuração dos mesmos. Seu grande inconveniente é simular a entrada e saída de dados no microcontrolador;
- Placas de desenvolvimento: são placas com um microcontrolador já montado, o qual pode ser conectado a um PC para programar o microcontrolador. Estas placas podem incluir displays LCD, teclados, *leds*, pinos de E/S, etc.
- Emuladores "*in circuit*": Faz a interface entre o programa monitor e o circuito do microcontrolador, permitindo depuração e simulação do *firmware* diretamente no *chip*.

MikroC [7] é uma poderosa ferramenta para o desenvolvimento de soluções para microcontoladores PIC. Foi desenvolvida para facilitar a vida do programador e utiliza a linguagem C para programação em alto nível.

O ambiente MickroC é um IDE (*Integrated Development Environment* - Ambiente Integrado de Desenvolvimento), composto de um editor de arquivos, montador Assembly, compilador C e C++, ligador, simulador e emulador.

Este ambiente é estruturado em *workspaces* ou espaços de trabalho que são módulos que podem agregar um ou mais projetos. Um projeto pode conter um ou mais arquivos de códigos fonte, utilizados para gerar um arquivo binário que será utilizado na simulação e programação do microcontrolador.

2.4. Motor de passo

O motor de passo é um transdutor que converte energia elétrica em movimento controlado através de pulsos, o que possibilita o deslocamento por passo, onde passo é o menor deslocamento angular [8].

Devido à total adaptação do motor de passo por controles digitais e, principalmente, pelo tamanho e custo reduzidos, houve um aumento na popularidade deste motor. Pode-se destacar também a estabilidade do motor de passo em relação aos outros motores. Para se obter uma rotação específica de um determinado grau, basta calcular o número de rotação por pulsos, o que possibilita uma boa precisão no movimento [8].

O motor de passo pode ser divido em duas partes, da seguinte maneira:

- Rotor: é denominado rotor o conjunto eixo-imã que rodam solidariamente na parte móvel do motor;
- Estator: define-se como estator a trave fixa onde as bobinas são enroladas.

Na figura 6 [9], podemos observar as partes mencionadas.

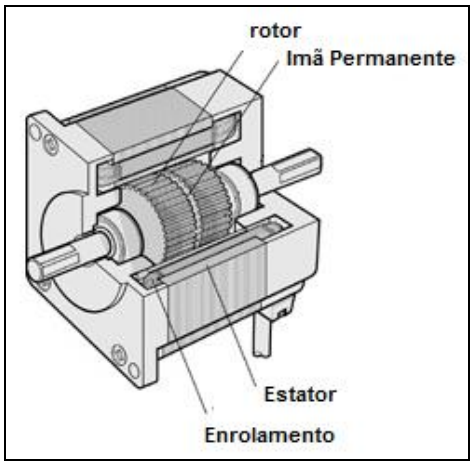


Figura 6 – Componentes do motor de passo.

Como os motores de passos têm movimentos precisos, qualquer equipamento que necessite de precisão de movimento pode utilizar estes motores. Podemos citar, por exemplo, o controle de micro-câmeras num circuito interno de vigilância, furação automática de acordo com instruções em fita sobre as posições dos furos, controle de leitores de disquete e CD, impressoras e robôs [8].

2.4.1. Tipos de Motores de Passo

Normalmente, os motores de passo são projetados com enrolamento de estator polifásico, semelhante aos demais motores. O número de pólos é determinado pelo passo angular desejado por pulsos de entrada. Os pulsos passam pelo circuito de alimentação, oferecendo corrente aos enrolamentos certos para fornecer o deslocamento desejado. Abaixo, estão citados os tipos de motores de passo [8]:

- Relutância Variável: apresenta um rotor com muitas polaridades construídas a partir de ferro doce, com estator laminado. Por não possuir imã, quando energizado, apresenta torque estático nulo. Tendo assim, baixa inércia de rotor e não pode ser utilizado com uma carga inercial grande.
- Imã Permanente: apresenta um rotor de material alnico ou ferrite e é magnetizado radialmente. Devido a isto, o torque estático não é nulo.
- Híbridos: é uma mistura dos dois anteriores e apresenta rotor e estator multidentados. O rotor é de imã permanente e magnetizado axialmente. Apresentam grande precisão (3%), boa relação torque/tamanho e ângulos pequenos (0,9 e 1,8 graus). Para que o rotor avance um passo, é necessário que a polaridade magnética de um dente do estator se alinhe com a polaridade magnética oposta de um dente do rotor.

Com relação aos motores com imã permanente, além do número de fases do motor, existe outra subdivisão entre estes componentes: a sua polaridade. Motores de passo unipolares são caracterizados por possuírem um *center-tap* entre o enrolamento de suas bobinas. Normalmente utiliza-se este *center-tap* para alimentar o motor, que é controlado aterrando-se as extremidades dos enrolamentos. Na figura 7, temos um motor de duas bobinas e quatro fases.

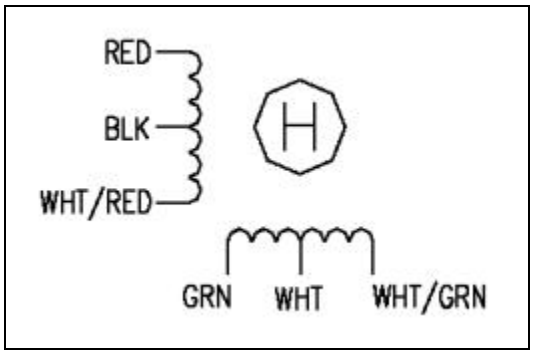


Figura 7 – Representação dos enrolamentos do motor de passo unipolar.

Diferentes dos unipolares, os motores bipolares exigem circuitos mais complexos. A grande vantagem em se usar os bipolares é prover maior torque, além de ter uma maior proporção entre tamanho e torque. Fisicamente, os motores têm enrolamentos separados, sendo necessária uma polarização reversa durante a operação para o passo acontecer. Na figura 8, temos um motor bipolar.

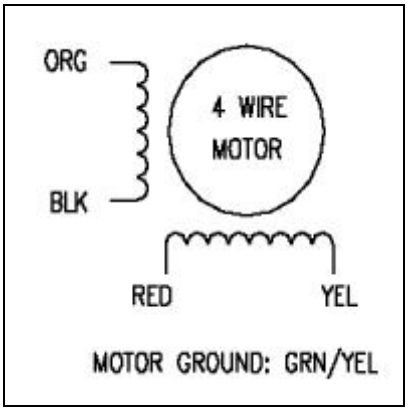


Figura 8 – Representação dos enrolamentos do motor de passo bipolar.

2.4.2. Funcionamento do motor de passo

O funcionamento do motor de passo é baseado em uma alimentação seqüencial e repetitiva, que respeita a seqüência requerida pelo motor. Existem três tipos básicos de movimentos, tanto para o motor bipolar como para o unipolar: passo inteiro, meio passo e o micro passo.

O micro passo tem sua tecnologia não muito divulgada, e baseia-se no controle da corrente que flui por cada bobina multiplicada pelo numero de passos por revolução. A energização de somente uma bobina de cada vez, produz um pequeno deslocamento no rotor. Este deslocamento ocorre simplesmente pelo fato de o rotor ser magneticamente ativo e a energização das bobinas criarem um campo magnético intenso que atua no sentido de se alinhar às pás do rotor. Assim, polarizando de forma adequada as bobinas, podemos movimentar o rotor somente entre as bobinas (passo inteiro), ou entre as bobinas e alinhadas com as mesmas. Na figura 9 [8], temos um exemplo de alinhamento do campo para o caso do motor bipolar, utilizando passo inteiro [8].

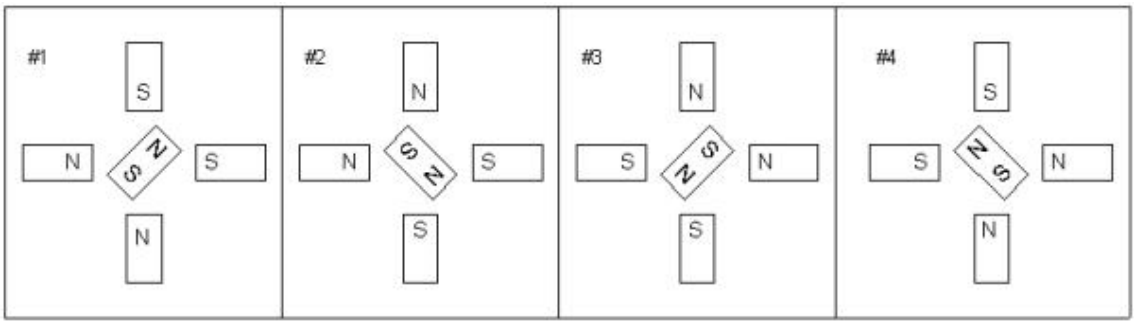


Figura 9 – Alinhamento entre as bobinas para motor bipolar com passo inteiro.

Para que se obtenha uma rotação constante é necessário que a energização das bobinas seja periódica. Esta periodicidade é proporcionada por circuitos eletrônicos que controlam a velocidade e o sentido de rotação do motor.

O pequeno ângulo deslocado pelo rotor depende do número de dentes e do número de fases do motor. Em geral, o número de dentes do rotor multiplicado pelo número de fases revela o número de passos por revolução.

Os motores de passo possuem uma curva de Torque vs. Velocidade representada pela figura 10. A região de partida é a área onde o motor permanece parado, mesmo com pulsos em sua entrada. A região entre curva de *Pull-in* e a curva *Pull-out* é a área ótima de trabalho, onde há grande precisão de movimento. Após a curva de *Pull-out*, pode-se trabalhar com uma velocidade maior, porém há risco de se perder sincronia e, conseqüentemente, perder a precisão.

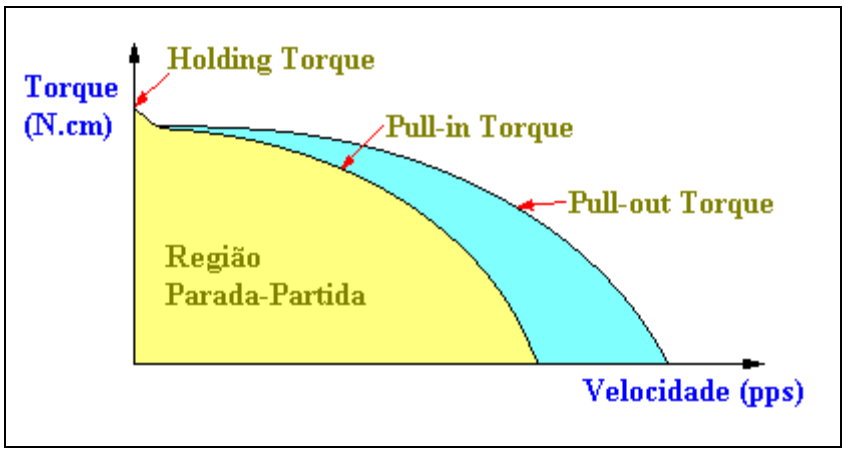


Figura 10 – Curva característica Torque vs. Velocidade do motor de passo.

2.4.3. Vantagens e desvantagens

Em relação aos primeiros motores, o motor de passos apresenta evidentes vantagens como tamanho e custo reduzidos, total adaptação à lógica digital (o que permite o controle preciso da velocidade direção e distância), características de bloqueio, pouco desgaste e dispensa realimentação.

São poucas as desvantagens, mais elas existem: má relação potência/tamanho, e principalmente, controle relativamente complexo, que necessita de *hardware* externo para funcionar [8].

2.5. *Driver* de potência

Alguns motores de passo necessitam de um circuito amplificador de saída, pois algumas aplicações exigem uma demanda de corrente relativamente elevada. Caberia ao circuito amplificador de saída fornecer estas correntes de forma segura, econômica e rápida. O circuito amplificador de saída seria constituído de transistores e/ou dispositivos de potência que drenam corrente em torno de 500 mA ou mais [8]. Motores de passo geralmente suportam correntes acima de 1,5 A. O amplificador de saída é o dispositivo mais solicitado em um projeto de controle de motor de passo. Devido às variações de trabalho a que pode ser submetido o motor de passo, um amplificador mal projetado pode limitar muito o conjunto como um todo.

O torque do motor de passo depende da freqüência aplicada à alimentação. Quanto maior a freqüência, menor o torque, porque o rotor tem menos tempo para mover-se de um ângulo para outro.

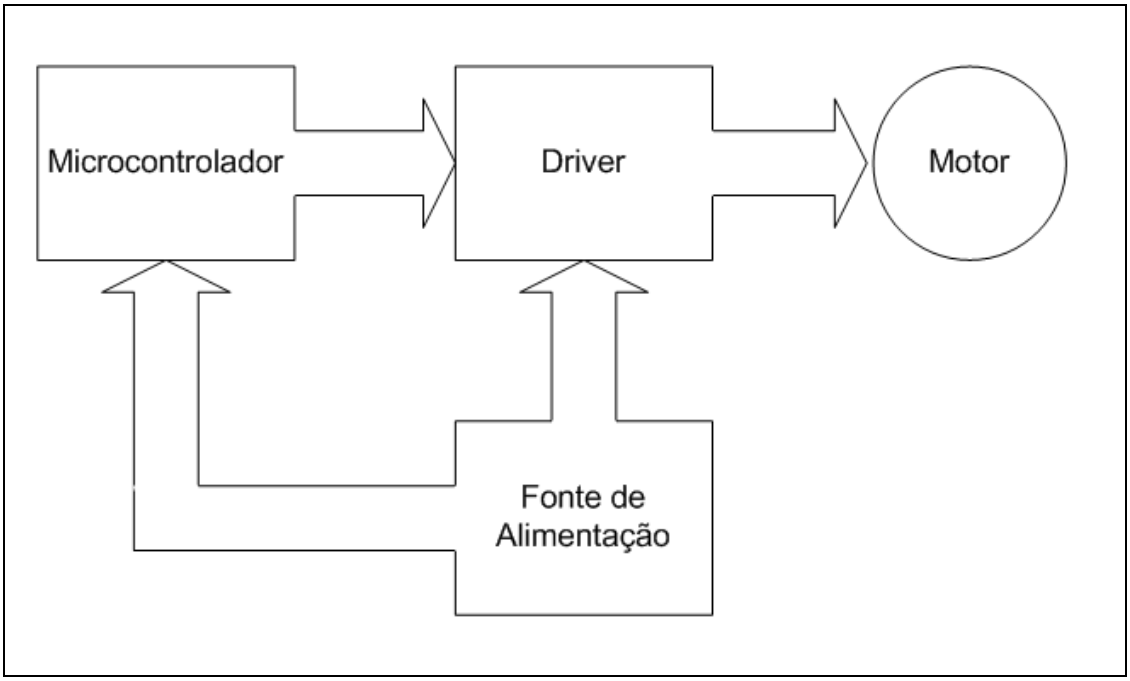


Figura 11 – Diagrama para a ligação do motor.

Na figura 11, temos o diagrama de blocos para o motor de passo. Uma vez que o *driver* já possui projetado em seu *hardware* a seqüência de energização das bobinas do motor, o único trabalho do microcontrolador é dizer qual a direção de rotação do motor (horário ou anti-horário) e fornecer os pulsos com a freqüência adequada.

3. Metodologia

3.1. Materiais e métodos

O projeto foi desenvolvido para criar um método de varredura automática a fim de analisar transdutores de ultra-som. Para isto, utiliza-se o tanque da figura 12, o qual disponibiliza de três motores de passo que geram deslocamentos lineares nos três eixos.

Figura 12 – Tanque de varredura.

O microcontrolador PIC 16F877A foi escolhido devido a sua grande versatilidade e diversidade de recursos. O PIC é utilizado neste trabalho para realizar o controle de três

motores de passo SM 1.8D1-SE, da fabricante Syncro. O acionamento foi realizado por três *drivers* bipolares *chopperizados* SYN-D2.28, também da fabricante Syncro.

O controle é realizado nos três eixos, sendo eles definidos como: Eixo x, Eixo y e Eixo x. Portanto, é possível o deslocamento nas três dimensões, aqui definidos como: direita e esquerda (Eixo x), cima e baixo (Eixo y), frente e trás (Eixo z).

A figura 13 ilustra o diagrama de blocos do PIC, *drivers*, motores e leds de sinalização. Foram usadas também três fontes de corrente contínua que fornecem aos *drivers* as tensões de +38VDC e +5VDC.

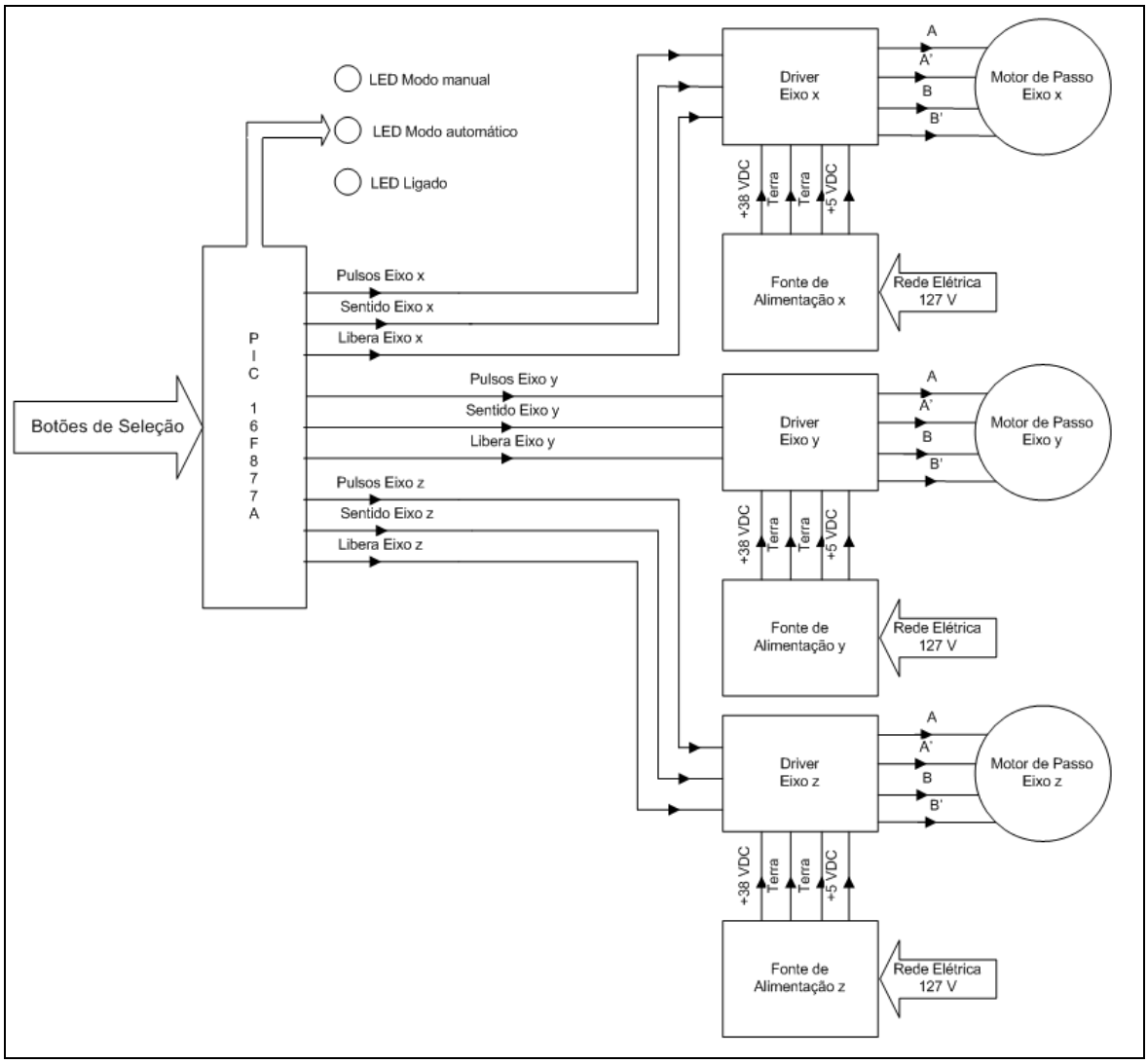


Figura 13 – Diagrama de blocos do sistema de movimentação.

3.2. Fluxograma de controle

O programa foi projetado para que o usuário opte por duas escolhas: modo manual e modo automático. A figura 14 mostra o fluxograma geral do programa.

O modo manual permite que haja o ajuste da posição do sensor nas três dimensões, definindo o ponto central da varredura. Uma vez posicionado, pode-se utilizar o modo automático, que fará a varredura no tanque em 41 planos de 41 mm x 41 mm, espaçados de 1 mm cada. Assim, teremos um volume total varrido de 41 mm x 41 mm x 41 mm. Este volume é pré-definido para que haja o mapeamento do campo acústico deste projeto. Um *led* indicador mostra que o programa está funcionando corretamente e que está pronto para o usuário utilizar os botões.

No modelo estudado, uma volta completa no motor de passo significa um deslocamento linear de 5 mm nos eixos. Uma vez que o motor necessita 400 passos para uma revolução completa, são necessários 80 passos para que haja o deslocamento de 1 mm.

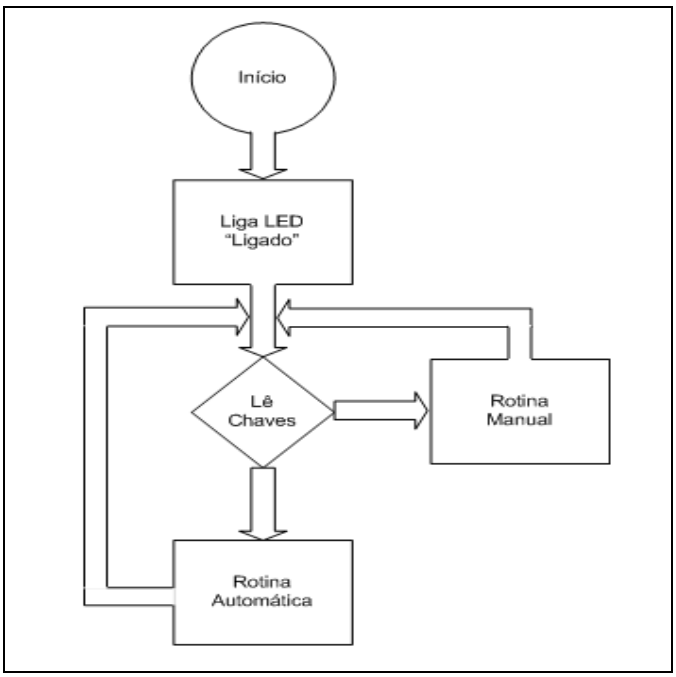


Figura 14 – Fluxograma geral do programa.

O gráfico da figura 15 representa a curva de torque vs. velocidade do motor utilizado no projeto. Foram utilizadas diferentes freqüências para cada propósito, sempre respeitando a região de operação ótima do motor para que não houvesse perda de sincronismo.

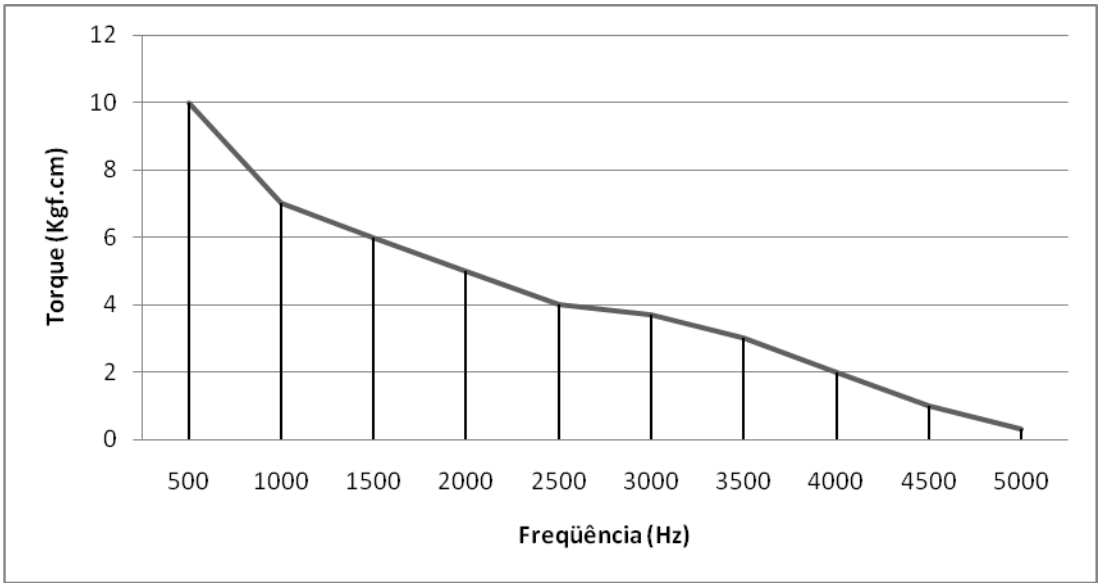


Figura 15 – Curva de torque do motor utilizado.

3.2.1. Modo manual

No modo manual, o controle é feito pelo o usuário. É possível navegar o sensor pelo tanque através dos botões de entrada. Neste modo, a freqüência utilizada é de 500 Hz, possibilitando um grande torque e uma velocidade reduzida para maior precisão e suavidade no deslocamento, segundo o gráfico de torque do motor na figura 15. As decisões são tomadas conforme o fluxograma da figura 16.

Quando um botão é pressionado, o microcontrolador identifica qual foi o botão e da um passo na direção selecionada. Isto significa que, a cada passo, há uma rotação de 0,9° no rotor ou 0,0125 mm em algum dos eixos. Caso o usuário queira um deslocamento mais rápido, basta manter pressionado o botão e pulsos serão gerados seqüencialmente, até que o botão seja liberado. Outro *led* é utilizado para indicar que o programa está operando no modo manual.

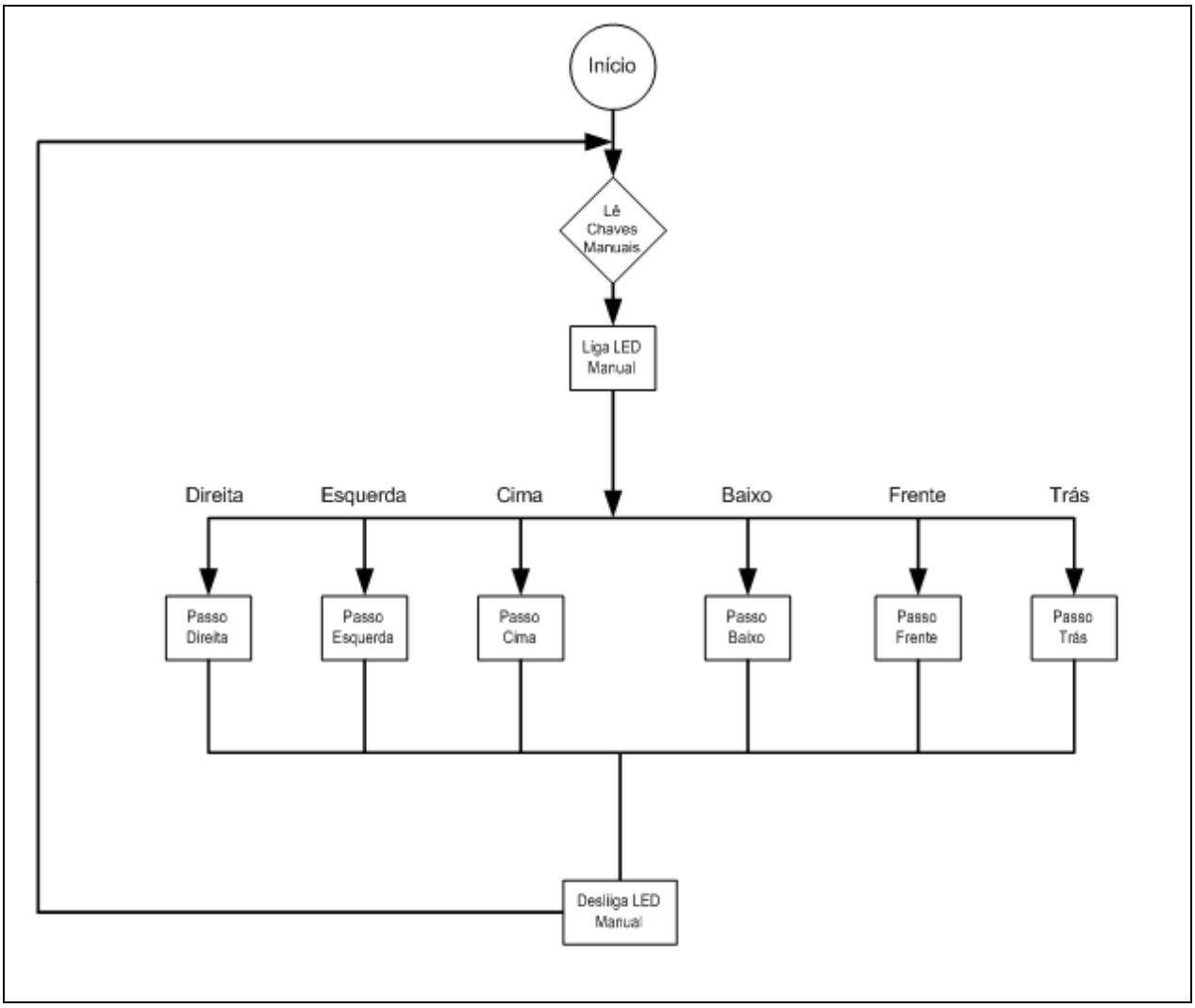


Figura 16 – Fluxograma da rotina manual.

3.2.2. Modo automático

O modo automático foi projetado para fazer a varredura no volume de 41 mm x 41 mm x 41 mm. Após o posicionamento do sensor (através do modo manual), a rotina de modo automático irá deslocar o sensor até a ponta superior esquerda de um “cubo imaginário”, onde será o ponto inicial da varredura. A varredura é feita conforme a figura 17: segue-se uma linha da esquerda para a direita, indicando o movimento no eixo x; após completar 41 mm de deslocamento no eixo x, há o deslocamento de 1 mm para baixo (eixo y) e então, se dá o deslocamento de mais 41 mm no eixo x, desta vez, da direita para a esquerda. O processo é

repetido até se varrer totalmente o primeiro plano xy e então, há o deslocamento de 1 mm para frente (eixo z). A varredura é feita nos próximos planos xy até se completar o cubo.

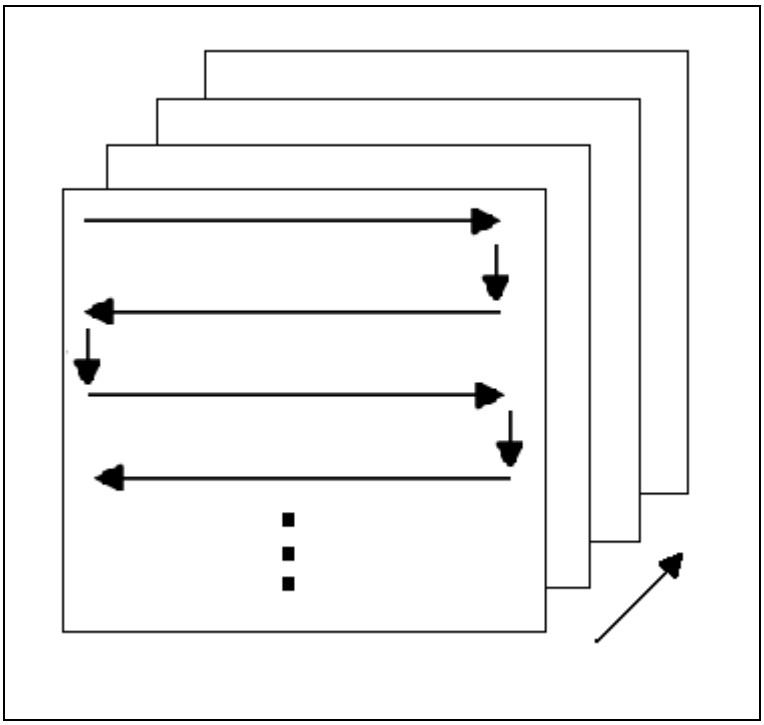


Figura 17 – Varredura do modo automático.

Para que o deslocamento seja preciso, foi criada uma rotina no programa que gera pulsos de freqüência variável para cada milímetro deslocado. Os 20 primeiros passos são os passos de aceleração, onde a freqüência inicia-se em 500 Hz e vai subindo gradualmente até chegar a 2000 Hz. Nos próximos 50 passos, a freqüência permanece constante em 2000 Hz. Os últimos 10 passos são os passos de desaceleração, onde a freqüência, que inicialmente era de 2000 Hz, chega a 500 Hz novamente, de forma gradual. A cada 80 passos no eixo x, é gerada uma *flag* de alerta para realizar as medidas.

Uma rotina de *delay* fixo com um multiplicador foi criada para que houvesse um período variável de acordo com o passo. Assim, quanto maior a freqüência, menor é o multiplicador. Para determinar a freqüência teoricamente, utilizou-se a ferramenta de contagem de ciclos do programa MikroC (ver anexo A). Na Tabela 2 temos, para base de cálculo, que cada ciclo leva 0,5 µs. Portanto, basta multiplicar pelo número de ciclos para estimar o período. Porém, o período real possui uma pequena diferença e foi ajustada finamente com a ajuda do

osciloscópio. A figura 18 representa a variação da freqüência em relação ao passo no modo automático, seguindo a contagem de ciclos.

Tabela 2 – Número de ciclos e freqüência no modo automático.

Passo	Ciclos	Freqüência
1	1953	512
2	1877	533
3	1801	555
4	1725	580
5	1649	606
6	1573	636
7	1497	668
8	1424	702
9	1345	743
10	1269	788
11	1193	838
12	1117	895
13	1041	961
14	965	1036
15	889	1125
16	813	1230
17	737	1357
18	661	1513
19	585	1709
20	509	1965
21-70	433	2309
71	509	1965
72	661	1513
73	813	1230
74	965	1036
75	1117	895
76	1269	788
77	1421	704
78	1573	636
79	1725	580
80	1877	533

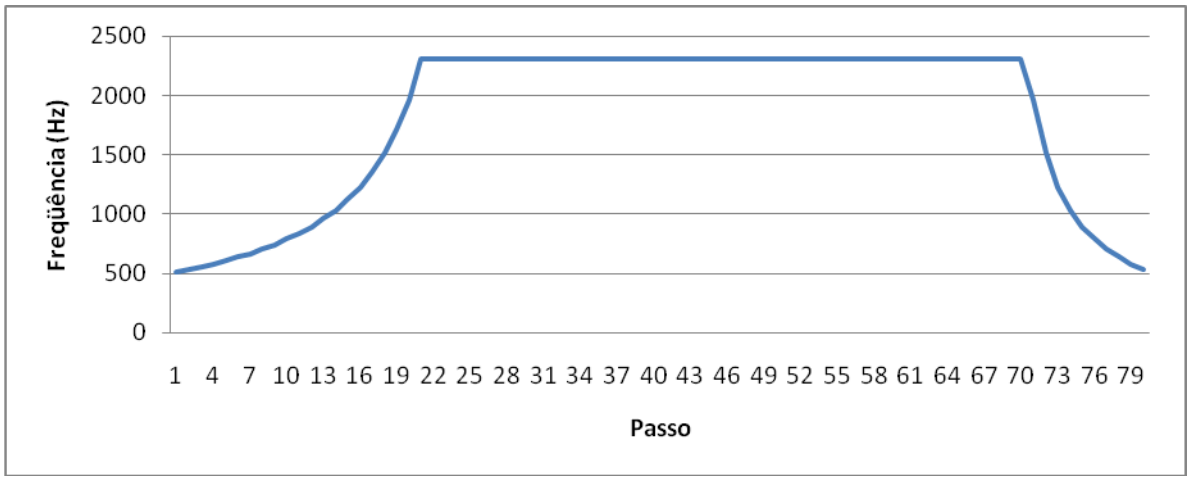


Figura 18 - Gráfico Freqüência vs. Passo

A figura 19 mostra o fluxograma do modo automático. A decisão é tomada a partir de contadores de controle, que mostram em qual passo o motor está e sua posição espacial no tanque.

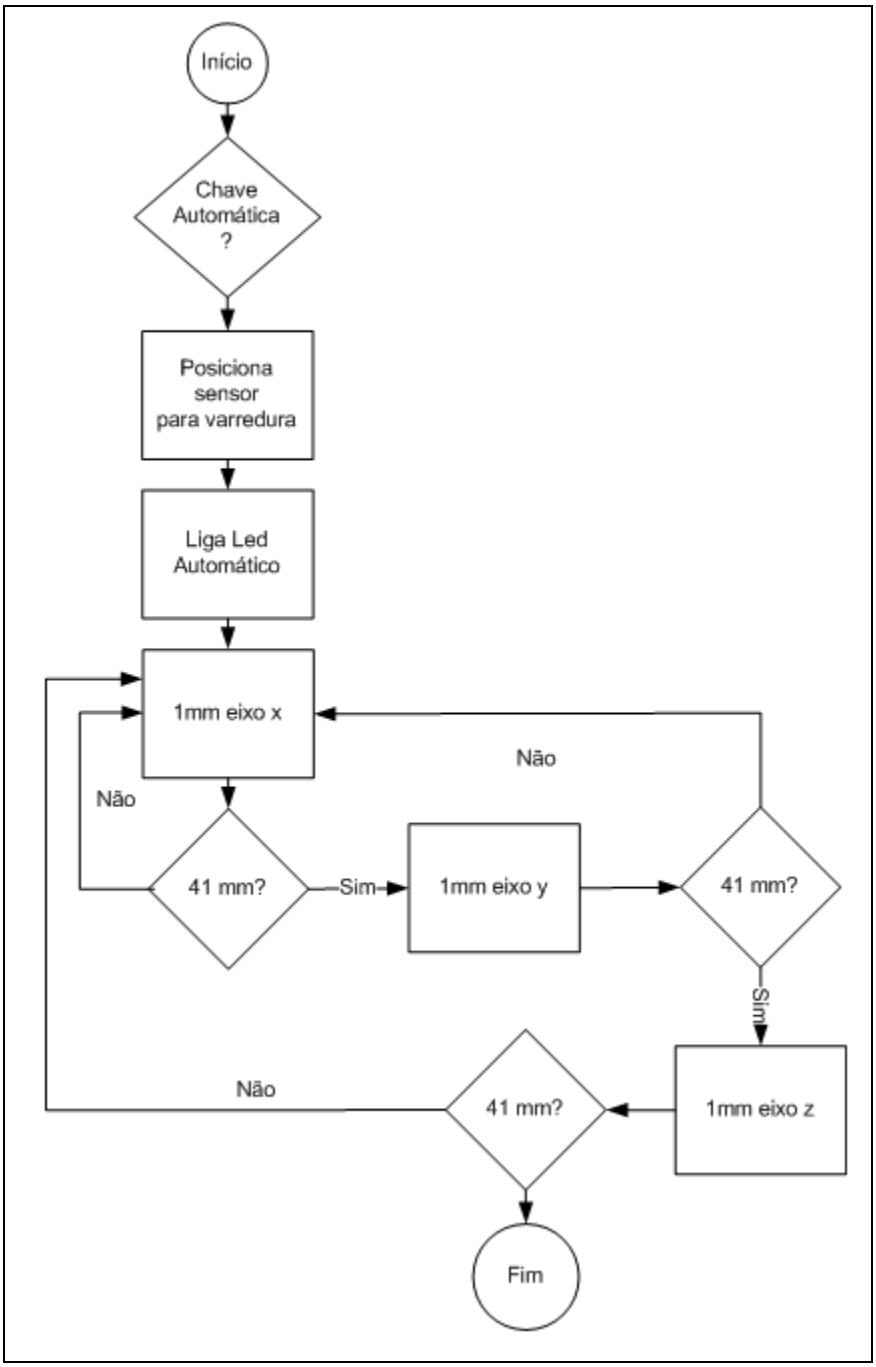


Figura 19 – Fluxograma da rotina automática.

3.3. Resultados e discussões

O gráfico da figura 20 representa a amostragem da saída do PIC no osciloscópio do modo manual e apresenta os pulsos sendo gerados seqüencialmente, com a freqüência constante de 500 Hz e tensão de +5 VDC. A onda quadrada gerada é simétrica (*Duty Cicle* 50%). Nota-se que, assim que o botão é pressionado, os pulsos gerados movimentam o motor para posição desejada no tanque e cessam assim que o botão é liberado.

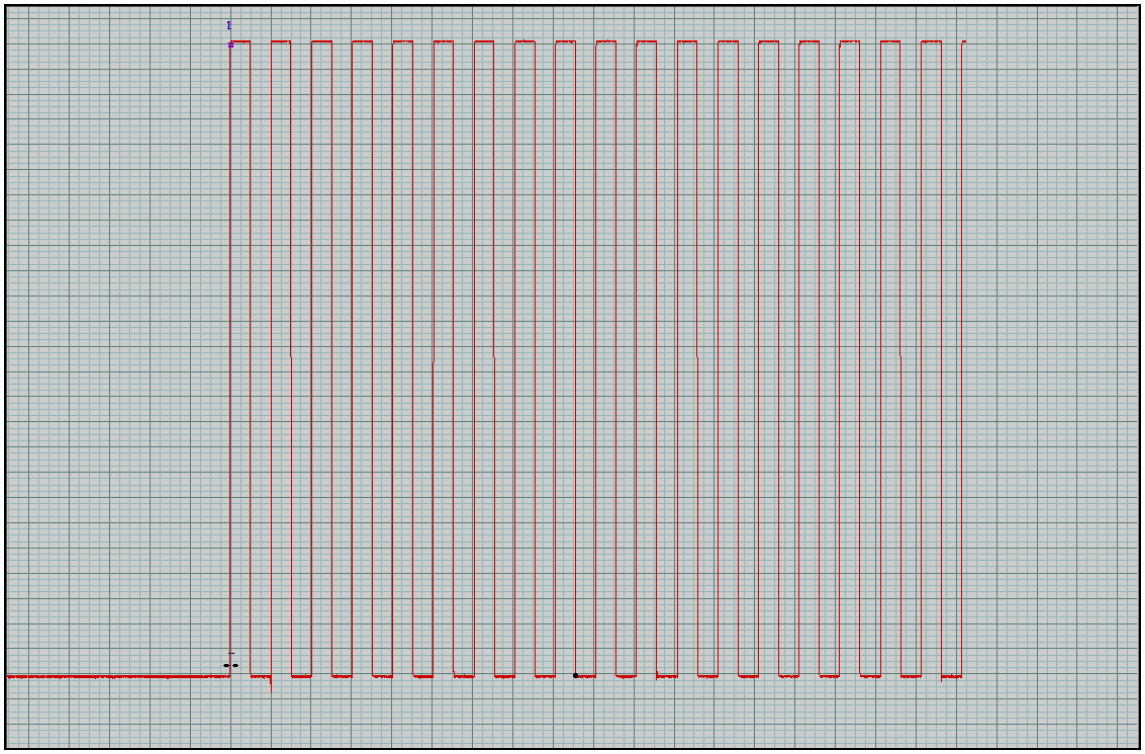


Figura 20 – Medida do osciloscópio para passos no modo manual.

A figura 21 apresenta a amostragem do osciloscópio para a onda quadrada com freqüência variável e +5VDC. Nota-se que nos 20 primeiros passos, o período da onda diminui a cada passo e acelera o motor. Nos próximos 50 passos, a freqüência fica constante em 2000 Hz e a velocidade do motor se mantém a mesma. O período voltar a aumentar nos últimos 10 passos, fazendo com que o motor desacelere sem solavancos. Esta seqüência de pulsos é gerada de acordo com a descrição do modo automático para cada eixo.

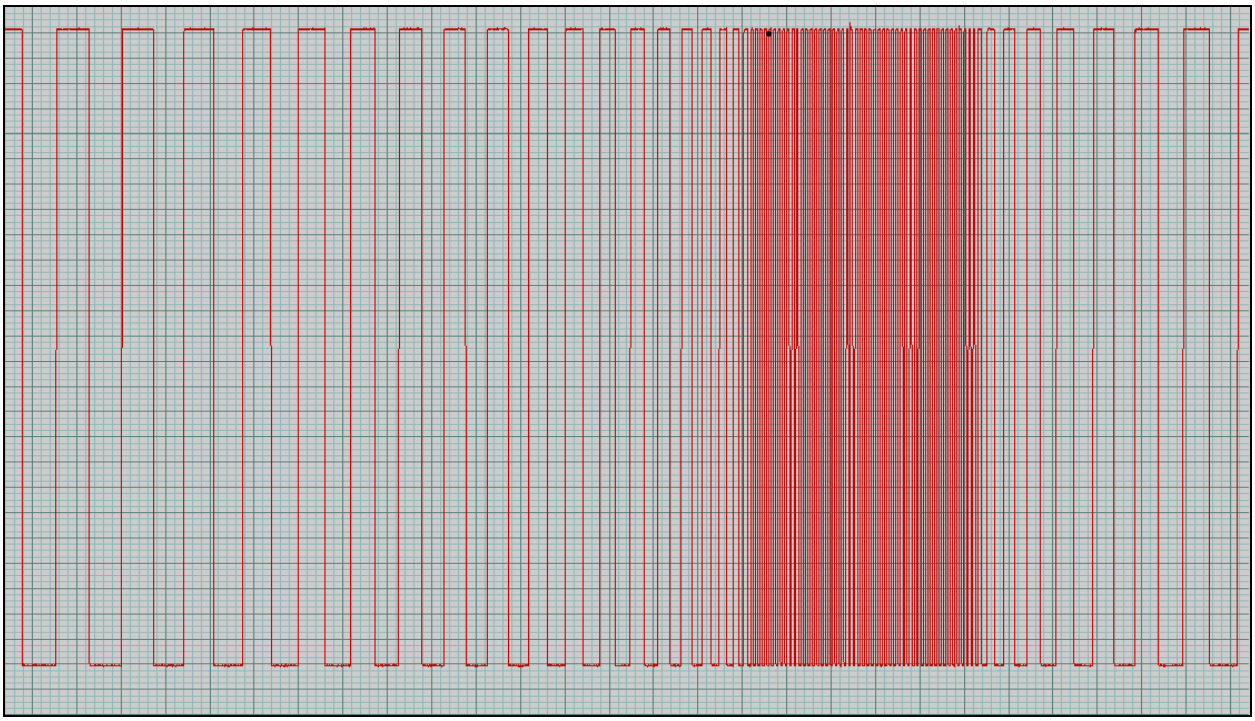


Figura 21 – Medida do osciloscópio para passos no modo automático.

4. Conclusão

A utilização de microcontroladores em processos digitais mostrou a sua grande capacidade de processamento e versatilidade. Rotinas complexas de controle foram facilitadas pela programação em C, que dispõe de muitos recursos e maneiras alternativas para realizar o controle.

Testes no laboratório e com osciloscópio revelaram a exatidão do motor de passo, sendo perfeito para a aplicação escolhida. O sistema de posicionamento foi aplicado com sucesso, cumprindo com o objetivo esperado e realizando os processos de varredura de forma precisa.

Com a realização deste trabalho, foi possível colocar em prática muitos dos conhecimentos adquiridos no transcorrer do curso de Engenharia Elétrica, dando ênfase na parte de programação e controle. Os conhecimentos sobre microcontroladores ajudaram na fase de escolha do dispositivo a ser utilizado e na programação, visto que foi necessário foi necessário dedicar grande parte do trabalho para estudá-los.

O objetivo deste trabalho foi alcançado, solucionando problemas relacionados à varredura no tanque a fim de calibrar os sensores de ultra-som. O trabalho contribui para que os sensores sejam calibrados da forma correta, evitando assim erros de medidas e possíveis perigos aos seres humanos, como no caso aplicações médicas.

Referências bibliográficas

[1] INMETRO E SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL (SENAI). **Vocabulário Intenacional de Termos Fundamentais e Gerais de Metrologia (VIM)**. Inmetro e Senai. Rio de Janeiro, 2007.

[2] USATEGUI, J. M. A.; MARTINEZ I. A. **Microcontroladores PIC. Diseño práctico de aplicaciones**. Editora McGraw-Hill. Aravaca, 2003.

[3] PEREIRA, F. **Microcontroladores PIC - Programação em C**. 7 ed. Editora Érica. São Paulo, 2009.

[4] MICROCHIP TECHNOLOGY INC. Disponível em: <http:// www.microchip.com>.

[5] MICROCHIP PIC16F877A – **Datasheet**. Disponível em: <http://www.microchip.com>.

[6] SOUZA, D. J. ; LAVINIA, N. C. **Conectando o PIC 16F877A: Recursos Avançados**. 4 ed. Editora Érica. São Paulo, 2007.

[7] MIKROC. Disponível em: <http://www.mikroe.com/en/compilers/mikroc/pic>.

[8] MOTOR DE PASSO. Disponível em: <http://recreio.gta.ufrj.br/grad/01_1/motor>.

[9] STEP MOTOR. Disponível em: <http://www.orientalmotor.com/MotionControl101/2phase-v-5phase.html>.

[10] COSTA-FELIX, R.P.B. **Aplicações metrológicas do ultra-som empregado em engenharia biomédica utilizando varreduras de senos (Chirps)**. Tese de Doutorado, Programa de Engenharia Biomedica, COPPE/UFRJ. Rio de Janeiro, 2005.

Anexo A

A.1 Criando um projeto em linguagem C

Para criar um projeto, devem-se seguir alguns passos. A tela inicial irá mostrar uma tela semelhante à figura 22. Para iniciar um novo projeto, devemos clicar no ícone da figura 23, localizado na barra de ferramentas principal.

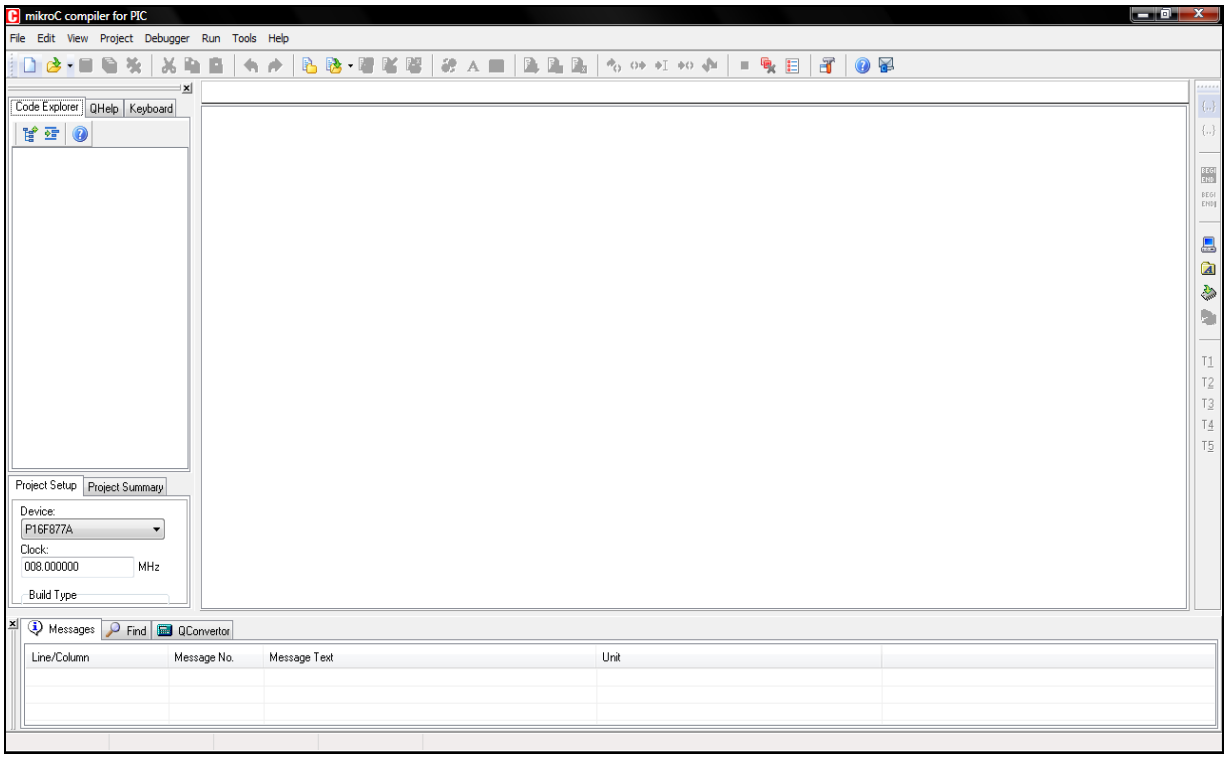


Figura 22 – Tela inicial no MikroC.

Figura 23 – Ícone para cria novo projeto.

A seguir, temos a tela do novo projeto na figura 24, onde será definido: o nome do projeto, o caminho que ele será salvo, a descrição, o tipo de microcontrolador e a freqüência do

oscilador utilizado. Ainda, é possível setar flags de controle do microcontrolador, que controlam o *watchdog* ou o *brownout*, por exemplo. Agora, basta começar programar na tela da figura 25.

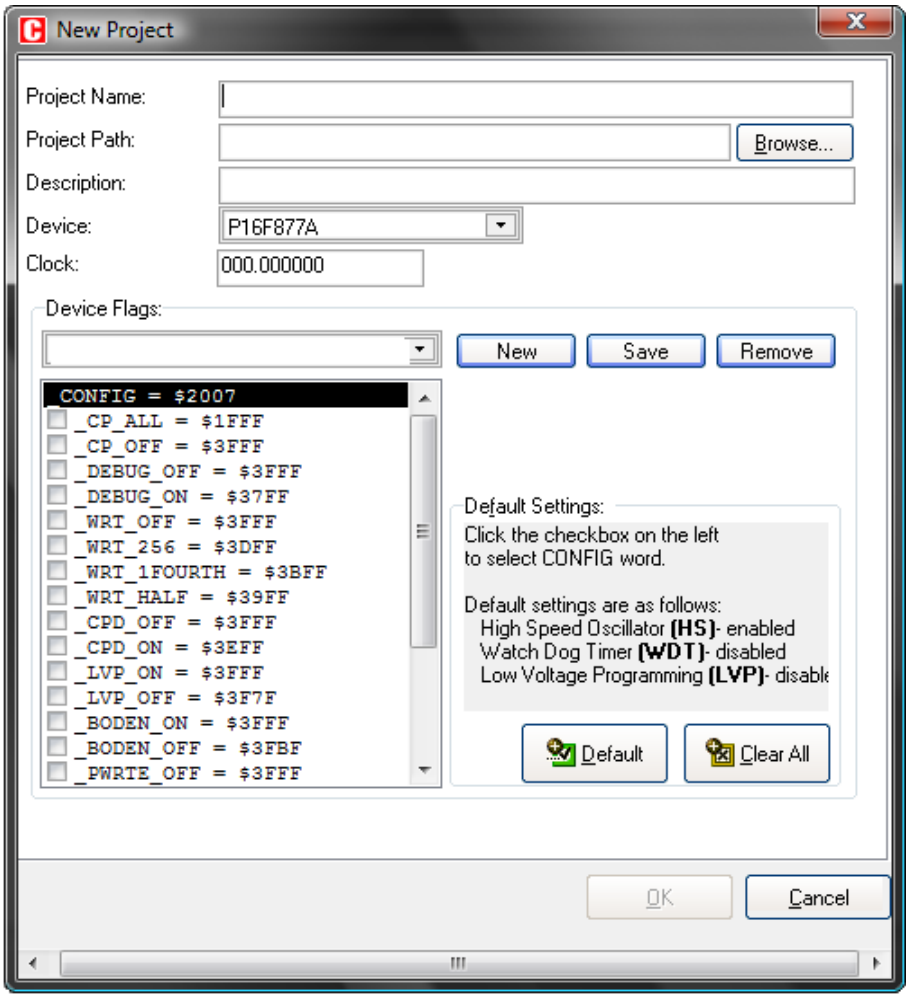
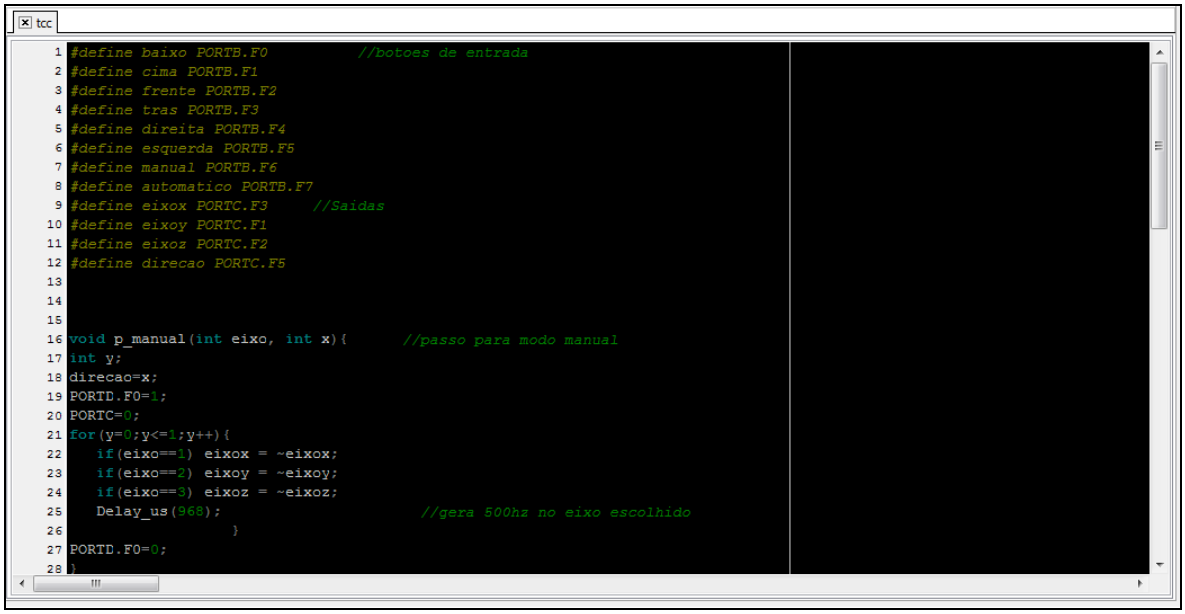


Figura 24 – Tela de configurações para novo projeto.



```
tcc
#define baixo PORTB.F0          //botoes de entrada
#define cima PORTB.F1
#define frente PORTB.F2
#define tras PORTB.F3
#define direita PORTB.F4
#define esquerda PORTB.F5
#define manual PORTB.F6
#define automatico PORTB.F7
#define eixox PORTC.F3      //Saidas
#define eixoy PORTC.F1
#define eixoz PORTC.F2
#define direcao PORTC.F5



void p_manual(int eixo, int x){       //passo para modo manual
int y;
direcao=x;
PORTD.F0=1;
PORTC=0;
for(y=0;y<=1;y++){
   if(eixo==1) eixox = ~eixox;
   if(eixo==2) eixoy = ~eixoy;
   if(eixo==3) eixoz = ~eixoz;
   Delay_us(968);                     //gera 500hz no eixo escolhido
                  }
PORTD.F0=0;
}
```

Figura 25 – Tela para a programação em C.

A.2 Compilando um projeto

Após a programação, devemos compilar o projeto. Para isto, basta clicar no ícone da figura 26, localizado na barra de ferramentas ou pelo atalho Ctrl+F7.

Figura 26 – Ícone para compilar o projeto.

Nota-se, ao lado esquerdo da figura 27, as informações sobre o programa como a rotina principal, sub-rotinas e as variáveis utilizadas no programa. Logo abaixo, temos o microcontrolador e sua freqüência em uso e no canto inferior, a barra de mensagem. Na barra de mensagem, são informadas a quantidade de memória RAM e ROM utilizadas pelo programa, assim como são dadas as mensagem de alerta e erro, caso o programa contenha algum problema.

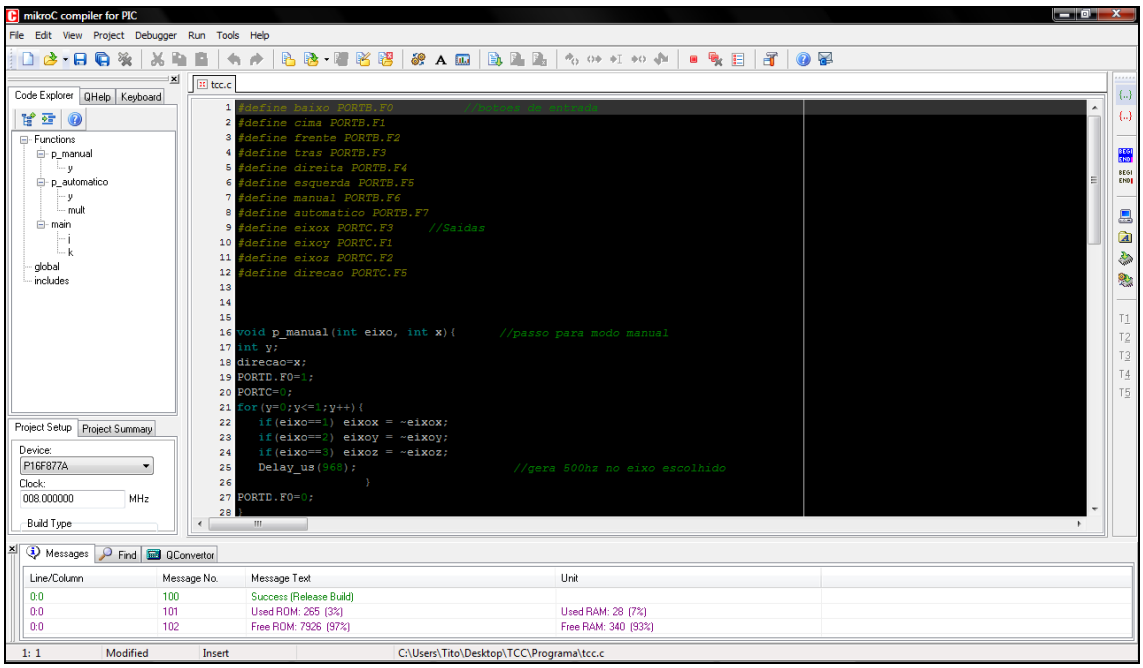


Figura 27 – Tela MikroC após a compilação.

No caso da figura 28, a mensagem avisa que fora esquecido de se utilizar o ponto-e-vírgula no final de uma instrução. Esta ferramenta é importante, pois auxilia o programador a encontrar erros facilmente em programas complexos, diminuindo o tempo total de programação.

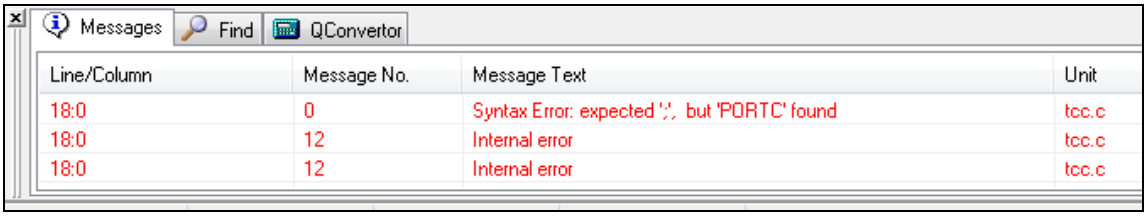


Figura 28 – Caixa de mensagens e alertas.

Clicando sobre o ícone da figura 29, no painel de ferramentas, temos as estatísticas. Elas são importantes para o programador, uma vez que a memória nos microcontroladores é limitada.

Figura 29 – Ícone para visualizar as estatísticas do programa.

Nesta ferramenta, é possível ver quanto espaço cada rotina ocupa na ROM, qual o endereço em que cada uma está alocada e o programa em Assembly, como mostra a figura 30.

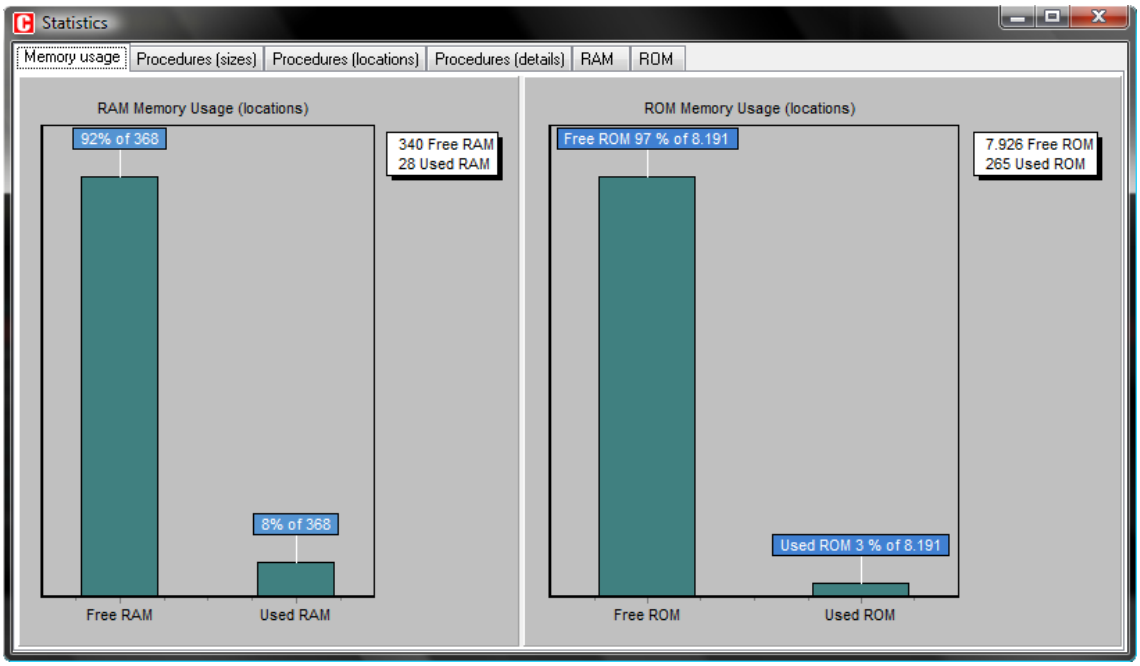


Figura 30 – Tela para a visualização das estatísticas.

A.3 Simulando a execução

Com o projeto devidamente compilado, devemos testar o programa com o simulador. No simulador, pode-se contar número de ciclos, testar o fluxo do programa, verificar o valor de variáveis durante a execução do programa em tempo real, entre outras.

Para entrar do modo de simulação, devemos clicar no ícone da figura 31 e então teremos a tela da figura 32.

Figura 31 – Ícone para iniciar a simulação.

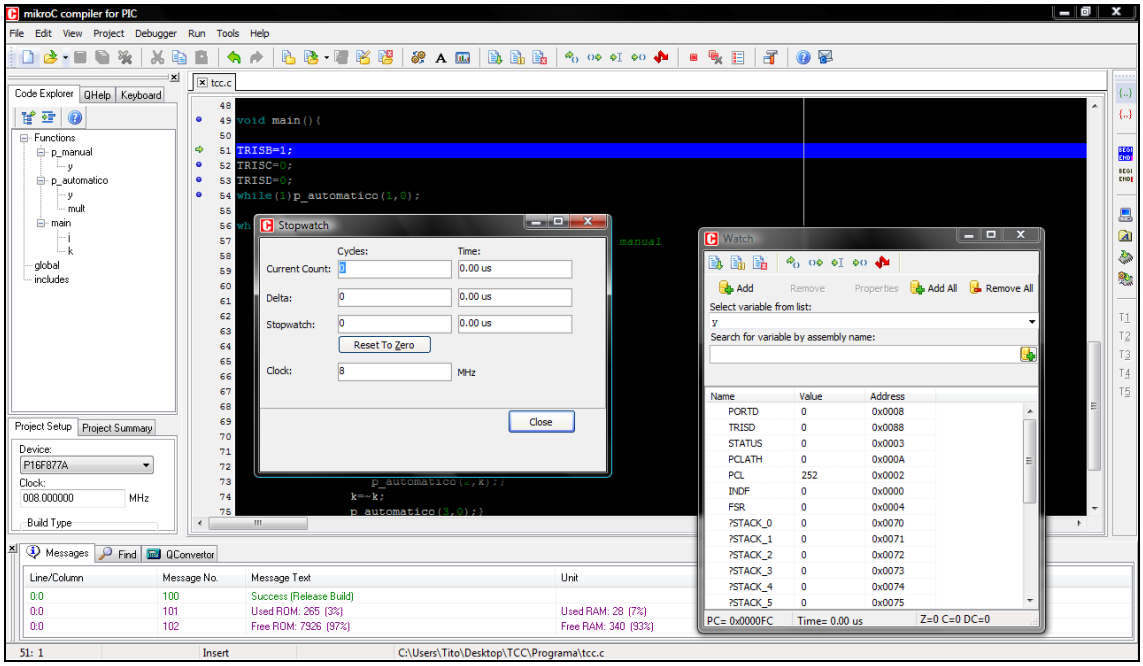


Figura 32 – Tela após entrar no modo de simulação.

Duas telas importantes são abertas. A primeira é o contador de ciclos, que conta quantos ciclos são necessários para realizar determinada tarefa. Também é possível saber quanto tempo leva para realizá-la, uma vez que se tem a freqüência do oscilador.

A segunda janela é a “*Watch*”, onde é possível acompanhar, em tempo real, o valor de variáveis, saídas e entradas, valores das pilhas e registradores, etc. Esta função permite ter um controle total sobre o programa, sabendo exatamente o que está ocorrendo durante a execução do programa.

Para simular, podemos ir passo a passo, definir um *breakpoint* em um determinado local do programa ou ir até onde está posicionado o cursor. Para isto, utilizamos a barra de ferramentas da figura 33.

Figura 33 – Barra de controle da simulação.

Onde cada ícone representa uma função específica durante a simulação:

1. *Step Over* - Executa o programa passo a passo e pára. Caso a instrução seja uma rotina, executa toda a rotina e pára no ponto após a rotina.
2. *Step Out* - Executa o programa passo a passo e pára. Caso a instrução esteja dentro de uma rotina, executa toda a rotina e pára na primeira instrução após a rotina.
3. *Run to cursor* - Executa todas as instruções entre a instrução atual até onde está o cursor.
4. *Step Into* - Executa o programa passo a passo e pára. Caso a instrução seja uma rotina, adentra a rotina e pára na primeira instrução dentro da rotina.

Anexo B

B.1 Programa em C

O programa foi desenvolvido em ambiente MikroC em sua versão demonstrativa 8.1. Os fusíveis de *brownout* e *watchdog* foram desligados e a freqüência do cristal oscilador utilizado no PIC foi de 8 MHZ (Anexo B).

Na primeira parte do programa, são definidas as saídas do PIC. São elas que irão controlar o *driver*, dar o sinal de *flag* para o sensor e controlar os *leds* indicadores. As portas C e D foram usadas para as saídas de controle.

A próxima rotina é a que controla o passo manual. Nesta rotina, será escolhido previamente o eixo a ser movimentado e qual a direção escolhida. Então é ligado o *led* indicador do modo manual e os pulsos são gerados para o eixo escolhido, com *duty- cicle* de 50%. Assim, espera-se 968 ms tanto no nível alto, quanto no nível baixo. O valor da sub-rotina de *delay* foi determinado inicialmente por contagem de ciclos do programa e o ajuste fino da freqüência é feito empiricamente com a ajuda do osciloscópio.

A rotina de passo automático funciona basicamente igual à rotina manual. A diferença é a fórmula que faz com que o multiplicador de *delay* de 1 ms se altere conforme o gráfico da tabela 2, onde foi feito a contagem de ciclos do programa. Quanto maior o multiplicador, maior será o período da onda e, conseqüentemente, menor será a freqüência. O motor só funciona quando a variável “libera” é igual a 1, que é setado somente para o eixo escolhido. O *flag* de sinalização e todos os controles da porta C são zerados no final da rotina.

A rotina de varredura é responsável por levar o sensor para a posição inicial de varredura e realizar todo o processo lógico que varrerá todo o cubo, plano por plano. Ela é responsável também por gerar o *flag* que sinalizará ao sensor para realizar as medidas.

A rotina final é a rotina “*Main*”. Ela é a rotina principal do programa e é responsável pela configuração das portas de entrada e saída. A porta B foi a escolhida para a entrada dos botões. Uma vez pressionado um botão, ele aterrará o pino correspondente e mudará o valor da porta B. O programa fica constantemente inspecionando a porta B para verificar se algum botão foi pressionado. Caso haja alguma alteração, a função “*switch*” lê a porta B e conduz o programa para a função que foi selecionada pelo botão. Caso seja necessário parar a seqüência de varredura, basta resetar o PIC.

```
#define  eixox        PORTC.F0      // Saídas - Porta C.
#define  liberax      PORTC.F1
#define  eixoy        PORTC.F2
#define  liberay      PORTC.F3
#define  eixoz        PORTC.F4
#define  liberaz      PORTC.F5
#define  direcao      PORTC.F6
#define  flag         PORTC.F7

#define  ledon        PORTD.F0      // Saídas - Porta D.
#define  ledmanual    PORTD.F1
#define  ledauto      PORTD.F2



void p_manual(int eixo, int x){          // Rotina para passo para modo manual.
int y;
direcao=x;
ledmanual=1;
   for(y=0;y>=1;y++){                    // Subrotina que gera pulso à 500Hz.
         if(eixo==1){                    // Cria o pulso no eixo desejado.
            liberax=1;
            eixox = ~eixox;}
         if(eixo==2){
            liberay=1;
            eixoy = ~eixoy;}
         if(eixo==3){
            liberaz=1;
            eixoz = ~eixoz;}

         Delay_us(968);                  // Espera de 968 ms.
                     }
ledmanual=0;
PORTC=0;                                 // Zera a Porta C.
                                  }


void p_automatico(int eixo, int x){        // Rotina para passo para modo
   int y,mult=104;                         // automático.
   direcao=x;
   for(y=0;y<160;y++){                     // Gera 80 passo no eixo desejado.
         if(eixo==1){
            liberax=1;
            eixox = ~eixox;}
         if(eixo==2){
            liberay=1;
            eixoy = ~eixoy;}
         if(eixo==3){
            liberaz=1;
            eixoz = ~eixoz;}
```

```
51
52          if(y<=40) mult=mult-2;                // Faz com que a frequência
53          if(y>=140) mult=mult+4;               // seja variável.
54
55          for(x=0;x<mult;x++) delay_us(1);      // Espera variável de acordo com o
56                        }                       // multiplicador.
57    PORTC=0;
58 }
59
60 void varredura(){
61
62    int k, l, m, a=0, b=0;
63    ledauto=1;
64
65    for(k=0;k<20;k++)p_manual(1,1);          // Faz o deslocamento para o
66    for(l=0;l<20;l++)p_manual(2,1);          // ponto inicial da varredura.
67    for(m=0;m<20;m++)p_manual(3,1);
68
69    for(k=0;k<40;k++){                        // 41 passos para o eixoz.
70       for(l=0;l<40;l++){                     // 41 passos para o eixoy.
71          for(m=0;m<40;m++){                  // 41 passos para o eixox.
72             flag=0;
73             p_automatico(1,a);
74             flag=1;}                        // Gera flag para o sensor a cada
75          a=~a;                              // 1mm deslocado pelo eixo x.
76          p_automatico(2,b);}
77       b=~b;
78       p_automatico(3,0);}
79
80    ledauto=0;}
81
82
83 void main(){
84
85 TRISB=1;       // Configura a Porta B como entrada.
86 TRISC=0;       // Configura a Porta C como saída.
87 TRISD=0;       // Configura a Porta D como saída.
88
89 PORTB=1;       // Seta a porta B
90 PORTC=0;       // Zera as portas C e D
91 PORTD=0;
92
93    while(1){
94
95         switch(PORTB){                       // Escolhe o tipo de controle
96                                              // atraves da porta B.
97                case 254: p_manual(1,0);      // Manual para direita.
98                          break;
99
```

```
100                 case 253: p_manual(1,1);       // Manual para esquerda
101                         break;
102
103                 case 251: p_manual(2,0);       // Manual para cima
104                         break;
105
106                 case 247: p_manual(2,1);       // Manual para baixo
107                         break;
108
109                 case 239: p_manual(3,0);       // Manual para frente
110                         break;
111
112                 case 223: p_manual(3,0);       // Manual para trás
113                         break;
114
115                 case 191: varredura();         // Modo automático
116                         break;
117
118                             }
119     }
120 }
```