Centro Universitário de Brasília.
Faculdade de Ciências Exatas e Tecnologia.

PROJETO FINAL DE GRADUAÇÃO DO CURSO DE ENGENHARIA DA COMPUTAÇÃO

ANIMATRÔNICOS

LUCIANE BARBOSA DE ANDRADE

R.A.: 2001607/3

BRASÍLIA, 1º SEMESTRE DE 2005.

Centro Universitário de Brasília.

Faculdade de Ciências Exatas e Tecnologia

PROJETO FINAL DE GRADUAÇÃO DO CURSO DE ENGENHARIA DA COMPUTAÇÃO

ANIMATRÔNICOS

AUTORA: LUCIANE BARBOSA DE ANDRADE
R.A.: 2001607/3

ORIENTADA POR: M.C. MARIA MARONY SOUSA FARIAS NASCIMENTO

BRASÍLIA, 1º SEMESTRE DE 2005.

DEDICATÓRIA

O presente é reflexo das nossas atitudes do passado; **sendo assim, hoje não seria possível essa conquista, se meus pais não tivessem apostado em mim, me dando força e perseverança para suportar os obstáculos que às vezes aparecem durante a caminhada.**

A amizade e o apoio de vocês foi fator decisivo em todas as minhas conquistas. Dedico este projeto a Deus por ter me dado o maravilhoso presente, que é esta grandiosa família.

AGRADECIMENTOS

Agradeço primeiramente a Deus pelo dom da vida e pela tentativa de obter o dom da Perseverança.

Ao meu querido noivo que sempre esteve ao meu lado, me fazendo rir e encontrar soluções onde pareciam não existir mais nada.

Agradeço também, ao Engenheiro e amigo Tiago Mitsuka, pela força e amizade oferecidas durante todo curso e principalmente durante o desenvolvimento deste trabalho.

E por último, mas com igual valor, agradeço a minha orientadora, Marony, por sempre estar disponível e disposta a ajudar.

RESUMO

Basicamente um animatrônico é um boneco mecanizado. Pode ser pré-programado ou remotamente controlado. Ele pode realizar movimentos a partir de uma faixa escolhida. Eles diferem dos robôs por não serem capazes de agir diante de uma determinada situação e por isso realizam ações pré-programadas.

Este trabalho consiste em um estudo sobre processos e conceitos necessários para a implementação da cabeça de um robô (animatrônico), utilizando técnicas de programação em microcontrolador PIC. O animatrônico realizará movimentos de abertura e fechamento de sua boca, sincronizado à função de fala, através do auxílio de um servomotor.

O material usado para fazer a estrutura mecânica é composto de um PVC, uma boca unida por parafusos para a geração dos movimentos.

Palavras-chave: animatrônico, PIC, servomotor.

ABSTRACT

Basically, an animatronic is a mechanized puppet. It may be preprogrammed or remotely controlled. The animatronic may only perform a limited range of movements. They differ from the robots for not being capable to act by situation and thus they carry through preprogrammed actions.

It consists of the necessary study on processes and concepts for the implementation of the head of a robot (animatronic), using techniques of programming in microcontroller PIC. The animatronic will carry through movements of opening and closing of its mouth, synchronized to the function of speaks, through the aid of a servo.

The material used to make mechanics structure is composed by a PVC, the mouth joined for screws to generate movements.

Keywords: animatronic, PIC, servo.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE EQUAÇÕES

CAPÍTULO 1

INTRODUÇÃO

1.1. ESTRUTURA DO TRABALHO

Este trabalho discorre sobre o tema animatrônicos, fazendo um breve histórico sobre sua origem e versando sobre os conceitos básicos que circundam todo o processo de construção e programação de um animatrônico com a funcionalidade da fala.

A partir deste trabalho é possível construir um animatrônico ou até mesmo aprimorar as técnicas utilizadas para a construção de um outro protótipo com mais funcionalidades.

No Capítulo 1 é dado um panorama geral sobre a monografia e sua motivação.

No Capítulo 2 é tratada em detalhes a proposta do animatrônico e quais funcionalidades foram implementadas.

No Capítulo 3 é feito um breve histórico sobre o surgimento do termo animatrônico, e onde é possível encontrá-los.

No Capítulo 4 é abordada a confecção do protótipo, descrevendo todas as etapas do processo de criação para a análise de requisitos, que permitiu a escolha do motor capaz de suportar o peso da articulação móvel.

No Capítulo 5 são tratados os conceitos necessários à manipulação dos timers encontrados no PIC16F628A.

No Capítulo 6 são tratadas as diretivas de áudio utilizadas para a sincronização de voz.

No Capítulo 7 é abordado o processo de montagem do circuito eletrônico, bem como a criação do programa de interface e sincronização entre movimentação do motor e áudio.

No Capítulo 8 são tratados os resultados obtidos, dificuldades encontradas, conclusões e sugestões para trabalhos futuros.

Por fim, os anexos contendo importantes diretivas de trabalho para o compilador utilizado, utilização do RS232 (comunicação serial), datasheets de microcontroladores utilizados, códigos em C para programação no PIC16F628A, dentre outros.

1.2. MOTIVAÇÃO

Os animatrônicos são mais utilizados em parques temáticos, tanto no Brasil como em outras partes do mundo. Facilmente encontrados na Disney, eles chamam atenção dos espectadores que se encantam pelas novidades proporcionadas pela engenharia. Eles recepcionam pessoas, interagem com o público e contam histórias.

Um dos fatores mais importantes para a escolha do tema foi a multidisciplinaridade proporcionada pelo processo de confecção de um animatrônico, pois é possível relacionar temas ligados às áreas de: mecânica, eletrônica, transmissão de sinais, modulação, técnicas de programação; todas elas intrínsecas à engenharia. Abrangendo, assim, uma gama de disciplinas que garantam o conhecimento de conceitos relacionados com várias áreas importantes para a engenharia.

Além da tentativa de mesclar o máximo de conhecimentos adquiridos, também há o fato de que a eletrônica aliada a tantas outras disciplinas é algo interessante e prazeroso de estudar.

CAPÍTULO 2

OBJETIVO DO ESTUDO.

2.1. PROPOSTA DO ANIMATRÔNICO

Este projeto tem como objetivo a construção de um animatrônico com a funcionalidade da fala, e um estudo sobre os processos e etapas realizadas para a finalização desta construção.

A pesquisa dos materiais que compõem a estrutura do animatrônico se deu pela necessidade da busca por materiais simples, de fácil acesso e baixo custo. Sendo assim, na estrutura de sustentação foram utilizados como material o PVC.

O animatrônico desenvolvido possui o movimento de abertura e fechamento da “boca”, sincronizado à função de fala, através do auxílio de um motor.

O protótipo da cabeça foi desenvolvido utilizando o PVC como material, onde foram cortadas peças para simular uma articulação da boca. Somente a parte inferior será movimentada, proporcionando a abertura e o fechamento, simulando um processo de fala.

A articulação é ligada através de um fio a uma das hélices de um servomotor. Este recebe informações do microcontrolador, gerenciado por um programa que faz a interface entre o animatrônico e usuário, utilizando a comunicação serial.

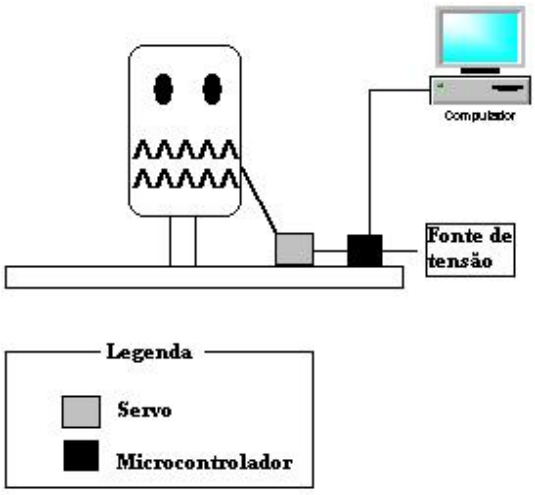

Figura 2-1 - Esquema básico da validação do projeto

Na Figura 2-1 é apresentada o esquema básico do protótipo desenvolvido.

Alinhado à construção do animatrônico, foi realizado um estudo sobre o microcontrolador PIC16F628A e um levantamento de suas particularidades mais importantes referentes ao acionamento de motores e função PWM.

CAPÍTULO 3

ANIMATRÔNICA

A “animatrônica” é a arte de utilizar à eletrônica e a mecânica para animar bonecos, sendo muito usada no cinema e até mesmo na vida real. **[BERNARDES - 2003]**

Ao final do ano podemos perceber perfeitamente a grande utilização destes bonecos em lojas e *shoppings centers*. Sua principal funcionalidade é o entretenimento ou recepção de pessoas, como em um balcão de informações, por exemplo, onde o freqüentador de um *shopping* pode solicitar ao animatrônico, através de um computador, informações de localização; mas o uso desses bonecos animados pode ser estendido para outras áreas (educação é uma delas).

A palavra animatrônica é utilizada para diferenciar-se dos robôs, que possuem autonomia e são capazes de tomar diferentes decisões conforme a situação. Os animatrônicos em sua maioria são programados para realizarem tarefas dedicadas e algumas vezes até movimentos repetitivos e não reagem de acordo com o ambiente, são programados para realizem tarefas definidas.

Este termo foi utilizado pela primeira vez pelos engenheiros da “Walt Disney Imagineering Company” no início de 1960, pela necessidade do departamento de efeitos visuais providenciar algo mais interessante para o público, visto que o mesmo estava tornando-se cada vez mais crítico e observador. **[BERNARDES - 2003]**

Atualmente, podemos ver os animatrônicos em museus, parques temáticos e principalmente no cinema.

Figura 3-1 - Boneco produzido pela ROBOARTE [ROBOARTE - 2005]

A empresa, ROBOARTE, sediada em São Paulo, foi fundada pelo Engenheiro Eletrônico Renato Kleiner em 1993 e produz mais de 50 animatrônicos por ano.

A empresa trabalha de duas formas: uma delas é a confecção total dos bonecos, desde a parte visual até os mecanismos de funcionamento. A outra forma é a inclusão apenas da parte mecatrônica em bonecos já prontos, cuja forma é onde se gera mais demanda de serviço. A principal demanda da empresa acontece na época do Natal. **[BERNARDES - 2003]**

Neste trabalho é possível verificar todas as etapas de criação de um animatrônico com a funcionalidade da fala. Ele poderia ser utilizado para a recepção de pessoas, ou para auxiliar na educação, estimulando o aprendizado.

CAPÍTULO 4

ESTRUTURAÇÃO MECÂNICA

Este capítulo contém informações básicas para o início da estruturação do animatrônico tais como: materiais utilizados, recursos de hardware e software necessários.

4.1. MATERIAIS UTILIZADOS

- TUBO DE PVC
    - Um pedaço de 31 cm de comprimento de um tubo de PVC de 75mm de diâmetro;
    - Um pedaço de 23 cm de comprimento de um tubo de PVC de 100 mm de diâmetro;
    - Uma luva de redução (PVC) de 100 mm para 75 mm de diâmetro;
    - Um tampão de PVC de 100 mm de diâmetro.
- CINCO PREGOS;
- MADEIRA
    - Um pedaço de madeira (Amapá) de 45 cm de largura por 30,5 cm de comprimento;
    - Um pedaço de madeira (Amapá) cortada com o “serra-copo” e com um furo no meio feito pela furadeira para encaixe de um parafuso de 0,46mm de diâmetro.
- FURADEIRA
    - Broca para parafuso de 0,46 mm de diâmetro;
    - Broca para passagem do fio que se ligará a uma das hélices do motor;
    - Serra-copo.
- UMA SERRA TICO-TICO;
- SETE ARRUELAS;

- SETE PORCAS;
- TRÊS PARAFUSOS DE 0,46 MM DE DIÂMETRO;
- UM PARAFUSO DE 0,46 MM DE DIÂMETRO COM UM COMPRIMENTO MAIOR;
- TINTA BRANCA.

4.1.1. Montagem do protótipo (parte mecânica).

Antes do estudo da estrutura eletrônica, é preciso dar início à pré-montagem do protótipo, uma vez que se faz necessário dimensionar as peças e escolher o motor. Esse motor está diretamente relacionado com a força necessária para elevar ou abaixar a parte inferior da boca.

No Anexo A – Montagem do Protótipo é mostrado passo a passo como foi montado o esqueleto do animatrônico.

4.2. SOFTWARES UTILIZADOS

4.2.1. Mplab (Versão 6.61)

O MPlab é um programa para PC que acompanha o kit para gravação do microcontrolador PIC (disponível em <http://www.mosaico-eng.com.br>).

Ele funciona sobre a plataforma Windows, e serve como ambiente de desenvolvimento de programas para PICs. É uma ferramenta muito poderosa e uma das principais responsáveis pela popularização do PIC, pois reúne, no mesmo ambiente, o gerenciamento de projetos, a compilação, a simulação, a emulação e a gravação do chip. Na maioria dos sistemas utilizados por outros microcontroladores, essas funções são executadas por programas separados, tornando o trabalho muito mais cansativo e demorado.

4.2.2. Compilador CCS C (Versão 3.46)

Este compilador consiste em um ambiente integrado de desenvolvimento para sistema operacional Windows e suporta quase toda linha de microcontroladores PIC (séries

PIC12, PIC14, PIC16 e PIC18). Os PICs da série 17, não são suportados por esta versão do compilador.

A versão demo utilizada pode ser encontrada no site do CCS (disponível em <http://www.ccsinfo.com/demo.shtml>).

4.2.2.1. Principais características do Compilador

- Compatibilidade com a padronização ANSI e ISO (algumas características do compilador não fazem parte da normatização ANSI devido ao fato de serem específicas para a arquitetura PIC);
- Grande eficiência no código gerado;
- Grande diversidade de funções e bibliotecas da linguagem C (padrão ANSI), tais como: entrada/saída serial, manipulação de strings e caracteres, funções matemáticas C, etc.:
- Veja no Anexo D as Diretivas do Compilador CCS.

4.2.3. Proteus (Versão 6.2)

Este simulador consiste em um ambiente integrado de desenvolvimento para sistema operacional Windows e suporta os microcontroladores mais populares, séries PIC e 8051 (disponível em: <http://www.labcenter.co.uk/>).

O simulador tem como função auxiliar no processo de criação de um circuito eletrônico, permitindo que o usuário gaste mais tempo na análise de requisitos do sistema do que propriamente na execução do processo de criação do circuito, ajudando a definir melhor o escopo e traçar melhores rotas de execução.

Permite realizar testes interativos com simulação de medição em instrumentos como: Amperímetro, Voltímetro e Osciloscópio. Além disso, possibilita a interatividade com periféricos como: LEDs, Displays LCDs, teclados, RS232 dentre muitos outros. Permite também que seja incluído um programa em C no microcontrolador para efetivação dos testes.

CAPÍTULO 5

ESTRUTURAÇÃO ELETRÔNICA

Antes de seguir para a montagem do circuito, se faz necessário o aprendizado de alguns conceitos importantes sobre a comunicação entre o microcontrolador e o motor.

O estudo da estruturação eletrônica tem início com o conhecimento de interrupções no PIC16F628A.

Neste capítulo, é possível conhecer algumas das interrupções do PIC e suas funcionalidades.

5.1. MICROCONTROLADOR

O microcontrolador escolhido para execução deste trabalho é da família PIC.

PIC, *Peripherals Integrated Controller*, é o nome que a Microchip adotou para a sua família de microcontroladores, sendo que a sigla significa, em português, *Controle Integrado de Periféricos.*

Atualmente todos os microcontroladores são muito parecidos quanto as suas funcionalidades, a grande diferença está tanto no aspecto de hardware quanto do software.

Talvez a maior vantagem do PIC sobre os demais é exatamente a sua popularização, pois hoje é muito fácil achar material sobre ele, como: livros, apostilas, até mesmos fóruns na Internet onde é possível trocar informações.

O PIC utilizado neste projeto foi o PIC16F628A. Acredita-se que a letra ao final de seu nome não faz diferença alguma, porém essas letras (A,B,C,...) representam novas versões para o mesmo modelo. Quanto mais nova a versão, mais barato é o PIC, mas podem haver alterações em certas características elétricas.

5.1.1. Watch Dog Timer

O watchdog é um recurso disponível no PIC que parte do princípio que todo sistema é passível de falha. Se todo sistema pode falhar, cabe ao mesmo ter recursos para que, em ocorrendo uma falha, algo seja feito de modo a tornar o sistema novamente operacional.

Dentro do PIC existe um contador incrementado por um sinal oscilador (RC) independente. Toda vez que este contador extrapola o seu valor máximo retornando a zero, é provocado a reinicialização do sistema (reset).

5.1.1.1. Clear Watchdog

Se o sistema estiver funcionado da maneira correta, de tempos em tempos uma instrução denominada clear watchdog timer (CLRWDT) zera o valor deste contador, impedindo que o mesmo chegue ao valor máximo. Desta maneira o Watchdog somente irá "estourar" quando algo de errado ocorrer (travando o sistema).

5.1.1.2. Pre-Scaler

O período normal de estouro do Watchdog Timer é de aproximadamente 18 ms. No entanto, algumas vezes este tempo é insuficiente para que o programa seja normalmente executado. A saída neste caso é alocar o recurso de prescaler de modo a aumentar este período. Se sem o prescaler o período é de 18ms, quando atribuímos ao Watchdog Timer um prescaler de 1:2 (um para dois) nós dobramos este período de modo que o processamento possa ser executado sem que seja feita uma reinicialização.

5.1.2. Ligando as chaves corretas

O conhecimento prévio de operações relativas ao uso das interrupções e chaves de habilitação será muito útil nas seções abaixo.

Existem três tipos de chaves de habilitação:

- **Individuais:** São as chaves que habilitam e desabilitam cada uma das interrupções individualmente (geralmente denominadas Máscaras).

- **Grupo:** No PIC 16F628A, e em muitos outros modelos, existe um tipo especial de chave que controla a habilitação de todo um grupo de interrupções, denominadas interrupções de periféricos.
- **Geral:** Desabilita todas as interrupções simultaneamente ou habilitam todas aquelas que estão com suas chaves individuais ligadas.

Portanto, para utilizarmos uma determinada interrupção, devemos primeiramente ligar sua chave individual, ligando depois a chave de grupo (se existir) e por último a chave geral. **[SOUZA - 2005]**

5.1.3. Timers Internos

Para comunicação com o motor se faz necessário o conhecimento sobre Timers. Neste tópico, encontram-se informações sobre geração de interrupções em intervalos fixos de tempo.

5.1.3.1. Timer 0

O TMR0 é um contador de 8 bits que pode ser acessado diretamente na memória, tanto para a leitura quanto para a escrita. A diferença entre ele e os demais registradores é que seu incremento é automático e pode ser feito pelo clock da máquina ou por um sinal externo. É importante salientar que o estouro desse contador pode gerar uma interrupção.

5.1.3.2. Timer 1

O Timer 1 é outro contador automático do sistema, mas com uma enorme vantagem sobre o Timer 0: trata-se de um contador de 16 bits, que também pode ser acessado diretamente na memória, tanto para leitura quanto para a escrita. Na entanto, devido ao seu tamanho, esse registrador é armazenado em dois endereços: TMRIH (parte alta) e TMR1L (parte baixa). Além disso, o registrador T1CON é o responsável pelas diversas configurações relacionadas ao Timer1, tais como: habilitação, prescaler, oscilador externo próprio, origem do incremento (interno ou externo) e sincronismo de incremento.

Outra grande vantagem do TMR1 é o fato de que ele pode trabalhar com um prescaler independente, configurado diretamente em T1CON <T1CKPS1:T1CKPS0>.

De forma semelhante ao TMR0, o TMR1 também pode ser incrementado pelo clock de máquina ou por um sinal externo, ligado ao pino T1CKI (borda de subida). Quem configura o tipo de incremento é o bit TMR1CS do registrador T1CON. Só que, quando configurado para trabalhar com incremento por sinal externo, os pinos T1OSO e T1OSI possuem também um circuito de oscilação, para poder utilizar diretamente um Ressoador/ Cristal externo só para o Timer 1. Com isso pode-se fazer um timer com clock diferente da máquina. Além disso, através do bit T1CON <T1OSCEN> esse oscilador pode ser habilitado ou não, fazendo com o TMR1 seja paralisado e economizando energia para o sistema. **[SOUZA – 2005]**

Vale comentar também que existe uma interrupção específica relacionada ao estouro do Timer 1. Para utilizar essa interrupção, as seguintes chaves devem ser ligadas.

- PIE1 <TMR1IE>: chave individual da interrupção de TMR1.
- INTCON <PIE>: chave do grupo de interrupções de periféricos.
- INTCON <GIE>: chave geral de interrupções.

Quando a interrupção acontecer, será configurado o bit PIR1 <TMR1IF>. Após o tratamento, este bit não será limpo automaticamente, ficando esta tarefa para o programador.

5.1.3.3. Timer 2

O TMR2, por sua vez, volta a ser de 8 bits e só pode ser incrementado internamente pelo clock da máquina. Por outro lado, ele possui duas vantagens sobre os demais.

A primeira delas é o fato de possuir um prescaler próprio e também um postscaler. A diferença é que o prescaler é utilizado para incrementar o timer (TMR2) propriamente dito, enquanto o postscaler conta a quantidade de estouros desse timer para poder gerar uma interrupção. Por exemplo, caso o prescaler seja de 1:4 e o postscaler seja de 1:14, o sistema funcionará da seguinte maneira: a cada quatro ciclos de máquina o TMR2 será incrementado. Depois que esse timer estourar 14 vezes, uma interrupção será gerada.

Em segundo lugar, existe um segundo registrador (PR2) utilizado para controlar o estouro de TMR2. No caso do Timer 0, que também é um timer de 8 bits, o estouro acontece sempre que o valor é incrementado de 0xFF para 0x00, pois o limite máximo de bits foi atingido. No Timer 2 esse limite não precisa ser atingido, pois define-se o número máximo permitido para TMR2 através de PR2. A comparação entre esses dois registradores é feita sempre, e de forma automática, pelo sistema. Caso TMR2 = PR2, no próximo incremento será considerado um estouro e TMR2 voltará à zero. Com isso não é mais necessário inicializar o valor do contador para que a quantidade de incrementos seja a desejada. Basta especificar esse valor em PR2.

Para utilizar a interrupção desse timer, as seguintes chaves devem ser ligadas:

- PIE1 <TMR2IE>: chave individual da interrupção de TMR2.
- INTCON <PIE>: chave do grupo de interrupções de periféricos.
- INTCON <GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será configurado o bit PIR1<TMR2IF>. Após o tratamento, este bit não será limpo automaticamente, ficando esta tarefa para o programador

5.1.4. Configurando os Timers

Nesta seção é possível encontrar os procedimentos necessários para a configuração dos Timers contidos no PIC16F628A.

5.1.4.1. Timer 0

Configuração para trabalhar com o timer 0 para fornecer uma base de tempo de um segundo.

Para saber quantas interrupções o TMR0 irá gerar a cada segundo, será utilizado a fórmula abaixo:

Número de interrupções / segundo = F_{OSC} / 4 / Prescaler / TMR0.

Equação 5.1.4-1 - Interrupções no Timer 0

Em que:

- F_{OSC} - é a freqüência do oscilador.

Considerando que o Prescaler esteja configurado para dividir por 256, que o TMR0 seja utilizado diretamente e a freqüência de operação do PIC seja de 4Mhz, teremos:

Número de interrupções / segundo = 4Mhz / 4 / 256 / 256.

O que resulta em aproximadamente 15,26 interrupções por segundo (15,26Hz).

5.1.4.2. Timer 1

Configuração para trabalhar com o timer 1 para fornecer uma base de tempo de um centésimo de segundo (ou 10 milissegundos).

Considerando que a freqüência de operação do PIC é de 4Mhz, temos que:

$$F_{INT} = CLOCK / PRESCALER / (65536 - TMR1)$$

Equação 5.1.4-2 - Interrupções no Timer 1

Em que:

- F_{INT} – é a freqüência de interrupções do timer1;
- CLOCK – é a freqüência de clock do timer 1 (no caso $F_{OSC} / 4$);
- PRESCALER – é o fator de divisão do prescaler do timer 1;
- TMR1 – é o valor inicial do registrador TMR1.

Desta forma, considerando que o prescaler do timer 1 seja configurado para fator de divisão 1, teremos que:

100 = 1.000.000,00 / 1 / (65536 – TMR1) OU,

TMR1 = 55536

Isto significa que o timer 1 deve iniciar sua contagem em 55536 e a cada interrupção (TMR1IF) este valor novamente ser carregado no registrador TMR1.

5.1.4.3. Timer 2

Configuração para trabalhar com o timer 2 para fornecer uma base de tempo de um milésimo de segundo (ou 1 milissegundos).

Considerando que a freqüência de operação do PIC é de 4Mhz, teremos que:

$$F_{INT} = CLOCK / PRESCALER / (256 - PR2) / POSTSCALER$$

Equação 5.1.4-3 - Interrupções no Timer 2

Em que:

- F_{INT} – é a freqüência de interrupções do timer2;
- CLOCK – é a freqüência de clock do timer 2 (no caso $F_{OSC} / 4$);
- PRESCALER – é o fator de divisão do prescaler do timer 2;
- PR2 – é o valor armazenado no registrador de período do timer2;
- POSTSCALER – é o fator de divisão do postscaler do timer2.

Desta forma, considerando que o prescaler esteja configurado para divisão por 4 e o postscaler para divisão por 2, teremos que:

1000 = 1.000.000,00 / 4 / (256 - PR2) / 2 ou,

PR2 = 131.

Isto significa que o timer 2 deve iniciar sua contagem em 131 e a cada interrupção (TMR2IF) este valor deve novamente ser carregado no registrador TMR2, o que acontece automaticamente graças ao registrador PR2.

5.1.5. Módulo CCP

Dentro do PIC16F628A, é possível encontrar um módulo que relaciona três tipos de recursos: ***C**apture*, ***C**ompare e **P**WM*. Cada um destes recursos é empregado para uma finalidade diferente, isto também significa que só podemos utilizar um dos sistemas de cada vez. Através do registrador CCP1CON poderemos configurar qual recurso será habilitado, alterando-se os valores dos bits CCP1M3:CCP1M0.

5.1.5.1. Modo Capture

O modo capture serve para contar o tempo, através do TMR1, entre pulsos externos através do pino CCP1 (RB3). Com isso podemos implementar facilmente um timer para períodos de ondas.

A primeira coisa a fazer é escolher de que forma o pulso será considerado em CCP1. Existem quatro opções disponíveis através do registrador CCP1CON:

- Contagem em cada borda de descida;
- Contagem em cada borda de subida;
- Contagem a cada 4 bordas de subida;
- Contagem a cada 16 bordas de subida.

Depois disso, toda vez que o evento correto acontecer no pino CCP1, os valores de TMR1L e TMR1H (Timer 1) serão transferidos para os registradores CCPRL e CCPRIH, respectivamente. Basta então configurar o TMR1 corretamente, conforme as necessidades da forma de onda. Observe que, como esse recurso utiliza diretamente o Timer 1, seu uso em outras tarefas pode ficar comprometido.

Para utilizar a interrupção que acontece junto com esse evento, as seguintes chaves devem ser ligadas:

- PIE1 <CCP1IE> : chave individual da interrupção de CCP1.
- INTCON <PIE> : chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será configurado o bit PIR1 <CCP1IF>. Após o tratamento, esse bit não será limpo automaticamente, ficando esta tarefa para o programador.

5.1.5.2. Modo Compare

Nesse modo também utilizamos o TMR1 como base de tempo. No entanto, aqui, o valor do contador é constantemente comparado ao valor escrito nos registradores CCP1L e CCP1H. Quando uma igualdade acontecer, uma das seguintes ações irá acontecer:

O pino CCP1 (se configurado como saída) será colocado em nível lógico alto (1);

O pino CCP1 (se configurado como saída) será colocado em nível lógico baixo(0);

O estado do pino CCP1 não será afetado, mas uma interrupção acontecerá.

Essas ações são configuradas ao setar o modo Compare dentro do registrador CCP1CON.

Para utilizar a interrupção descrita na última ação, as seguintes chaves devem ser ligadas:

- PIE1 <CCP1IE> : chave individual da interrupção de CCP1.
- INTCON <PIE> : chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1 <CCP1IF>. Após o tratamento, esse bit não será limpo automaticamente, ficando esta tarefa para o programador.

5.1.5.3. Modo PWM

Provavelmente o recurso PWM é o mais poderoso dos módulos CCPs, pois com ele podemos obter uma tensão analógica a partir de um sinal digital. Na verdade, esta saída é meramente digital, isto é, somente pode assumir os estados 0 e 1. Porém, pelo conceito aplicado ao PWM, podemos transformá-la em uma tensão variável.

O nome PWM tem sua origem no inglês *Pulse Width Modulation* que em Português pode ser traduzida para Modulação por Largura de Pulso. Trata-se de uma onda com freqüência constante (período fixo) e largura de pulso (duty cycle) variável.

Na figura 5-1 temos a representação de duas formas de onda tipo PWM, cada uma delas com uma largura de pulso diferente:

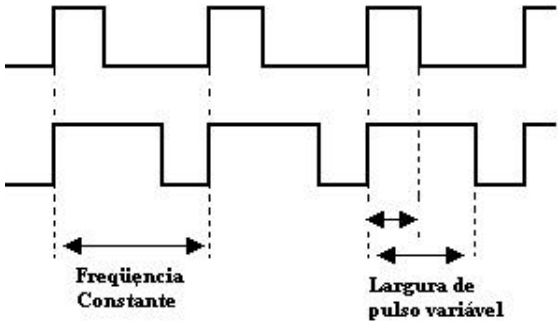

Figura 5-1 - Largura de Pulso (PWM)

Esse tipo de sinal é particularmente importante, já que a partir dele é possível implementar um conversor digital analógico com um único pino do microcontrolador, uma vez que controlando a largura do pulso é possível obter uma tensão analógica variável.

Genericamente, a tensão média de uma forma de onda é calculada pela fórmula abaixo:

$$V_{dc} = \frac{1}{T}\int_0^T V(t)dt$$

Equação 5.1.5-1 - Tensão média da onda

Onde:

- T é o período da forma de onda e V(t) é a função da tensão no tempo.

Para o caso do PWM temos que:

$$V(t) = \begin{cases} V_{pulso} & \rightarrow 0 \le t \le t_p \\ 0 & \rightarrow t_p < t \le T \end{cases}$$

Onde: t_p *é a duração do pulso em nível lógico* 1
V_{pulso} *é a tensão de pulso do sinal PWM*.

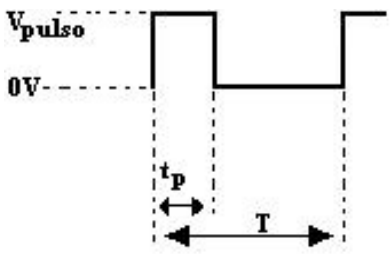

Figura 5-2 - Relação Tensão x período no PWM

Então,

$$V_{dc} = \frac{1}{T}\left(\int_{0}^{t_p} V_{pulso}\, dt + \int_{t_p}^{T} 0 dt \right)$$

$$V_{dc} = \frac{t_p}{T} V_{pulso}$$

Equação 5.1.5-2 - Relação de Tensão x Período PWM

O duty cycle é a razão entre a largura de pulso e o período da forma de onda. O pulso da onda PWM apresenta tensão fixa, porém o valor médio da tensão desta forma de onda varia em função do *duty cycle.* A tensão média (V_{dc}) é diretamente proporcional ao *duty cycle* e como este varia entre 0 (quando $t_p = 0$) e 1 (quando $t_p = T$) temos que a tensão média de onda pode variar entre 0 e V_{pulso}. No nosso caso a variação será de V_{SS} *a* V_{DD}, ou seja, de 0 a 5V. **[SOUZA - 2004]**

5.1.6. Base de tempo no PWM

A base de tempo para o PWM será o Timer2.

O valor do período do PWM está vinculado a um estouro do TMR2, isto é, sempre que TMR2 = PR2. Portanto, para definirmos o tempo desse período, podemos utilizar a seguinte fórmula:

$$PWM_{PERÍODO} = (PR2 + 1) * 4 * T_{OSC} * TMR2_{PRESCALER}$$

Equação 5.1.6-1 - Base de tempo para PWM no Timer 2

Por exemplo, para um sistema rodando a 4MHZ poderíamos ter a seguinte configuração:

$$PWM_{PERÍODO} = (249 + 1) * 4 * 250ns * 4$$
$$PWM_{PERÍODO} = 1ms$$

5.1.7. Conclusões sobre o uso de PWM

Uma das vantagens do PWM é que o sinal continua digital por todo o caminho entre o processador e o controlador; não se tornando necessária uma conversão digital-analógica. Para manter o sinal digital, o ruído é minimizado. O barulho pode afetar somente um sinal digital se for forte o suficiente para mudar o estado de 1 para 0, ou vice-versa.

Incrementar a imunidade do ruído é ainda um outro benefício da escolha do PWM através do controle analógico e é a principal razão que o PWM é às vezes usado para a comunicação. Mudando de um sinal analógico para o PWM pode aumentar o tamanho do canal de comunicação drasticamente. Ao final da recepção, um apropriado circuito RC (resistor-capacitor) ou um LC (indutor-capacitor) pode retirar uma onda quadrada de alta freqüência modulada e retornar o sinal para analógico.

5.2. SERVOMOTOR

Os servomotores utilizam um pequeno motor DC acoplado através de um sistema de engrenagens a uma carga, e um dispositivo de realimentação constituído por um potenciômetro acoplado mecanicamente ao eixo do motor DC e eletricamente ao circuito eletrônico. A variação da resistência do potenciômetro em função da rotação informa ao circuito eletrônico a rotação e direção do motor.

Ao movimentar-se em determinada direção o motor DC ativa uma série de engrenagens que amplificam e transferem o torque do motor ao eixo externo onde são conectados os controles mecânicos.

Figura 5-3 - Engrenagens do Servomotor

Quando uma alimentação e um pulso são aplicados a um servomotor, o mesmo começa a rotacionar em determinada direção até chegar à velocidade de operação, transferindo ao eixo um torque baseado no tamanho do motor e na alimentação fornecida.

Figura 5-4 - Servomotor CS-60 (Hobbico)

Os servomotores são bastante atrativos porque oferecem vários modelos com especificações diversas de velocidade angular e torque; e também pelo fato de serem relativamente baratos. O preço de um servo pode variar entre R$ 30,00 até R$ 400,00.

O modelo escolhido é o CS-60 que opera a 0,19 segundos a cada 60°, o que significa dizer que a cada segundo o motor rotaciona 315,79°, com uma tensão de 4,8 volt produzindo um torque de 3kg.cm.

Para um torque pequeno, como o necessário para a movimentação da articulação do protótipo, não foi preciso um motor muito complexo. Sendo assim, foi escolhido o modelo CS-60, utilizado por *hobbystas,* pelo baixo torque suportado, baixo custo, simplicidade de manuseio, porém trabalha com uma baixa velocidade.

Os servos têm uma interface elétrica muito simples, usualmente tem três fios, um para alimentação positiva, um para terra e o outro para o pulso. A alimentação deve estar entre 4,8 volts até no máximo 6 volts. O fio para pulso está preparado para receber um sinal de pulso na forma PWM.

O *duty cycle* é de 20ms, e a largura de pulso desses 20ms varia entre 1ms até 2ms. É essa a variação que controla o servo.

A Tabela 5-1 mostra a especificação do modelo escolhido encontrado na embalagem do produto.

	Mínimo	Máximo
Alimentação	4.8 Volt	6.0 Volt
Torque	3,06 kg-cm	3,57 kg-cm
Dimensões	41mm x 20mm x 36mm	
Peso	44,9g	
Velocidade (60°)	0,19 s	0,16 s

Tabela 5-1 - Especificações do servomotor, modelo CS-60

CAPÍTULO 6

ÁUDIO NO ANIMATRÔNICO

Foi criado um software para servir de interface entre usuário e microcontrolador, capaz de simular comandos do teclado no terminal e ao mesmo tempo gerenciar arquivos de áudio, enviando pulsos ao microcontrolador.

A interface tem como principais características:

- Configurar uma porta serial;
- Abrir uma porta serial;
- Enviar dados para a porta;
- Abrir um arquivo de áudio;
- Gerenciar o sincronismo entre áudio e pulsos ao microcontrolador;
- (Opção de manutenção) Abrir terminal para verificar as saídas do programa; que auxilia a encontrar erros de programação no PIC;
- (Opção de manutenção) Botões para testar os ângulos de movimentação do servomotor.

6.1. EXECUÇÃO DO ÁUDIO

Para execução do áudio no animatrônico foi necessário fazer o uso da placa de som do microcomputador para a emissão de voz. Para gravação do mesmo, foi utilizado o software cool edit e um microfone acoplado à placa de som.

6.1.1. Interface

A interface foi desenvolvida na linguagem Java pela grande variedade de funcionalidades existentes na ferramenta para execução deste propósito.

Java é uma linguagem computacional completa, adequada para o desenvolvimento de aplicações baseadas em desktop (stand-alone) ou ainda na rede Internet e redes fechadas.

É uma linguagem de alto nível, com sintaxe quase similar à do C, e algumas características herdadas de outras linguagens. Classificada como uma linguagem simples por ser mais próxima da linguagem humana, é ainda fortemente tipada, independente de arquitetura, robusta, segura, extensível, bem estruturada, distribuída, vários processos e com garbage collection.

A memória alocada dinamicamente é gerenciada pela própria linguagem, que usa algoritmos de garbage collection para desalocar regiões de memória que não estão mais em uso. Este recurso é bastante utilizado, uma vez que se torna necessário, por exemplo, excluir os objetos instanciados da conexão à porta serial que naquele instante não estão sendo utilizados.

Ao contrário do C++, que é uma linguagem híbrida, Java é uma linguagem orientada a objetos que segue a linha purista iniciada por Smalltalk. Com a exceção dos tipos básicos da linguagem (int, float, etc.), a maior parte dos elementos de um programa Java são objetos. O código é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si. Cada classe da interface realiza uma determinada função uma vez que possa, de certa forma, modularizar cada etapa do animatrônico: uma classe estrutura a interface do aplicativo, a outra responsável pela fala e uma movimenta o servomotor.

A linguagem Java tem o suporte a multitarefa embutido na linguagem; um programa Java pode possuir mais de uma linha de execução ou processo (thread). Por exemplo, a parte de interface com o usuário, que depende mais dos periféricos de I/O que do processador, pode ser executada em thread separada de outros processos.

Na classe que executa arquivos de áudio (é possível saber, através de um método, quando uma música termina), possibilita parar o envio de pulsos ao microcontrolador. Ainda assim, entre o intervalo de um pulso e outro, a thread de envio dos pulsos é pausada por um determinado período configurável para possibilitar que o servo se movimente normalmente sem que tenha terminado seu ciclo.

Com o Java, é possível tratar erros, sabendo o porquê de cada problema encontrado, possibilitando assim encontrar saídas para esses erros. Isso se torna importante uma vez se é possível saber, por exemplo, se uma determinada porta está aberta ou fechada, se os dados foram enviados à porta serial ou não. A máquina virtual Java faz uma verificação em tempo de execução quanto aos acessos de memória, abertura de arquivos e uma série de eventos que podem gerar uma "travada" em outras linguagens, mas que geram exceções em programas Java.

Diferentemente de outras linguagens de programação em que o programador tinha que preocupar-se em gerenciar corretamente a memória, liberando-a quando um determinado bloco não estivesse sendo mais utilizado, no Java, ao contrário, utiliza-se de um conceito já explorado por Smalltalk, que é o de *garbage collection* (coleta de lixo). Sua função é a de varrer a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Se por um lado isso pode deixar o aplicativo um pouco mais lento, por manter uma thread paralela que dura todo o tempo de execução do programa, evita problemas como referências perdidas e avisos de falta de memória quando sabe-se que há memória disponível na máquina.

Veja no Anexo I, as classes Java utilizadas no desenvolvimento do programa.

Nas subseções abaixo é possível encontrar informações sobre as funcionalidades dos itens da tela do programa da interface.

Cada subseção traz informações sobre a área circulada na figura

6.1.2. Configuração

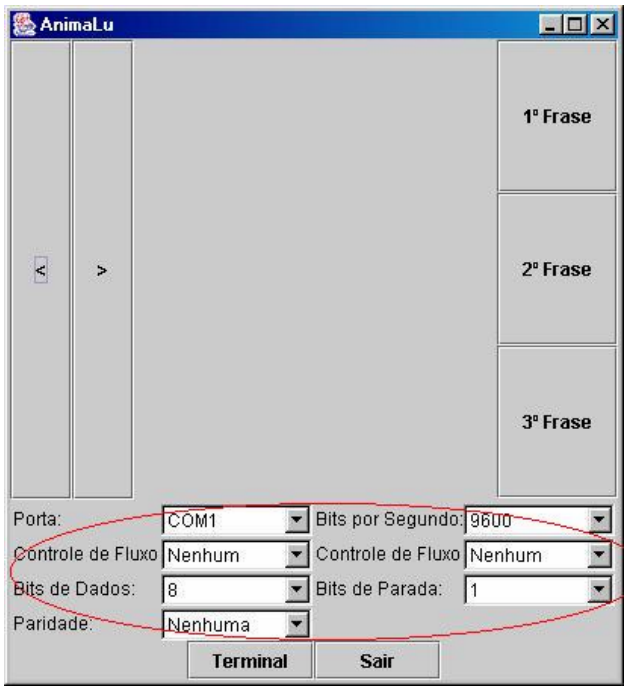

Figura 6-1 - Configuração da porta

É necessário efetuar a configuração correta desta tela para que os dados possam ser corretamente enviados ao microcontrolador.

Como está sendo utilizado a porta serial, é necessário ajustar as configurações “bits por segundo” com a velocidade igual a 9600, o valor de “bits de dados” igual à 8, e ajustar à porta serial disponível no sistema.

O sistema é capaz de buscar as portas seriais disponíveis e mostra quando há uma porta em uso. (Para mais informações sobre transmissão de dados seriais, veja Anexo E -- Comunicação com o Computador)

6.1.3. Executando a movimentação

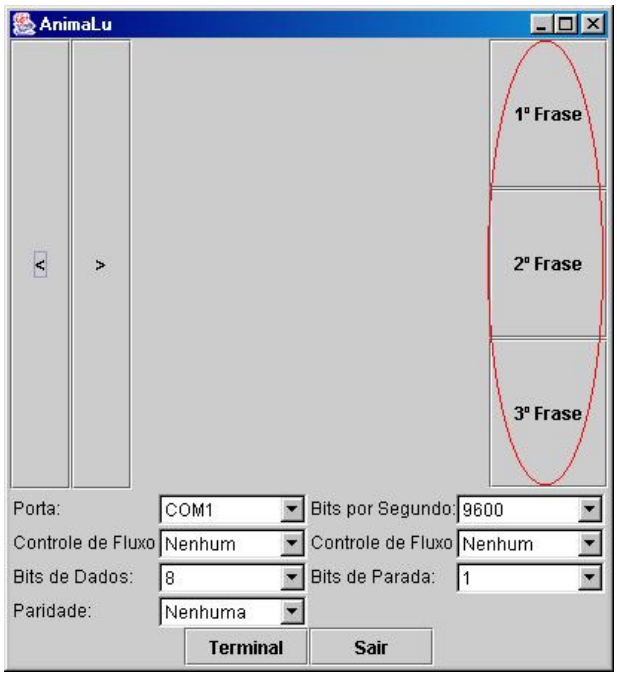

Figura 6-2 - Iniciando a fala no animatrônico

Ao pressionar qualquer um dos botões da direita em destaque, a interface envia um sinal ao microcontrolador e inicia o processo de execução do áudio, efetuando um controle de forma a sincronizar o áudio com o envio de pulsos. A interface envia pulsos que variam de 1 a 5, formando aproximadamente um ângulo de 180º entre eles. Quando o áudio encerra sua execução, a interface envia uma informação ao microcontrolador e este envia o pulso ao servo, fazendo que o boneco retorne à posição inicial que é a boca fechada.

6.1.4. Opções de manutenção

6.1.4.1. Simulação o envio de pulsos

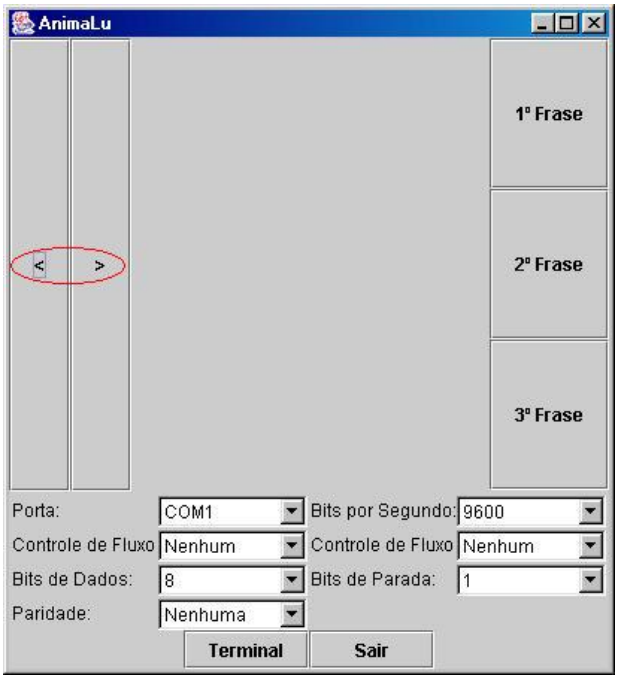

Figura 6-3 - Simulando envio de pulsos

Envia pulsos gradativos ao motor de forma a rotacioná-lo.

Simula as teclas de 1 a 5 do teclado no terminal de acordo com a programação inserida no microcontrolador (Veja no Anexo F -- Código fonte – Microcontrolador).

6.1.4.2. Terminal

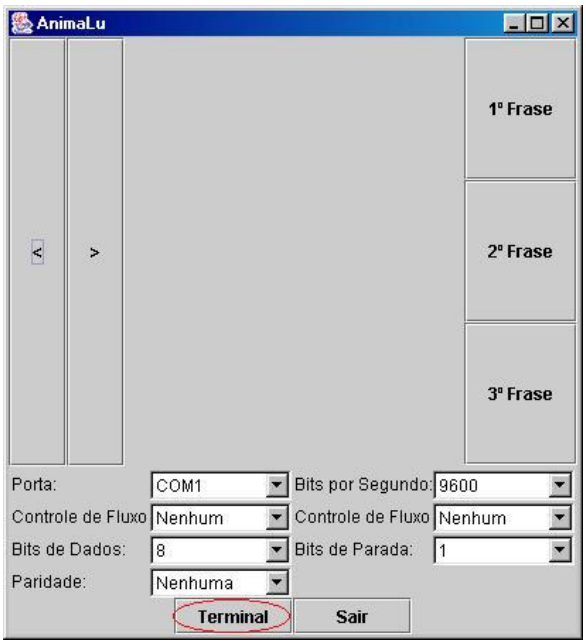

Figura 6-4 - Menu/Terminal Java

Abre a tela de um terminal para verificação através de saídas do microcontrolador e testes de envio e recebimento de dados pela serial.

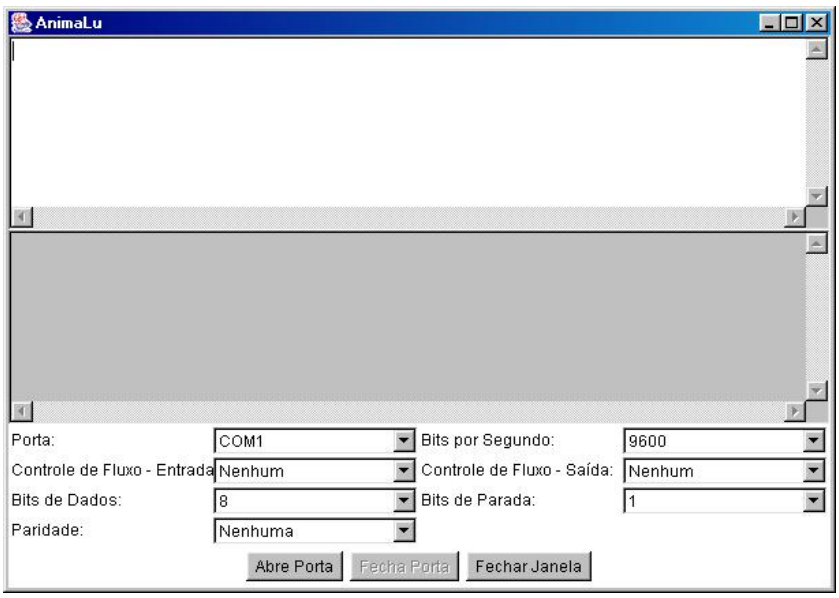

Figura 6-5 - Terminal Java

Como é visto na Figura 6-5, o terminal possui dois campos de área de texto, sendo o primeiro àquele que permite enviar os dados à serial e o segundo recebe dados da porta.

CAPÍTULO 7

MONTAGEM DO PROTÓTIPO

7.1. MONTAGEM DO PROTÓTIPO (PARTE ELETRÔNICA)

O protótipo do circuito foi simulado primeiramente com a ajuda do software *Proteus*. Como pode ser visto na figura 7-1.

Quando um novo componente (microcontrolador) é adicionado à tela de simulação, é possível perceber que os pinos de alimentação positiva e terra, não se encontram no desenho, valendo ressaltar o fato de ser de extrema importância a ligação correta destes pinos no protoboard ou no desenho feito para o circuito impresso.

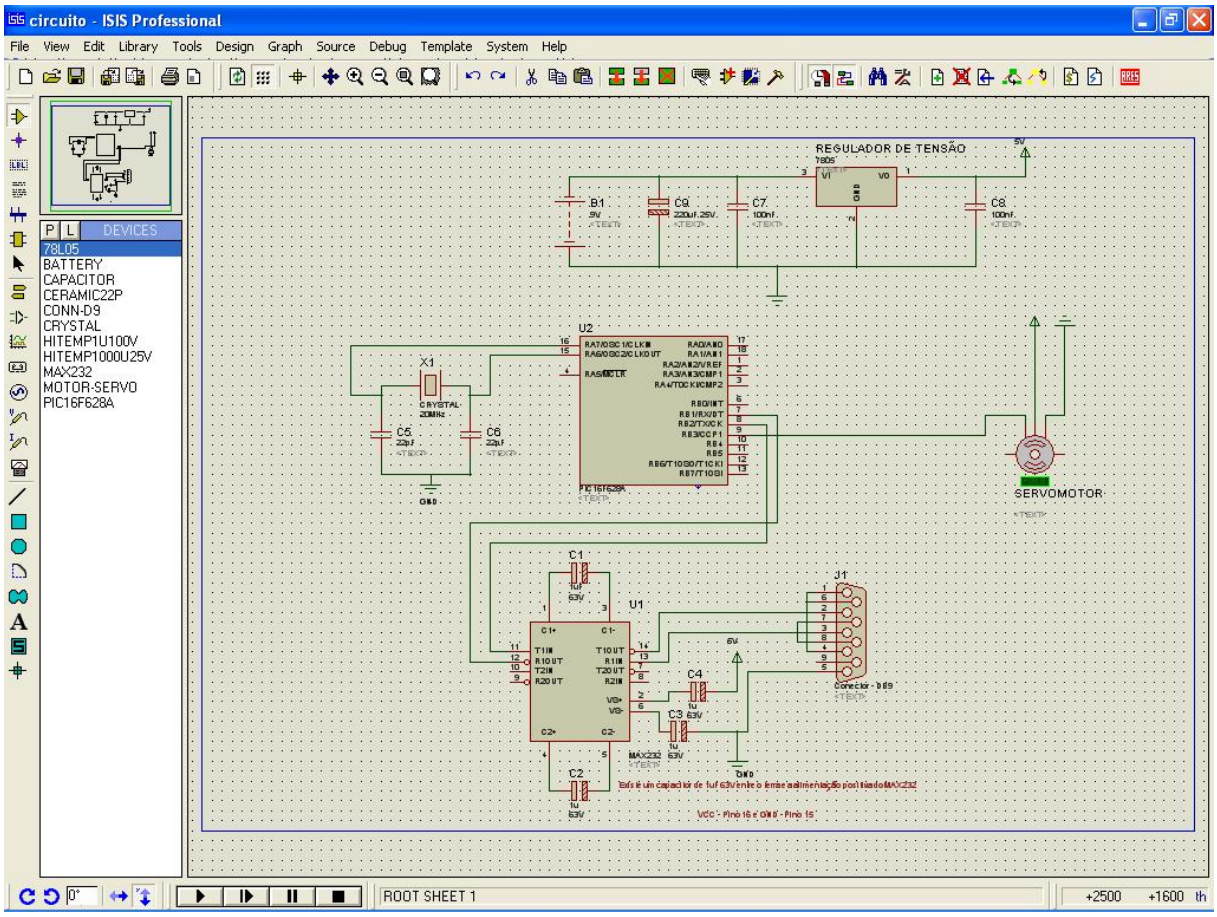

Figura 7-1 - Desenho do circuito montado no software Proteus

7.2. FINALIZAÇÃO DA MONTAGEM DO PROTÓTIPO

Na Figura 7-2 - Estruturas mecânica e eletrônica é possível visualizar a correta junção das estruturas para a montagem do protótipo.

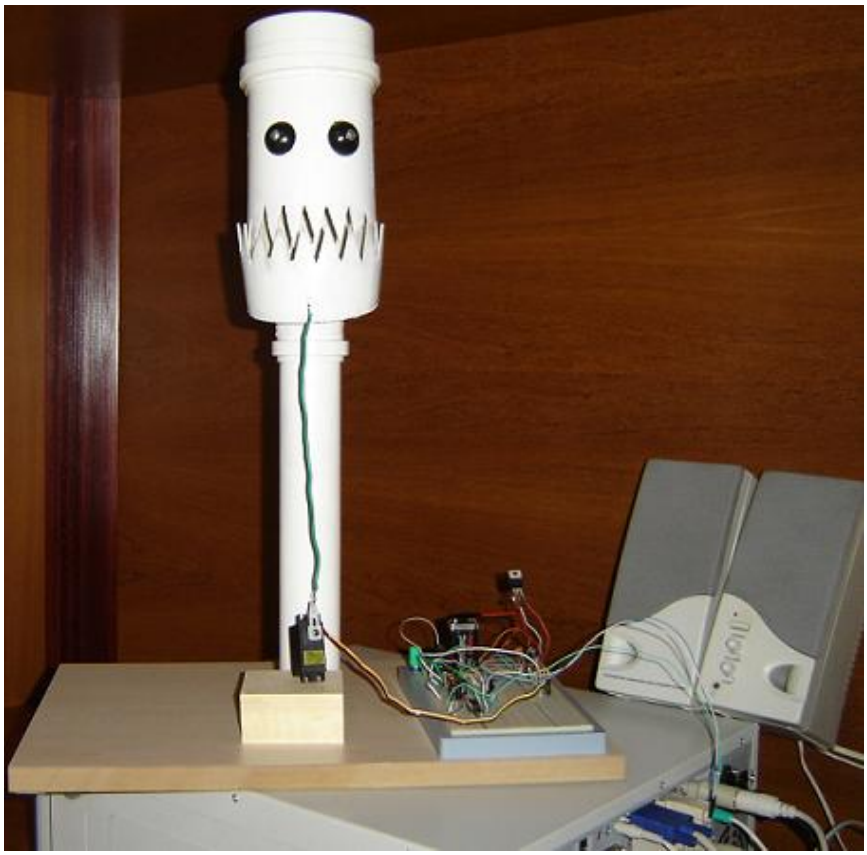

Figura 7-2 - Estruturas mecânica e eletrônica

CAPÍTULO 8

CONSIDERAÇÕES FINAIS

8.1. CONCLUSÃO

A utilização dos animatrônicos é, em sua grande maioria, para o entretenimento, porém nada impede que o estudo feito sobre ativação de motores com um microcontrolador PIC, possa ser utilizado em outras áreas, com aplicações diferentes, como por exemplo, a movimentação de um braço mecânico e outros.

No projeto eletrônico, quanto à alimentação do circuito e do motor, foi utilizada uma bateria de nove volts, porém é importante salientar que não é correto trabalhar com baterias juntamente com motores, pois os mesmos quando estão em altas rotações consomem muita energia, sendo assim, seria adequado trocar a bateria por uma fonte. Como o protótipo é pequeno e foi utilizado um motor de especificação simples, uma bateria de nove volts atende perfeitamente a situação. Porém, em animatrônicos de grande porte, esta mesma bateria seria insuficiente para a movimentação correta do motor, pois seria descarregada rapidamente.

A principal dificuldade foi o fato do microcontrolador PIC16F628A não ter sido encontrado em Brasília, sendo necessário comprá-lo no Estado de São Paulo, acarretando atrasos durante a fase de testes na implementação da movimentação do motor. Esse problema poderia ser resolvido com a troca do microcontrolador por um outro da família PIC que atendesse os mesmos requisitos, porém, os modelos possíveis para a troca também não foram encontrados.

Na primeira tentativa de desenvolvimento do código de acionamento do servo utilizando-se de diretivas sugeridas pela literatura encontrada a respeito do PWM, não se obteve sucesso. A princípio, não se era possível saber qual é o prescaler adotado por estas funções (Veja no Anexo F. Código fonte – Microcontrolador – Utilização do PWM).

Nessa primeira análise realizada, surgiram dúvidas sobre o motor a ser utilizado, que poderia não ser adequado para a aplicação. Sendo assim, foi feito uma nova análise sugerindo a troca do servomotor por um motor de passo, revalidando o código utilizado anteriormente. Porém, a melhor alternativa foi realizar novas alterações no código, sendo necessário calcular e estabelecer um prescaler manualmente de acordo com as necessidades do motor. Nessa tentativa o êxito foi alcançado (Veja no Anexo F. Código fonte – Microcontrolador – Código utilizado no projeto).

8.2. SUGESTÕES PARA TRABALHOS FUTUROS

A estrutura de movimentação de motores permite uma gama de aplicações diferentes.

- Ainda no tema animatrônicos pode-se aproveitar a estrutura deste projeto e criar uma espécie de “motor rítmico”, ou seja, um circuito que aciona um motor a partir dos sinais de áudio que sejam injetados no circuito. Quando os sons se tornam mais fortes, isto é, nos “picos”, o circuito liga e o motor gira. Nos pontos de mínimos, o motor não é acionado e fica parado. Assim, com as variações de picos e ausências de som que ocorrem normalmente quando uma pessoa fala, o motor irá rodar e parar de funcionar acompanhando a voz da pessoa.
- No assunto de motores poderia ser desenvolvido um protótipo de cadeira de rodas automática com detectores de obstáculos.
- Implementação de novas funcionalidades ao animatrônico feito neste trabalho, como a construção de um braço mecânico, por exemplo.
- O servomotor escolhido é mais utilizado em aplicações com aeromodelos, por esta razão poderia ser desenvolvida alguma solução dentro da área de aeromodelismo.

REFERÊNCIAS BIBLIOGRÁFICAS

[ASM51 - 2005] – ASM51, **FÓRUM PIC**. Disponível em: < http://www.asm51.eng.br/forum/default.asp>. (Acesso em:11 mai 2005).

[BERNARDES - 2003] – BERNARDES, **LUIZ HENRIQUE. REVISTA MECATRÔNICA FÁCIL** – Ano 2 – Nº. 11 – Editora Saber, 2003.

[CANZIAN - 2005] – CANZIAN, Edmur., **COMUNICAÇÃO SERIAL – RS232**. Disponível em: <http://www.eletricazine.hpg.ig.com.br/apostilas/telecomunicacoes/comun_serial.pdf>. (Acesso em: 11 mai 2005).

[DEITEL - 2001] – DEITEL, Harvey M., DEITEL, Paul J., **JAVA: COMO PROGRAMAR** – 3ª Edição. Editora Bookman, 2001.

[DINIZ - 2005] – DINIZ ESTEVES ENTREPRISE, **SERVOS "COMMAND" HOBBICO**. Disponível em: <http://www.dinizesteves.com.br/dicas_artigo.php?codigo=21>. (Acesso em:10 mai 2005).

[FLADEMIR - 2005] – FLADEMIR AEROMODELISMO, **SERVOS**. Disponível em: <http://www.flademir.com.br/artigos/artigo_5.asp>. (Acesso em: 10 mai 2005.)

[MATIC – 2005] – MATIC, Nebojsa., **MPLAB.** Disponível em: <http://www.mikroelektronika.co.yu/portuguese/product/books/picbook/capitulo5.htm>. (Acesso em: 29 abr. 2005).

[MAX232 - 2005] – MAXIM. **DATASHEET MAX 232**. Disponível em: < http://www.mosaico-eng.com.br/arquivos/MAX232.pdf>. (Acesso em: 23 mai 2005).

[MAXIM - 2005] – MAXIM, **DATA SHEET MAX232**. Disponível em: <http://www.mosaico-eng.com.br/arquivos/MAX232.pdf>. (Acesso em: 03 mai. 2005).

[NORTON - 1997] – NORTON, Peter., **INTRODUÇÃO À INFORMÁTICA**. Editora Makron Books, 1997.

[PEREIRA - 2003] - PEREIRA, Fábio., **MICROCONTROLADORES PIC - PROGRAMAÇÃO EM C.** Editora Érica, 2003.

[PEREIRA - 2004] - PEREIRA, Fábio., **MICROCONTROLADORES PIC - TÉCNICAS AVANÇADAS**. Editora Érica, 2004.

[ROBOARTE – 2005] – **ROBOARTE MECATRÔNICA**. Disponível em: <http://www.roboarte.com.br> (Acesso em: 14 mar 2005).

[SALOMON - 2001] – SALOMON, Délcio Vieira., **COMO FAZER UMA MONOGRAFIA**. Editora Martins Fontes, 2001.

[SERVOMOTORES - 2005] -- ROBÓTICA EDUCACIONAL, **SERVOMOTORES** . Disponível em: <http://www.geocities.com/Eureka/Enterprises/3754/robo/interf/servom.htm>. (Acesso em: 10 mai 2005).

[SOUZA LAVINIA - 2005] - SOUZA, David José., LAVINIA, Nicolás César., **PIC16F877A - CONECTANDO O PIC – RECURSOS AVANÇADOS**. Editora Érica, 2005.

[SOUZA - 2005] - SOUZA, David José., **DESBRAVANDO O PIC – AMPLIADO E ATUALIZADO PARA PIC16F628A**. Editora Érica, 2005.

[SUN COMM - 2005] – JAVA COMMUNICATIONS API – **CLASSES JAVA PARA COMUNICAÇÃO SERIAL**. Disponível em: <http://java.sun.com/products/javacomm>. (Acesso em: 26 mai 2005).

[SUN JMF - 2005] – JAVA MEDIA FRAMEWORK API (JMF) – **CLASSES JAVA PARA ÁUDIO**. Disponível em: <http://java.sun.com/products/java-media/jmf/> (Acesso em:26 mai 2005).

[WHITE - 1997] – WHITE, Ron. **COMO FUNCIONA O COMPUTADOR III**. Editora Quark, 1997.

ANEXOS

Anexo A. MONTAGEM DO PROTÓTIPO

- **1º PASSO**

Foi utilizado um pedaço de madeira (cortado um pequeno furo ao centro utilizando o serra-copo) para encaixar um parafuso de 0,46 mm de diâmetro.

O pedaço de madeira foi encaixado no tubo de PVC de 75 mm e fixado com o auxílio de cinco pregos, para que a estrutura fique bem firme e não se solte.

Para este primeiro passo realiza-se a sustentação do tubo de PVC para o encaixe na base de madeira.

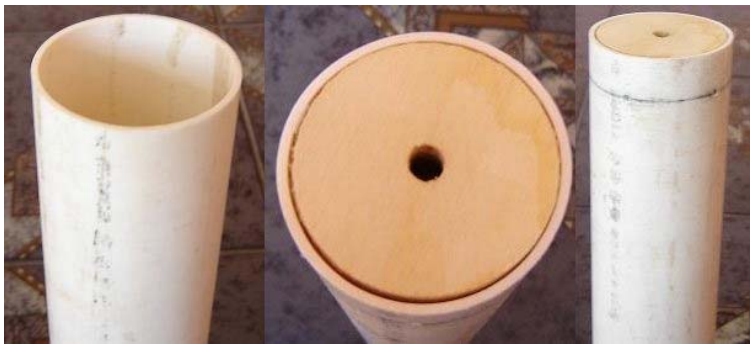

Figura (Anexo) - 1 - Montagem do tubo de 75mm

- **2º PASSO**

Para a base de madeira de 40 x 30,5 cm foi feito um furo (utilizando-se a furadeira) para encaixar um parafuso de 0,46 mm de diâmetro, utilizado para sustentar o tubo à base. O tubo de PVC de 75mm foi direcionado, com a parte contendo o encaixe de madeira, para baixo, e o parafuso de maior comprimento foi utilizado para unir o pedaço de madeira ao tubo de PVC de 75mm. Uma porca foi utilizada para segurar esse parafuso.

Figura (Anexo) - 2 - Base para sustentação do suporte da cabeça.

- **3º PASSO**

Para a parte inferior da boca, um tubo de PVC (100 mm) foi cortado com um tamanho de seis centímetros de altura. Em uma das extremidades do tubo foram desenhados dentes de dois centímetros de altura por dois centímetros de largura. O acabamento nas laterais foi feito na forma de elipse para prender o parafuso.

Com a furadeira foram feitos dois buracos, um de cada lado para encaixar um parafuso de 0,46 mm de diâmetro.

Foi feito também um buraco bem pequeno no centro da peça cortada.

Figura (Anexo) - 3 - Parte inferior da boca do animatrônico.

- **4º PASSO**

Para a parte superior do animatrônico, em um tubo de PVC (100 mm) foram desenhados dentes de forma que a parte inferior da boca encaixasse neste tubo.

Um parafuso de 0,46 mm de diâmetro deve transpassar a parte de PVC que compreende a feita no 3º passo e a peça a ser montada neste 4º passo. Para isso, foi necessário marca-la e furá-la no ponto correto de forma que a cabeça e o tubo fixe bem na base de madeira.

Um furo (com a mesma broca da furadeira) foi feito entre os dois furos de sustentação entre as peças do 3º e 4º passo.

Figura (Anexo) - 4 - Montagem da parte superior da cabeça do animatrônico.

- **5º PASSO**

Foi feito um corte com a serra tico-tico na luva de redução em um pedaço de aproximadamente 9,5 cm de largura por 5,0 cm de altura.

A peça feita no 4º passo foi utilizada como molde para marcar corretamente os pontos para a passagem dos parafusos.

Figura (Anexo) - 5 - Peça para encaixe no tubo de 75mm.

- **6º PASSO**

Foi encaixada a parte inferior da luva de redução no tubo de PVC de 75 mm.

Figura (Anexo) - 6 -Encaixe da Luva de redução no tubo de 75 mm.

A Figura (Anexo) - 7 relaciona a quantidade de parafusos, porcas e arruelas utilizados. Sendo que o parafuso maior já foi utilizado no 2º passo.

Lembrando que a cada conexão de peças foi utilizada uma arruela.

Figura (Anexo) - 7 - Porcas, parafusos e arruelas utilizados para a montagem do protótipo.

A peça feita no 4º passo foi encaixada na luva de redução, tomando cuidado para alinhar os pontos onde serão passados os parafusos.

Figura (Anexo) - 8 - Encaixe da cabeça na luva de redução.

Foi encaixada a parte inferior da boca no boneco, e presos corretamente os parafusos.

Foram colocadas duas porcas nos dois parafusos de articulação (laterais) da peça para evitar que as porcas se soltem ao abrir e fechar a estrutura inferior da boca.

Após prender todos os parafusos foi encaixado na parte superior o tampão de 100 mm.

Figura (Anexo) - 9 - Encaixe do tampão de 100 mm

Figura (Anexo) - 10 - Protótipo

Anexo B. DESCRIÇÃO DOS PINOS - PIC 16F628A

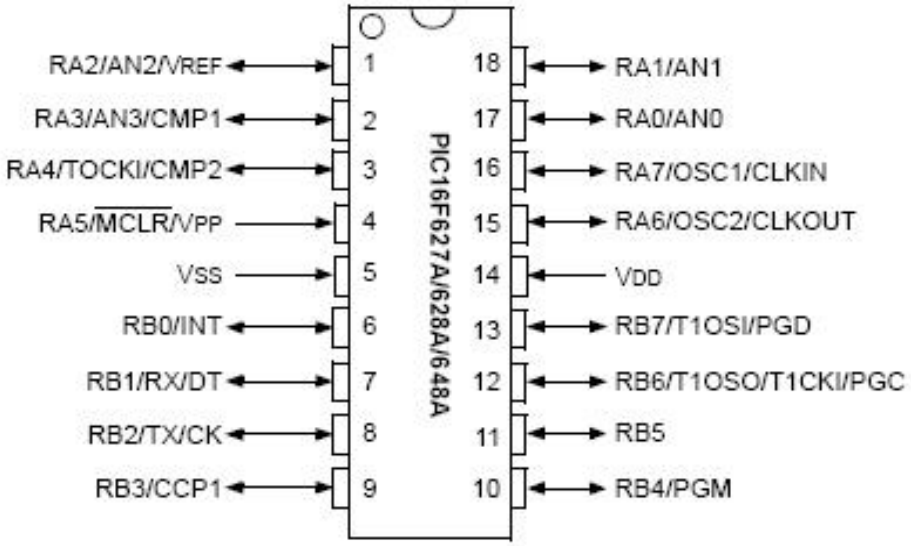

Figura (Anexo) - 11 - Descrição dos pinos - PIC16F628A

Pino	Função	Tipo	Descrição
1	RA2/AN2/Vref	Entrada / saída	Porta A (bit 2) / entrada do comparador analógico / saída da referência de tensão.
2	RA3/AN3/CMP1	Entrada / saída	Porta A (bit 3) / entrada do comparador analógico / saída do comparador 1.
3	RA4/T0CKI/CMP2	Entrada / saída	Porta A (bit 4) / entrada de clock externo do timer 0 / saída do comparador 2.
4	RA5/MCLR/THV	Entrada	Porta A (bit 5) / reset CPU / tensão de programação.
5	Vss	Alimentação	Terra.
6	RB0/INT	Entrada / saída	Porta B (bit 0) / entrada de interrupção Externa.
7	RB1/RX/DT	Entrada / saída	Porta B (bit 1) / recepção USART (modo assíncrono) / dados (modo síncrono).
8	RB2/TX/CK	Entrada / saída	Porta B (bit 2) / transmissão USART (modo assíncrono) / clock (modo síncrono).
9	RB3/CCP1	Entrada / saída	Porta B (bit 3) / entrada / saída do módulo

			CCP.
10	RB4/PGM	Entrada / saída	Porta B (bit 4) / entrada de programação LVP.
11	RB5	Entrada / saída	Porta B (bit 5).
12	RB6/T1OSO/T1CKI	Entrada / saída	Porta B (bit 6) / saída do oscilador TMR1 / entrada clock TMR1.
13	RB7/T1OSI	Entrada / saída	Porta B (bit 7) / entrada do oscilador TMR1.
14	Vdd	Alimentação	Alimentação positiva.
15	RA6/OSC2/CLKOUT	Entrada / saída	Porta A (bit 6) / entrada para o cristal oscilador / saída de clock.
16	RA7/OSC2/CLKIN	Entrada / saída	Porta A (bit 7) / entrada para o cristal oscilador / entrada de clock externo.
17	RA0/AN0	Entrada / saída	Porta A (bit 0) / entrada comparador analógico.
18	RA1/AN1	Entrada / saída	Porta A (bit 1) / entrada comparador analógico.

Tabela (Anexo) - 1 - Descrição dos Pinos – PIC 16F628A

Anexo C. DESCRIÇÃO DOS PINOS MAX232

CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figura (Anexo) - 12 - Descrição dos Pinos e capacitores do MAX 2XX

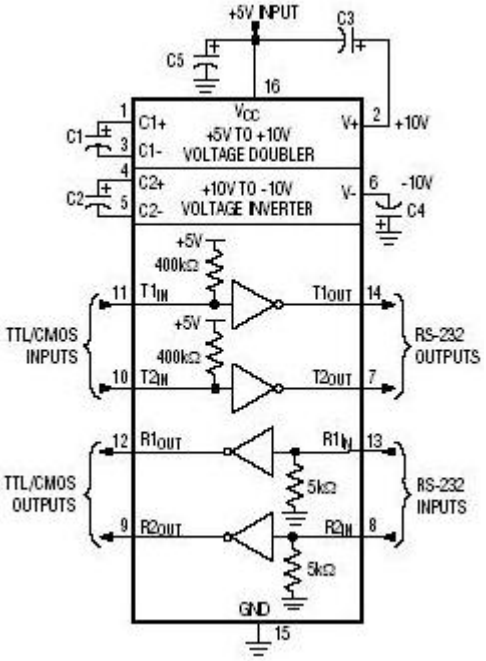

Figura (Anexo) - 13 - Montagem do circuito com RS232

Anexo D. DIRETIVAS DO COMPILADOR CCS

Todo compilador, seja ele C, Pascal, BASIC, etc., possui uma lista de comandos internos que não são diretamente traduzidos em código. Esses comandos são, na realidade, utilizados para especificar determinados parâmetros internos utilizados pelo compilador no momento de compilar o código-fonte e são chamados de diretivas do compilador ou diretivas do pré-processador. **[PEREIRA - 2003]**

O compilador CCS C possui uma grande quantidade de diretivas, que podem ser utilizadas para controlar diversos parâmetros, além de outros usos.

Abaixo será encontrada uma lista de algumas das muitas funções permitidas pelo compilador.

- **DIRETIVAS DO COMPILADOR**

#FUSES

Esta diretiva é utilizada para programar as opções da palavra de configuração (configuration word) dos PICs.

Sintaxe:

#fuses opções

Na tabela abaixo será possível encontrar somente as *opções* utilizadas neste projeto.

OPÇÃO	DESCRIÇÃO
HS	Oscilador HS
NOWDT	Watchdog desabilitado
PUT	Temporizador de Power-up desligado
NOBROWNOUT	Reset por queda de tensão desabilitado
NOLVP	Programação em baixa tensão desabilitada
NOMCLR	Pino MCLR utilizado como entrada

Tabela (Anexo) - 2 - Diretivas #fuses

Exemplo:

#fuses HS, NOWDT, PUT, NOBROWNOUT, NOLVP, NOMCLR

#INT_XXX

Informa ao programa que a função seguinte será utilizada para tratamento de interrupção.

A lista de sintaxes deste comando é extensa, porém aqui só será encontrada a diretiva utilizada na programação do PIC.

Sintaxe:

#INT_TIMER1 – TMR1IF – Estouro de contagem do timer 1

Exemplo:

```
#int_timer1
void trata_t1 ()
{
        x++;
}
```

#USE RS232

Ativa o suporte à comunicação serial.

Sintaxe:

#use rs232 (opções)

OPÇÃO	DESCRIÇÃO
BAUD = Valor	Especifica a velocidade de comunicação serial.
XMIT = Pino	Especifica o pino de transmissão de dados.
RCV = Pino	Especifica o pino de recepção de dados.

Tabela (Anexo) - 3 - Diretivas #RS232

Esta diretiva permite a construção de interfaces seriais por software (ou hardware se ele estiver presente), permitindo a utilização de funções como **putc, printf** e **getc**.

Esta diretiva permanece ativa até que outra diretiva #RS232 seja definida.

- **DIRETIVAS DE ATRASO**

DELAY_CYCLES ()

Aguarda n ciclos de máquina.

Sintaxe:

```
delay_cycles ( n )
```

Em que:

n é uma constante inteira de 8 bits.

Esta função retarda a execução do programa por *n* ciclos de máquina.

Exemplo:

delay_cycles (5); // atrasa 5 ciclos de máquina.

- **DIRETIVAS DE ENTRADA / SAÍDA**

OUTPUT_LOW ()

Coloca o pino especificado no microcontrolador em nível 0.

Sintaxe:

output_low (pino)

Em que:

Pino é um identificador do pino do microcontrolador.

Exemplo:

output_low (pin_b1); // coloca o pino RB1 em 0

OUTPUT_HIGH ()

Coloca o pino especificado do microcontrolador em nível 1.

Sintaxe:

output_high (pino)

Em que:

Pino é um identificador do pino do microcontrolador.

Exemplo:

output_high (pin_a0); // coloca o pino RA0 em 1

- **MANIPULAÇÃO DE TIMERS**

SETUP_TIMER_1 ()

Configura o timer 1.

Sintaxe:

setup_timer_1 (modo)

Em que:

- *modo* é uma variável ou constante inteira de 8 bits.

O *modo* pode ser uma ou mais constantes dos seguintes grupos:

- T1_DISABLED: desliga o timer 1;
- T1_INTERNAL: com clock interno;
- T1_EXTERNAL: clock externo assíncrono;
- T1_EXTERNAL_SYNC: clock externo síncrono;
- T1_CLOCK_OUT: ativa saída externa de clock;
- T2_DIV_BY_1: prescaler do timer 1 dividindo pro 1;
- T2_DIV_BY_2: prescaler do timer 1 dividindo pro 2;
- T2_DIV_BY_4: prescaler do timer 1 dividindo pro 4;
- T2_DIV_BY_8: prescaler do timer 1 dividindo pro 8.

As constantes anteriores podem ser juntadas por meio de operação OR (|).

Exemplo:

setup_timer_1 (T1_INTERNAL | T1_DIV_BY_4);

SET_TIMERX ()

Modifica o conteúdo de um timer interno.

Sintaxe:

set_rtcc (valor)

set_timer0 (valor)

set_timer1 (valor)

set_timer2 (valor)

set_timer3 (valor)

Em que:

Valor é a variável ou constante inteira de 8 ou 16 bits.

O valor a ser escrito depende do tipo de temporizador e do modelo do PIC em questão.

Timer1, de 16 bits para o modelo PIC16F628A.

- **COMPARAÇÃO / CAPTURA / PWM**

SETUP_CCPX()

Configura o funcionamento do watchdog.

Sintaxe:

```
setup_ccp1 ( modo )
setup_ccp2 ( modo )
```

Em que:

- *modo* é uma variável ou constante inteira de 8 bits.

Configura o módulo de captura, comparação e geração de sinais PWM. As constantes definidas para configuração do modo são:

- CCP_OFF: módulo CCP desligado;
- CCP_CAPTURE_FE: captura na borda de descida do sinal;
- CCP_CAPTURE_RE: captura na borda de subida do sinal;
- CCP_CAPTURE_DIV_4:captura a cada 4ª borda de subida do sinal;
- CCP_CAPTURE_DIV_16:captura a cada 16ª borda de subida do sinal;
- CCP_COMPARE_SET_ON_MATCH: modo de comparação, seta o pino CCPx;
- CCP_COMPARE_CLR_ON_MATCH: modo de comparação, apaga pino CCPx;
- CCP_COMPARE_INT: modo de comparação com geração de interrupção;
- CCP_COMPARE_RESET_TIMER: modo de comparação com reset do timer1 ou timer3;

- CCP_PWM: configura o modo de geração de sinal PWM.

O compilador define ainda as variáveis inteiras longas CCP_1 e CCP_2 para acesso aos registradores de comparação do microcontrolador.

Exemplo:

```
setup_ccp1 ( CCP_CAPTURE_RE ).
```

SET_PWMX_DUTY()

Configura o ciclo ativo do módulo CCP no modo PWM.

Sintaxe:

```
set_pwm1_duty ( valor )
set_pwm2_duty ( valor )
```

Em que:

- *valor* é uma variável ou constante inteira de 8 ou 16 bits.

O valor especificado como argumento da função pode ser de 8 ou 16 bits:

- Se 8 bits, o valor é armazenado diretamente no registrador CCPR1L ou CCPR2L, dependendo do módulo CCP em questão;
- Se 18 bits, o valor é rotacionado de forma a ser armazenado nos registradores CCPR1L ou CCPR2L e CCP1CON ou CCP2CON, dependendo do módulo CCP em questão.

Note que a utilização de valores de 8 bits é mais eficiente que valores de 16 bits.

Exemplo:

```
set_pwm1_duty ( 100 );
```

- **CONTROLE DO PROCESSADOR**

ENABLE_INTERRUPTS ()

Habilita uma interrupção.

Sintaxe:

enable_interrupts (valor)

Em que:

Valor é uma constante inteira de 8 bits;

Esta função pode ser utilizada para ativar um ou mais tipos de interrupção do processador.

O valor utilizado como argumento deve ser uma das constantes:

- GLOBAL - GIE: Habilita global de interrupções;
- INT_TIMER1 – TMR1IF: Estouro de contagem do timer1.

É possível agrupar mais de uma constante em uma mesma função, utilizando o operador OR (|), desde que estejam localizadas no mesmo registrador de controle.

Exemplo:

enable_interrupts (GLOBAL | INT_TIMER1) // habilita o bit GIE e TMR1IF.

- **COMUNICAÇÃO SERIAL ASSÍNCRONA**

GETC()

Aguarda a chegada de um caractere pela porta serial padrão e retorna o seu valor.

Sintaxe:

```
valor = getc()
valor = getch()
valor = getchar()
```

Em que:

- *valor* é um valor inteiro de 8 bits.

A função GETC aguarda a chegada de um caractere pela linha serial.

Caso seja utilizada a USART interna, esta função pode retornar imediatamente caso existam caracteres no buffer de recepção (que armazena um máximo de três caracteres).

Observe que no caso de não utilização da USART interna, somente pode ocorrer à recepção de um caractere durante a execução da função.

O caractere recebido é retornado pela função.

Requisito: #use rs232

Exemplo:

```
#uses rs232 (baud = 4800, xmit = pin_b3, rcv = pin_b2)
int x;
x = getc(); // aguarda a recepção de um caractere pela interface serial
```

KBHIT ()

Verifica se há um caractere sendo recebido pela porta serial padrão (STDIN).

Sintaxe:

```
res = kbhit ()
```

Em que:

- *res* é um valor booleano

Esta função verifica se há um caractere sendo recebido (no caso da interface serial por software) ou um caractere recebido está disponível no buffer de recepção da USART (RCREG).

Note que para não ocorrerem perdas de dados no caso de comunicação por software, esta função deve ser chamada no mínimo a uma taxa de dez vezes a velocidade de comunicação utilizada. Ou seja: para uma velocidade de 1200 Bps, esta função deveria ser chamada 12000 vezes por segundo.

A função retorna 1 (true) caso um caractere esteja disponível para ser lido, ou 0 (false) caso não exista caractere disponível para recepção.

Requisito: #use rs232

Exemplo:

```
Boolean erro_timeout;
...
char getc_temporizado();
/* esta função aguarda 0,5 segundo para a recepção de um caractere. Caso seja recebido algum nesse período, ele é retornado pela função. Caso contrário, a função retorna o valor 0; */
{
        long tempo;
        erro_timeout = false;
        while ( !kbhit () && (++tempo<50000) ) delay_us (10);
        if ( kbhit() ) return ( getc() ); else
        {
                erro_timeout = true;
                return (0);
        }
}
```

Anexo E. COMUNICAÇÃO COM O COMPUTADOR

- **INTERFACE SERIAL**

Com a interface serial, os bits dados fluem um de cada vez em um único arquivo.

Na transmissão serial um único circuito é suficiente para transferir um sinal de um ponto a outro, porém, o computador não oferece nenhuma garantia que lerá sempre o primeiro bit de cada série quando começar a acompanhar a marcha dos sinais seriais; basta um erro nesse momento para que todos os bytes subseqüentes sejam interpretados equivocadamente.

Um dos métodos para solucionar esse problema é a transmissão assíncrona. A alternativa é acrescentar marcas aos *strings* de dados, indicando o início e o fim de cada bloco de dados. O sistema receptor poderá, então, identificar o início de cada série e evitar erros de interpretação sem que precise recorrer, para isso, a um sinal de sincronização.

Na maioria dos sistemas assíncronos, os dados são divididos em pequenos grupos, cada um dos quais correspondem aproximadamente a um byte. Esses grupos de dados são chamados de *palavras*, e podem ter entre cinco e oito bits de dados. Os tamanhos de palavras mais comuns têm sete e oito bits: o primeiro, porque é suficiente para comportar todos os caracteres ASCII maiúsculos e minúsculos; o segundo, porque cada palavra corresponde exatamente a um byte de dados.

Um pulso muito especial de comprimento duplo denominado *start bit* (bit de início, ou bit de partida) é acrescentado a esses dados, e indica o início de uma palavra de dados. Um *stop bit* (bit de fim, ou bit de término) indica o fim da palavra. Entre o último bit de uma palavra e o *stop bit*, há geralmente um bit de *paridade* para verificação da integridade dos dados. **[WHITE - 1997]**.

O padrão atual para as comunicações seriais chama-se RS232, mas há muitas variações. Por exemplo, uma porta serial pode ter 9 ou 25 pinos.

- **O QUE É RS232 ?**

RS é uma abreviação de “ Recommended Standard ”, que em português significa Padrão Recomendado. Ela relata uma padronização de uma interface comum para comunicação de dados entre equipamentos, criada no início dos anos 60, por um comitê conhecido atualmente como “Electronic Industries Association” (EIA). Naquele tempo, a comunicação de dados compreendia a troca de dados digitais entre um computador central (mainframe) e terminais de computador remotos, ou entre dois terminais sem o envolvimento do computador. Estes dispositivos poderiam ser conectados através de linha telefônica, e conseqüentemente necessitavam um modem em cada lado para fazer a decodificação dos sinais. **[CANZIAN - 2005].**

Dessas idéias nasceu o padrão RS232. Ele especifica as tensões, temporizações e funções dos sinais, um protocolo para troca de informações, e as conexões mecânicas. Há mais de 30 anos desde que essa padronização foi desenvolvida, a EIA publicou três modificações.

A mais recente, EIA232E, foi introduzida em 1991. Ao lado da mudança de nome de RS232 para EIA232, algumas linhas de sinais foram renomeadas e várias linhas novas foram definidas.

- **TAXA DE TRANSFERÊNCIA (BAUD RATE)**

A taxa de transferência é a velocidade com que as informações (dados) são enviadas através de um canal, sendo medido em transições elétricas por segundo. Na norma EIA232, ocorre uma transição de sinal por bit, e a taxa de transferência e a taxa de bit (bit rate) são idênticas. Neste caso, uma taxa de 9600 bauds corresponde a uma transferência de 9600 dados por segundos, ou um período de aproximadamente, 104 μs (1/9600 s).

Outro conceito é a eficiência do canal de comunicação que é definido como o número de bits de informação utilizável (dados) enviados através do canal por segundo. Ele não inclui bits de sincronismo, formatação, e detecção de erro que podem ser adicionados à informação antes da mensagem ser transmitida, e sempre será no máximo igual a um.

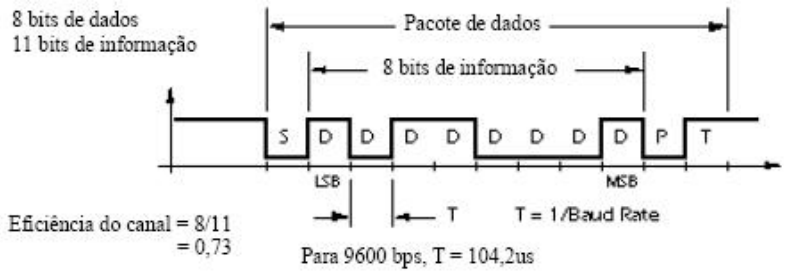

Figura (Anexo) - 14 - Baud rate

- **DEFINIÇÃO DE SINAIS**

Se a norma EIA232 completa for implementada, o equipamento que faz o processamento dos sinais é chamado DTE (Data Terminal Equipment – usualmente um computador ou terminal), tem um conector DB9 macho. O equipamento que faz a conexão (no caso do projeto será um MAX232) é denominado de DCE (Data Circuit-terminating Equipment), tem um conector DB9 fêmea. Um cabo de extensão entre dispositivos DTE e DCE contém ligações em paralelo, não necessitando mudanças na conexão de pinos. Se todos os dispositivos seguissem essa norma, todos os cabos seriam idênticos, e não haveria chances de haver conexões incorretas.

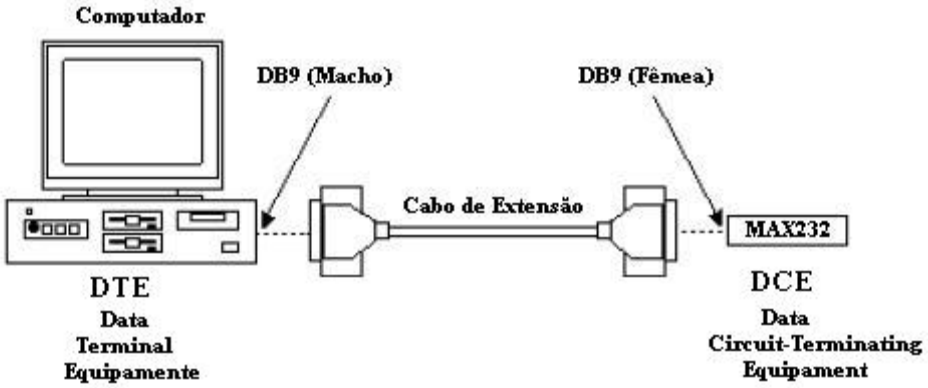

Figura (Anexo) - 15 - Conexão DTE/DCE

- **CONECTORES**

Na figura a seguir é apresentada à definição dos sinais para um dispositivo DTE (usualmente um micro PC). Os sinais mais comuns são apresentados em negrito.

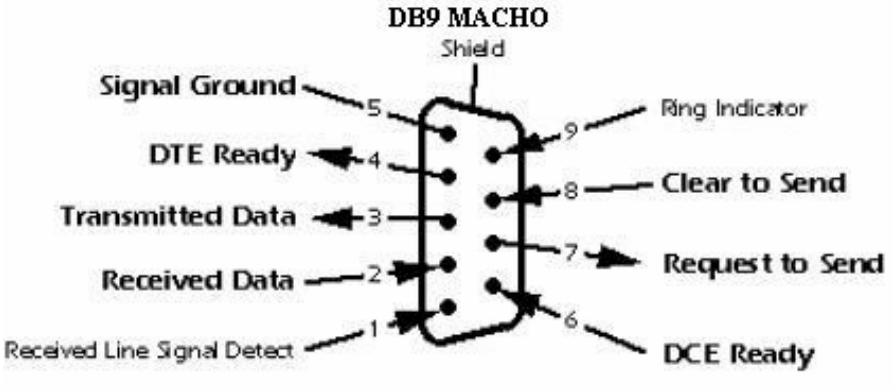

Figura (Anexo) - 16 - DB9 – Macho

Na Figura (Anexo) - 17 é apresentada a definição dos sinais para um dispositivo DCE. Os sinais mais comuns são apresentados em negrito.

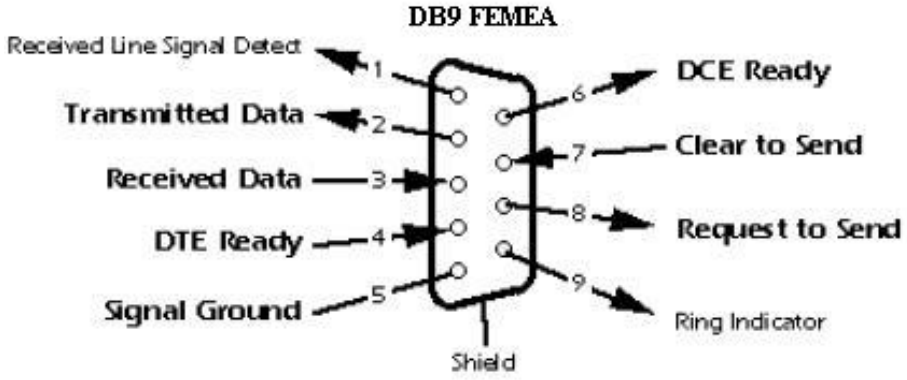

Figura (Anexo) - 17 - DB9 – Fêmea

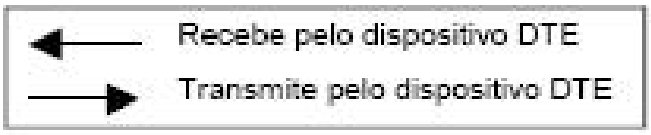

Figura (Anexo) - 18 - Definições dos sinais do DB9 Macho/Fêmea

A Tabela (Anexo) - 4 apresenta a convenção utilizada para os sinais mais comuns do conector DB-9:

Pino	Nome	Descrição
1	Carrier Detect (CD)	Também chamado de Data Carrier Detect (DCD). Este sinal é relevante quando o DCE for um modem. Ele é habilitado (nível lógico “0”) quando a linha telefônica está “fora do gancho”, uma conexão for estabelecida, e um tom de resposta começar a ser recebido do modem remoto. Este sinal é desabilitado (nível lógico “1”) quando não houver tom de resposta sendo recebido, ou quando o tom de resposta for de qualidade inadequada para o Modem local.
2	Received	Este sinal está ativo quando o DTE receber dados do DCE. Quando o DCE estiver

	Data (RxD)	em repouso, o sinal é mantido na condição de marca (nível lógico “1”, tensão negativa).
3	Transmitted Data (TxD)	Este sinal está ativo quando dados estiverem sendo transmitidos do DTE para o DCE. Quando nenhum dado estiver sendo transmitido, o sinal é mantido na condição de marca (nível lógico “1”, tensão negativa).
4	DTE Ready (DTR)	Também chamado de Data Terminal Ready. Este sinal é habilitado (nível lógico “0”) pelo DTE quando for necessário abri o canal de comunicação. Se o DCE for um modem, a habilitação do sinal DTR prepara o modem para ser conectado ao circuito do telefone, e uma vez conectado, mantém a conexão. Quando o sinal DTR for desabilitado (nível lógico “1”), o modem muda para a condição “no gancho” e termina a conexão.
5	Ground (GND)	Sinal de terra utilizado como referência para outros sinais.
6	DCE Ready (DSR)	Também chamado de Data Set Ready. Quando originado de um modem, este sinal é habilitado (nível lógico “0”) quando as seguintes forem satisfeitas: 1 – O modem estiver conectado a uma linha telefônica ativa e “fora do gancho”; 2 – O modem estiver no modo dados; 3 – O modem tiver completado a discagem e está gerando um tom de resposta. Se a linha for tirada do gancho, uma condição de falha for detectada, ou uma conexão de voz for estabelecida, o sinal DSR é desabilitado (nível lógico “1”).
7	Request To Send (RTS)	Este sinal é habilitado (nível lógico “0”) para preparar o DCE para aceitar dados transmitidos pelo DTE. Esta preparação inclui a habilitação dos circuitos de recepção, ou a seleção a direção do canal em aplicações half-duplex. Quando o DCE estiver pronto, ele responde habilitando o sinal CTS.
8	Clear To Send (CTS)	Este sinal é habilitado (nível lógico “0”) pelo DCE para informar ao DTE que a transmissão pode começar. Os sinais RTS e CTS são comumente utilizados no controle do fluxo de dados em dispositivos DCE.
9	Ring Indicator (RI)	Este sinal é relevante quando o DCE for um modem, e é habilitado (nível lógico “0”) quando um sinal de chamada estiver sendo recebido na linha telefônica. A habilitação desse sinal terá aproximadamente a duração do tom de chamada, e será desabilitado entre os tons ou quando não houver tom de chamada presente.

Tabela (Anexo) - 4 - Descrição dos pinos do conector DB9

Anexo F. CÓDIGO FONTE – MICROCONTROLADOR

- **UTILIZAÇÃO DO PWM**

Primeiro código desenvolvido, porém sem sucesso para a movimentação do Hobbico CS-60.

```
#include <16F628A.h>
#fuses HS,NOWDT, NOLVP, PUT, NOBROWNOUT, NOMCLR
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_B2, rcv=PIN_B1)

main()
{
	long int ciclo=0;
	printf ("\r\nTeste do PWM\r\n");
	setup_timer_2 (T2_DIV_BY_16, 255, 16);
	setup_ccp1 (ccp_pwm);  // configura CCP1 para modo PWM
	set_pwm1_duty ( 0 );  // configura o ciclo ativo em 0 (desligado)
	while (true)
	{
		if (kbhit()) // se uma tecla for pressionada
		{
			switch (getc()) // verifica a tecla
			{
				case '1' :  ciclo = 50;
						break;
				case '2' :  ciclo = 100;
						break;
				case '3' :  ciclo = 255;
						break;
				case '4' :  ciclo = 350;
						break;
				case '5' :  ciclo = 500;
						break;
				case '6' :  ciclo = 700;
						break;
				case '7' :  ciclo = 900;
						break;
				case '8' :  ciclo = 1023;
						break;
				case '0' :  ciclo = 0;
						break;
				case '-' :  ciclo -= 10;
						break;
				case '=' :  ciclo += 10;
						break;
			}
			if (ciclo > 1023) ciclo = 1023;
			printf ("Ciclo ativo =  %lu\r\n", ciclo);
			set_pwm1_duty (ciclo); // configura o ciclo ativo
		}
```

```
        }
}
```

- **CÓDIGO UTILIZADO NO PROJETO**

```
#include <16f628A.h>
#use delay(clock=20000000)
#fuses HS, NOWDT, PUT, NOBROWNOUT, NOLVP, NOMCLR
#use rs232 (baud=9600, xmit=PIN_B2,rcv=PIN_B1)

static int ciclo=0;
#int_timer1
void trata_t1()
{
  static int conta;
  set_timer1(64911+get_timer1());

  if(ciclo > 0)
  {
    conta++;

    if(conta == ciclo)
    {
      output_low(PIN_B3);
    }

    if(conta == 40)
    {
      conta = 0;
      output_high(PIN_B3);
    }
  }
}
main() {
  setup_timer_1(T1_INTERNAL | T1_DIV_BY_4);
  set_timer1(64911);
  enable_interrupts(global);
  enable_interrupts(int_timer1);
  printf("\r\nTeste do PWM\r\n");
  while(true)
  {
    if(kbhit())
    {
      switch(getc())
      {
        case '0': ciclo = 0;

 break;
        case '1': ciclo = 1;
                  break;
        case '2': ciclo = 2;
                  break;
        case '3': ciclo = 3;
```

```
                break;
            case '4': ciclo = 4;
                break;
            case '5': ciclo = 5;
                break;
        }
        if(ciclo > 40) ciclo=40;
        printf("Ciclo ativo: = %u\r\n",ciclo);
      }
    }
}
```

Anexo G. TERMINAL PARA ENVIO E RECEBIMENTO DE DADOS.

- **HYPERTERMINAL**

Os códigos descritos acima apresentam, por exemplo, comandos do tipo ***printf***. Este comando é somente para controle do programador e pode ser verificado no software Hyperterminal do Sistema operacional Windows.

Com a seguinte configuração:

1º PASSO.

Na guia Iniciar/ Todos os programas / Acessórios / Comunicação é encontrada o software Hyperterminal. Ao abrir este software é necessário informar um nome qualquer para a conexão.

Figura (Anexo) - 19 - Informando 'nome' para conexão com hyperterminal

2º PASSO

Após informar nome da conexão, é necessário clicar no botão OK, e selecionar a porta COM1.

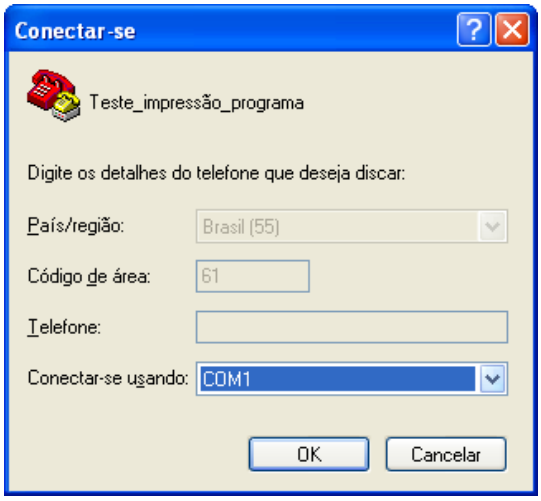

Figura (Anexo) - 20 - Informa porta de comunicação para Hyperterminal

Após informar a porta de comunicação, é necessário clicar no botão OK, e selecionar a velocidade da porta. Como os testes estão sendo efetuados com a porta serial, no campo bits por segundo deve-se informar a velocidade de 9600.

Deve-se confirmar a seleção com o botão OK, e o ambiente está configurado para os devidos testes.

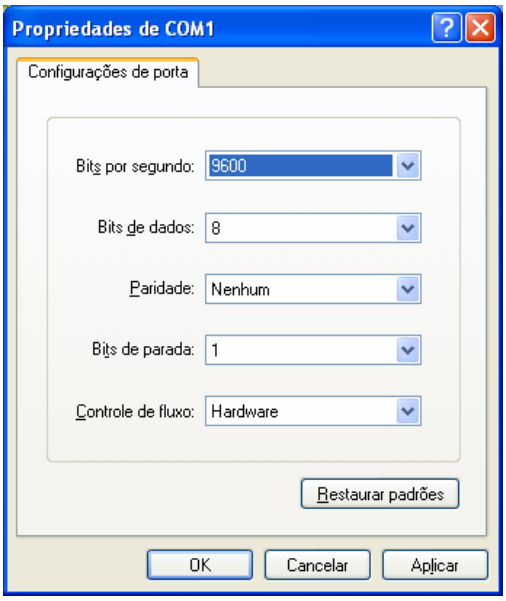

Figura (Anexo) - 21 - Informar velocidade da Porta para Hyperterminal

Anexo H. MATERIAIS UTILIZADOS (CIRCUITO ELETRÔNICO)

- **Microcontrolador**
  - MAX232;
  - PIC16F628A.
- **Socket**
  - Socket de 16 pinos para o MAX232;
  - Socket de 18 pinos para o PIC16F628A.
- **Capacitores**
  - 5 capacitores eletrolíticos de 1µF x 16V;
  - 1 capacitor eletrolítico de 220mF x 25V;
  - 2 capacitores cerâmicos de 22pF;
  - 2 capacitores cerâmicos de 100Nf.
- 1 Regulador de Tensão 7805;
- Conector DB9 fêmea (para placa);
- Cabo de extensão DB9 (macho-fêmea);
- Cristal 20MHZ.

Anexo I. ESPECIFICAÇÃO DAS CLASSES UTILIZADAS:

- **TELAPRINCIPAL.JAVA**

Classe principal da interface. Responsável pela montagem da tela e eventos dos botões.

Inclui as configurações de porta, com combos de: Portas disponíveis no sistema, Velocidade de porta, Bits por Segundo, Controle de Fluxo - Entrada e Saída, Bits de Dados, Bits de Parada e Paridade.

BOTÕES DE FALA DO ANIMATRÔNICO

Envia informações do arquivo de áudio a ser utilizado e os parâmetros de configuração da porta. Chama o método da classe responsável pela fala.

BOTÕES DE ENVIO GRADATIVO DE PULSOS

Não emitem som ao animatrônico mas enviam os pulsos de forma a movimentá-lo. A cada clique no botão ">", aumenta um pulso e um clique no botão "<" diminui um pulso.

BOTÃO TERMINAL

Abre uma tela de terminal, classe TerminalSerial.java

BOTÃO SAIR

Finaliza o sistema.

CÓDIGO-FONTE

```
/*
 * Created on 30/05/2005
 *
 * Classe Responsável por receber comando da interface
 * e tratar envio de pulsos ao microcontrolador.
 */

import java.awt.BorderLayout;
import java.awt.Choice;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Label;
```

```
import java.awt.Panel;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.util.Enumeration;
import javax.comm.CommPortIdentifier;
import javax.swing.JButton;
import javax.swing.JFrame;

/**
 * @author Luciane Barbosa de Andrade
 * Este é o programa principal da interface
 */

public class TelaPrincipal extends JFrame implements ActionListener {

	//Declaração de Atributos da classe
static final long serialVersionUID = 0;
	final int HEIGHT = 450;
final int WIDTH  = 410;

private static int pulso = 0;
private String mov = "";
private JButton terminalBotao;
private JButton sairBotao;
private JButton oiBotao;
private JButton boaNoiteBotao;
	private JButton nomeBotao;
private Panel botoesPainel;
private Panel botoesDireita;
private JButton cimaBotao;
private JButton baixoBotao;
private Panel animatronico;
private BorderLayout layout;
	private PainelConfiguracao painelConfiguracao;
	private SerialParameters parameters;

	/**
Método Principal
*/

   public static void main(String[] args) {
		TelaPrincipal telaPrincipal = new TelaPrincipal();
		telaPrincipal.setVisible(true);
		telaPrincipal.repaint();
   }

   /**
   Cria a <code>TelaPrincipal</code> e a inicializa.
   */

   public TelaPrincipal(){
		super("AnimaLu");
```

```
parameters = new SerialParameters();

layout = new BorderLayout(2,1);
Container c = getContentPane();
c.setLayout(layout);

animatronico = new Panel();

//Painel do animatronico com botoes de sobe e desce parte inferior da boca
animatronico.setLayout(new GridLayout(1, 2));

baixoBotao = new JButton("<");
baixoBotao.addActionListener(this);
baixoBotao.setEnabled(true);
baixoBotao.setSize(0,0);
animatronico.add(baixoBotao);

cimaBotao = new JButton(">");
cimaBotao.addActionListener(this); //cria uma ação
cimaBotao.setEnabled(true);
cimaBotao.setSize(0,0);
animatronico.add(cimaBotao);

c.add(animatronico, "West"); //adiciona painel à esquerda

//Painel dos botoes
botoesDireita = new Panel();
botoesDireita.setLayout(new GridLayout(3, 1));

oiBotao = new JButton("1º Frase");
oiBotao.addActionListener(this); //cria uma ação
oiBotao.setSize(1,1);
botoesDireita.add(oiBotao);

boaNoiteBotao = new JButton("2º Frase");
boaNoiteBotao.addActionListener(this);
boaNoiteBotao.setEnabled(true);
boaNoiteBotao.setSize(1,1);
botoesDireita.add(boaNoiteBotao);

nomeBotao = new JButton("3º Frase");
nomeBotao.addActionListener(this);
nomeBotao.setEnabled(true);
nomeBotao.setSize(1,1);
botoesDireita.add(nomeBotao);
c.add(botoesDireita, "East");

painelConfiguracao = new PainelConfiguracao(this);

//Painel dos botoes
botoesPainel = new Panel();
botoesPainel.setLayout(new GridLayout(1, 2));

terminalBotao = new JButton("Terminal");
```

```
		terminalBotao.addActionListener(this);
		terminalBotao.setEnabled(true);
		terminalBotao.setSize(2,2);
		botoesPainel.add(terminalBotao);

		sairBotao = new JButton("Sair");
		sairBotao.addActionListener(this);
		sairBotao.setEnabled(true);
		sairBotao.setSize(2,2);
		botoesPainel.add(sairBotao);

		c.add(botoesPainel, "South");

		Panel southPanel = new Panel();

		GridBagLayout gridBag = new GridBagLayout();
		GridBagConstraints cons = new GridBagConstraints();

		southPanel.setLayout(gridBag);

		cons.gridwidth = GridBagConstraints.REMAINDER;
		gridBag.setConstraints(painelConfiguracao, cons);
		cons.weightx = 1.0;
		southPanel.add(painelConfiguracao);
		gridBag.setConstraints(botoesPainel, cons);
		southPanel.add(botoesPainel);

		c.add(southPanel, "South");

		Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

		setLocation(screenSize.width/2 - WIDTH/2,
			screenSize.height/2 - HEIGHT/2);

		setSize(WIDTH, HEIGHT);
}

/**
Responde a eventos dos botões.
*/
public void actionPerformed(ActionEvent e) {
		String comando = e.getActionCommand();

		// Abre o terminal.
		if (comando.equals("Terminal")) {
			TerminalSerial terminal = new TerminalSerial( );
			terminal.setVisible(true);
		}

		// Sair do sistema.
		if (comando.equals("Sair")) {
		   shutdown();
		}

		// Faz o animatrônico falar a primeira frase.
```

```
if (comando.equals("1º Frase")) {
	File file = new File("audio/oi.wav");
	painelConfiguracao.setParameters();
	Talker talker = new Talker(file, parameters);
	talker.talk();

}

// Faz o animatrônico falar a segunda frase.
if (comando.equals("2º Frase")) {
	File file = new File("audio/oi.wav");
	painelConfiguracao.setParameters();
	Talker talker = new Talker(file, parameters);
	talker.talk();
}

// Faz o animatrônico falar a terceira frase.
if (comando.equals("3º Frase")) {
	File file = new File("audio/oi.wav");
	painelConfiguracao.setParameters();
	Talker talker = new Talker(file, parameters);
	talker.talk();
}

//Faz o servo movimentar-se aumentando os pulsos.
if (comando.equals(">")) {
	painelConfiguracao.setParameters();
	if (pulso < 5){
		pulso++;
	}
	try {
		ServoController servo = new ServoController(parameters);
		mov = Integer.toString(pulso);
		System.out.println("Valor do pulso: " + mov);
		servo.movimentarServo(mov, 1000);
		servo.disconnect();
	} catch (IOException e1) {
		e1.printStackTrace();
	} catch (InterruptedException e1) {
		e1.printStackTrace();
	} catch (SerialConnectionException e1) {
		e1.printStackTrace();
	}
}

// Faz o servo movimentar-se diminuindo os pulsos.
if (comando.equals("<")) {
	painelConfiguracao.setParameters();
	if (pulso > 1){
		pulso--;
	}
	try {
		ServoController servo = new ServoController(parameters);
		mov = Integer.toString(pulso);
		System.out.println("Valor do pulso: " + mov);
```

```
                                servo.movimentarServo(mov, 1000);
                                servo.disconnect();
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        } catch (InterruptedException e1) {
                                e1.printStackTrace();
                        } catch (SerialConnectionException e1) {
                                e1.printStackTrace();
                        }
                }
}

/**
Elemento do painel que prende o uso da aplicação para
a configuração da conexão.
*/
class PainelConfiguracao extends Panel{

                private static final long serialVersionUID = 1L;

                private Frame parent;

                private Label portNameLabel;
                private Choice portChoice;

                private Label baudLabel;
                private Choice baudChoice;

                private Label flowControlInLabel;
                private Choice flowChoiceIn;

                private Label flowControlOutLabel;
                private Choice flowChoiceOut;

                private Label databitsLabel;
                private Choice databitsChoice;

                private Label stopbitsLabel;
                private Choice stopbitsChoice;

                private Label parityLabel;
                private Choice parityChoice;

                /**
                Cria e inicializa o painel de configurações. As configurações iniciais
                vem dos parametros dos objetos.
                */
                public PainelConfiguracao(Frame parent) {
                   this.parent = parent;

                   setLayout(new GridLayout(4, 4));

                   //Cria combo com portas disponíveis no sistema.
                   portNameLabel = new Label("Porta:", Label.LEFT);
                   add(portNameLabel);
```

```
        portChoice = new Choice();
        add(portChoice);

        //Busca dinamicamente as portas disponíveis
        listPortChoices();
        portChoice.select(parameters.getPortName());


        /*Cria combos com seleção de:
         * "Bits por Segundo"
         * "Controle de Fluxo - Entrada e Saida
         * "Bits de Dados"
         * "Bits de Parada"
         * "Paridade"
        */

        baudLabel = new Label("Bits por Segundo:", Label.LEFT);
        add(baudLabel);

        baudChoice = new Choice();
        baudChoice.addItem("300");
        baudChoice.addItem("2400");
        baudChoice.addItem("9600");
        baudChoice.addItem("14400");
        baudChoice.addItem("28800");
        baudChoice.addItem("38400");
        baudChoice.addItem("57600");
        baudChoice.addItem("152000");
        baudChoice.select(Integer.toString(parameters.getBaudRate()));
        add(baudChoice);

    flowControlInLabel = new Label("Controle de Fluxo - Entrada:", Label.LEFT);
        add(flowControlInLabel);

        flowChoiceIn = new Choice();
        flowChoiceIn.addItem("Nenhum");
        flowChoiceIn.addItem("Xon/Xoff In");
        flowChoiceIn.addItem("RTS/CTS In");
        flowChoiceIn.select(parameters.getFlowControlInString());
        add(flowChoiceIn);

     flowControlOutLabel = new Label("Controle de Fluxo - Saída:", Label.LEFT);
        add(flowControlOutLabel);

        flowChoiceOut = new Choice();
        flowChoiceOut.addItem("Nenhum");
        flowChoiceOut.addItem("Xon/Xoff Out");
        flowChoiceOut.addItem("RTS/CTS Out");
        flowChoiceOut.select(parameters.getFlowControlOutString());
        add(flowChoiceOut);

        databitsLabel = new Label("Bits de Dados:", Label.LEFT);
        add(databitsLabel);
```

```
            databitsChoice = new Choice();
            databitsChoice.addItem("5");
            databitsChoice.addItem("6");
            databitsChoice.addItem("7");
            databitsChoice.addItem("8");
            databitsChoice.select(parameters.getDatabitsString());
            add(databitsChoice);

            stopbitsLabel = new Label("Bits de Parada:", Label.LEFT);
            add(stopbitsLabel);

            stopbitsChoice = new Choice();
            stopbitsChoice.addItem("1");
            stopbitsChoice.addItem("1.5");
            stopbitsChoice.addItem("2");
            stopbitsChoice.select(parameters.getStopbitsString());
            add(stopbitsChoice);

            parityLabel = new Label("Paridade:", Label.LEFT);
            add(parityLabel);

            parityChoice = new Choice();
            parityChoice.addItem("Nenhuma");
            parityChoice.addItem("Uniforme");
            parityChoice.addItem("Impar");
            parityChoice.select("Nenhuma");
            parityChoice.select(parameters.getParityString());
            add(parityChoice);
        }

        /**
        Busca configurações do painel acima e guarda.
        */
        public void setConfigurationPanel() {
            portChoice.select(parameters.getPortName());
            baudChoice.select(parameters.getBaudRateString());
            flowChoiceIn.select(parameters.getFlowControlInString());
            flowChoiceOut.select(parameters.getFlowControlOutString());
            databitsChoice.select(parameters.getDatabitsString());
            stopbitsChoice.select(parameters.getStopbitsString());
            parityChoice.select(parameters.getParityString());
        }

        /**
        Busca configurações guardadas e seta na conexão.
        */
    public void setParameters() {
        parameters.setPortName(portChoice.getSelectedItem());
        parameters.setBaudRate(baudChoice.getSelectedItem());
        parameters.setFlowControlIn(flowChoiceIn.getSelectedItem());
        parameters.setFlowControlOut(flowChoiceOut.getSelectedItem());
        parameters.setDatabits(databitsChoice.getSelectedItem());
        parameters.setStopbits(stopbitsChoice.getSelectedItem());
        parameters.setParity(parityChoice.getSelectedItem());
    }
```

```
                /**
                Método que busca dinamicamente as portas disponíveis no sistema
                Utiliza o tipo "emuneration" de portas comm retornadas pelo
                CommPortIdentifier.getPortIdentifiers(), então seta a escolha
                atual para a combo.
                */
    void listPortChoices() {
        CommPortIdentifier portId;

        Enumeration en = CommPortIdentifier.getPortIdentifiers();

        // interage entre as portas.
        while (en.hasMoreElements()) {
            portId = (CommPortIdentifier) en.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                portChoice.addItem(portId.getName());
            }
        }
        portChoice.select(parameters.getPortName());
    }
  }


  /**
  Fecha o sistema. Primeiramente fecha qualquer porta aberta e
  a libera, então fecha.
  */
  private void shutdown() {
        System.exit(0);
  }
}
```

- **TERMINALSERIAL.JAVA**

Ao clicar no botão “Terminal” na tela principal, é chamado esta classe que permite simular um ambiente terminal para comunicação serial. Contém as combos de configuração de portas seriais (portas disponíveis no sistema, velocidade de porta, bits por Segundo, controle de Fluxo - Entrada e Saída, bits de Dados, bits de Parada e Paridade).

Possui botões para abrir ou fechar uma determinada porta serial e o botão de “Fechar Janela” que permite retornar à classe que a chamou (retorna à TelaPrincipal).

CÓDIGO-FONTE

```
/*
 * Created on 02/06/2005
 * Esta classe é responsável pelo terminal
 */
```

```
import javax.comm.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Properties;
import java.util.Enumeration;

/**
 *  Classe Principal do Terminalú.
 *  @author Luciane Barbosa de Andrade
*/

public class TerminalSerial extends Frame implements ActionListener {

        private static final long serialVersionUID = 1L;

   //constantes de tamanho da tela
   final int HEIGHT = 450;
   final int WIDTH  = 610;

   private Button openButton;
   private Button closeButton;
   private Button exitButton;
   private Panel buttonPanel;

   private Panel    messagePanel;
   private TextArea messageAreaOut;
   private TextArea messageAreaIn;

   private ConfigurationPanel configurationPanel;
   private SerialParameters parameters;
   private SerialConnection connection;

   private Properties props = null;

   /**
   Método Principal. Mostra a tela do terminal
   */
   public static void main(String[] args) {

        TerminalSerial terminal = new TerminalSerial();
        terminal.setVisible(true);
        terminal.repaint();
   }

   /**
   Prepara e inicializa o terminal.
   */
   public TerminalSerial(){

                super("AnimaLu");

                parameters = new SerialParameters();

                // Prepara a classe para tratar eventos da tela
```

```
addWindowListener(new CloseHandler(this));

//Monta painel de entrada e saida do terminal
messagePanel = new Panel();
messagePanel.setLayout(new GridLayout(2, 1));

messageAreaOut = new TextArea();
messagePanel.add(messageAreaOut);

messageAreaIn = new TextArea();
messageAreaIn.setEditable(false);
messagePanel.add(messageAreaIn);

add(messagePanel, "Center");

configurationPanel = new ConfigurationPanel(this);

//Monta botões
buttonPanel = new Panel();

openButton = new Button("Abre Porta");
openButton.addActionListener(this);
buttonPanel.add(openButton);

closeButton = new Button("Fecha Porta");
closeButton.addActionListener(this);
closeButton.setEnabled(false);
buttonPanel.add(closeButton);

exitButton = new Button("Fechar Janela");
exitButton.addActionListener(this);
buttonPanel.add(exitButton);

Panel southPanel = new Panel();

GridBagLayout gridBag = new GridBagLayout();
GridBagConstraints cons = new GridBagConstraints();

southPanel.setLayout(gridBag);

cons.gridwidth = GridBagConstraints.REMAINDER;
gridBag.setConstraints(configurationPanel, cons);
cons.weightx = 1.0;
southPanel.add(configurationPanel);
gridBag.setConstraints(buttonPanel, cons);
southPanel.add(buttonPanel);

add(southPanel, "South");

//Prepara comunicação serial
connection = new SerialConnection(this, parameters,
                                  messageAreaOut, messageAreaIn);
setConfigurationPanel();

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
```

```
                setLocation(screenSize.width/2 - WIDTH/2,
                        screenSize.height/2 - HEIGHT/2);

                setSize(WIDTH, HEIGHT);
}

/**
Configura os elementos da tela no painel.
*/
public void setConfigurationPanel() {
        configurationPanel.setConfigurationPanel();
}

/**
Responde às ações dos menus e botões.
*/
public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();

        // Sai do Programa
        if (cmd.equals("Fechar Janela")) {
                connection.closeConnection();
           this.setVisible(false);
        }

        // Abre uma porta.
        if (cmd.equals("Abre Porta")) {
           openButton.setEnabled(false);
           Cursor previousCursor = getCursor();
           setNewCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
           configurationPanel.setParameters();
           try {
                connection.openConnection();
           } catch (SerialConnectionException e2) {
                AlertDialog ad = new AlertDialog(this,
                                        "Erro ao abrir porta!",
                                        "Erro ao abrir porta,",
                                        e2.getMessage() + ".",
                                        "Refaça as configurações, então tente novamente.");
              openButton.setEnabled(true);
                setNewCursor(previousCursor);
                return;
           }
           portOpened();
           setNewCursor(previousCursor);
        }

        // Fecha uma porta.
        if (cmd.equals("Fecha Porta")) {
           portClosed();
        }

}
```

```
/**
Muda estados quando uma porta é aberta.
*/
public void portOpened() {
	openButton.setEnabled(false);
	closeButton.setEnabled(true);
}

/**
Chama o método "closeConnection" da classe "SerialConnection" e altera os botões
para o estado quando uma porta é fechada.
*/
public void portClosed() {
	connection.closeConnection();
	openButton.setEnabled(true);
	closeButton.setEnabled(false);
}

/**
Seta o cursor para a aplicação.
@param c New <code>Cursor</code>
*/
private void setNewCursor(Cursor c) {
	setCursor(c);
	messageAreaIn.setCursor(c);
	messageAreaOut.setCursor(c);
}

/**
Fecha a aplicação. Primeiro fecha qualquer porta e
a libera, depois sai do aplicativo.
*/
private void shutdown() {
	connection.closeConnection();
	//System.exit(1);
	this.setVisible(false);
}

/**
Elemento do painel que prende o uso da aplicação para
a configuração da conexão.
*/
class ConfigurationPanel extends Panel implements ItemListener {

		private static final long serialVersionUID = 1L;

		private Frame parent;

		private Label portNameLabel;
		private Choice portChoice;

		private Label baudLabel;
		private Choice baudChoice;

		private Label flowControlInLabel;
```

```
                private Choice flowChoiceIn;

                private Label flowControlOutLabel;
                private Choice flowChoiceOut;

                private Label databitsLabel;
private Choice databitsChoice;

                private Label stopbitsLabel;
                private Choice stopbitsChoice;

                private Label parityLabel;
                private Choice parityChoice;

                /**
                Cria e inicializa o painel de configurações. As configurações iniciais
                vem do parametros dos objetos.
                */
                public ConfigurationPanel(Frame parent) {
                   this.parent = parent;

                   setLayout(new GridLayout(4, 4));

                   //Cria combo com portas disponíveis no sistema.
                   portNameLabel = new Label("Porta:", Label.LEFT);
                   add(portNameLabel);

                   portChoice = new Choice();
                   portChoice.addItemListener(this);
                   add(portChoice);

                   //Busca dinamicamente as portas disponíveis
                   listPortChoices();
                   portChoice.select(parameters.getPortName());


                   /*Cria combos com seleção de:
                    * "Bits por Segundo"
                    * "Controle de Fluxo - Entrada e Saida
                    * "Bits de Dados"
                    * "Bits de Parada"
                    * "Paridade"
                   */

                   baudLabel = new Label("Bits por Segundo:", Label.LEFT);
                   add(baudLabel);

                   baudChoice = new Choice();
                   baudChoice.addItem("300");
                   baudChoice.addItem("2400");
                   baudChoice.addItem("9600");
                   baudChoice.addItem("14400");
                   baudChoice.addItem("28800");
                   baudChoice.addItem("38400");
                   baudChoice.addItem("57600");
```

```
        baudChoice.addItem("152000");
        baudChoice.select(Integer.toString(parameters.getBaudRate()));
        baudChoice.addItemListener(this);
        add(baudChoice);

flowControlInLabel = new Label("Controle de Fluxo - Entrada:", Label.LEFT);
        add(flowControlInLabel);

        flowChoiceIn = new Choice();
        flowChoiceIn.addItem("Nenhum");
        flowChoiceIn.addItem("Xon/Xoff In");
        flowChoiceIn.addItem("RTS/CTS In");
        flowChoiceIn.select(parameters.getFlowControlInString());
        flowChoiceIn.addItemListener(this);
        add(flowChoiceIn);

flowControlOutLabel = new Label("Controle de Fluxo - Saída:", Label.LEFT);
        add(flowControlOutLabel);

        flowChoiceOut = new Choice();
        flowChoiceOut.addItem("Nenhum");
        flowChoiceOut.addItem("Xon/Xoff Out");
        flowChoiceOut.addItem("RTS/CTS Out");
        flowChoiceOut.select(parameters.getFlowControlOutString());
        flowChoiceOut.addItemListener(this);
        add(flowChoiceOut);

        databitsLabel = new Label("Bits de Dados:", Label.LEFT);
        add(databitsLabel);

        databitsChoice = new Choice();
        databitsChoice.addItem("5");
        databitsChoice.addItem("6");
        databitsChoice.addItem("7");
        databitsChoice.addItem("8");
        databitsChoice.select(parameters.getDatabitsString());
        databitsChoice.addItemListener(this);
        add(databitsChoice);

        stopbitsLabel = new Label("Bits de Parada:", Label.LEFT);
        add(stopbitsLabel);

        stopbitsChoice = new Choice();
        stopbitsChoice.addItem("1");
        stopbitsChoice.addItem("1.5");
        stopbitsChoice.addItem("2");
        stopbitsChoice.select(parameters.getStopbitsString());
        stopbitsChoice.addItemListener(this);
        add(stopbitsChoice);

        parityLabel = new Label("Paridade:", Label.LEFT);
        add(parityLabel);

        parityChoice = new Choice();
        parityChoice.addItem("Nenhuma");
```

```
                parityChoice.addItem("Uniforme");
                parityChoice.addItem("Impar");
                parityChoice.select("Nenhuma");
                parityChoice.select(parameters.getParityString());
                parityChoice.addItemListener(this);
                add(parityChoice);
            }

            /**
            Busca configurações do painel acima e guarda.
            */
            public void setConfigurationPanel() {
                portChoice.select(parameters.getPortName());
                baudChoice.select(parameters.getBaudRateString());
                flowChoiceIn.select(parameters.getFlowControlInString());
                flowChoiceOut.select(parameters.getFlowControlOutString());
                databitsChoice.select(parameters.getDatabitsString());
                stopbitsChoice.select(parameters.getStopbitsString());
                parityChoice.select(parameters.getParityString());
            }

            /**
            Busca configurações guardadas e seta na conexão.
            */
        public void setParameters() {
            parameters.setPortName(portChoice.getSelectedItem());
            parameters.setBaudRate(baudChoice.getSelectedItem());
            parameters.setFlowControlIn(flowChoiceIn.getSelectedItem());
            parameters.setFlowControlOut(flowChoiceOut.getSelectedItem());
            parameters.setDatabits(databitsChoice.getSelectedItem());
            parameters.setStopbits(stopbitsChoice.getSelectedItem());
            parameters.setParity(parityChoice.getSelectedItem());
        }

            /**
            Método que busca dinamicamente as portas disponíveis no sistema
            Utiliza o tipo "emuneration" de portas comm retornadas pelo
            CommPortIdentifier.getPortIdentifiers(), então seta a escolha
            atual para a combo.
            */
void listPortChoices() {
    CommPortIdentifier portId;

    Enumeration en = CommPortIdentifier.getPortIdentifiers();

    // interage entre as portas.
    while (en.hasMoreElements()) {
        portId = (CommPortIdentifier) en.nextElement();
        if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
            portChoice.addItem(portId.getName());
        }
    }
    portChoice.select(parameters.getPortName());
}
```

```
                /**
                Método que trata os eventos das mudanças de configurações.
                Se uma porta está aberta a porta não pode ser alterada.
                Se a escolha não é suportada pelo sistema então o usuário
                será notificado e as configurações serão revertidas para a estado (escolha) inicial.
                */
                public void itemStateChanged(ItemEvent e) {
                   // Checa se uma porta está aberta.
                   if (connection.isOpen()) {
                         // Se a porta está aberta não permite alterações na escolha de porta.
                         if (e.getItemSelectable() == portChoice) {
                            // Avisa o usuário.
                            AlertDialog ad = new AlertDialog(parent, "Porta já está aberta!",
                                                             "A porta não pode ser",
                                                             "alterada enquanto uma",
                                                             "porta está aberta.");

                            // Retorna as configurações anteriores.
                            setConfigurationPanel();
                            return;
                         }
                         // Seta os parametros com as configurações anteriores.
                         setParameters();
                         try {
                            // Tentativa de mudar as configurações em uma porta aberta.
                            connection.setConnectionParameters();
                         } catch (SerialConnectionException ex) {
                            // Se as configurações não pode ser alteradas, avisa o usuário,
                            // e retorna às configurações anteriores.
                            AlertDialog ad = new AlertDialog(parent,
                                  "Configuração não suportada!",
                                  "Configuração de parametros não suportada,",
                                  "Selecione um novo valor.",
                                  "Retornando para as configurações anteriores.");
                            setConfigurationPanel();
                         }
                   } else {
                         // Enquanto uma porta não está aberta, configura os parametros.
                         setParameters();
                   }
                }
}

/**
Evento que fecha a janela do terminal. Permite que a aplicação
seja fechada com o botão de Fechar Janela.
*/
class CloseHandler extends WindowAdapter {

      TerminalSerial sd;

      public CloseHandler(TerminalSerial sd) {
         this.sd = sd;
      }
```

```
        public void windowClosing(WindowEvent e) {
           sd.shutdown();
        }
    }
}
```

- **TALKER.JAVA**

Classe que recebe os parâmetros da TelaPrincipal (parâmetros da porta “Com” e caminho do arquivo wave).

Instancia a classe player para iniciar o áudio do animatrônico.

Esta classe também inicia o processo de movimentação do servomotor (instanciando a classe ServoController), de forma sincronizada com o áudio enquanto houver continuidade deste.

CÓDIGO-FONTE

```
/*
 * Created on 02/06/2005
 *
 * Classe Responsável por receber comando da interface
 * e tratar envio de pulsos ao microcontrolador.
 */

import java.io.File;
import java.io.IOException;

import javax.media.Manager;
import javax.media.NoPlayerException;
import javax.media.Player;

/**
 * @author Luciane Barbosa de Andrade
 * Inicia o player e envia dados para a serial.
 */
public class Talker {

        private File audioFile;
        private SerialParameters serialParameters;

        public Talker(File audioFile, SerialParameters serialParameters) {
                //seta dados nos atributos da classe
                this.audioFile = audioFile;
                this.serialParameters = serialParameters;
        }

        /**
         * Classe Responsável pela criação do player e início da música
```

```
         *  Além disso envia pulso e timeout para a classe ServoController que movimentará o servo.
         */
        public void talk() {
        try {
        //instancia player e inicia wave.
        Player player = Manager.createPlayer(audioFile.toURL());
        ServoController servoController = new ServoController(this.serialParameters);
        player.start();

        //começa a movimentar o servo. Recebe como parâmetro os pulsos e o tempo do intervalo (microssegundos) entre eles.
        while (player.getState() != Player.Prefetched) {  //enquanto não concluído o audio...
                servoController.movimentarServo("34", 300);
        }
                servoController.disconnect(); //disconecta porta
                player.close(); //fecha player
                System.out.println("Fim da música e do envio de pulsos");

                //trata exceções
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                } catch (SerialConnectionException e) {
                        e.printStackTrace();
                } catch (NoPlayerException e) {
                        e.printStackTrace();
                }
        }
}
```

- **SERVOCONTROLLER.JAVA**

Uma vez instanciada esta classe em Talker.java, o processo de abertura da porta "com" escolhida é iniciado. Configuram-se os parâmetros de porta dentro da classe SerialPort instanciada (método connect()) e o envio dos dados ao servomotor (movimentarServo()).

CÓDIGO-FONTE

```
/*
 * Created on 05/06/2005
 *
 * Classe Responsável por receber envio de pulsos da
 * classe "TALK" e tempo entre os pulsos.
 */

import java.io.IOException;
import java.io.OutputStream;
import java.util.Enumeration;

import javax.comm.CommPortIdentifier;
import javax.comm.NoSuchPortException;
```

```
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.UnsupportedCommOperationException;

/**
 * @author Luciane Barbosa de Andrade
 * Inicia o envio dos pulsos e envia dados para a serial.
 */
public class ServoController {

   private boolean connected;
   private Enumeration portList;
   private CommPortIdentifier portId;
   private OutputStream outputStream;
   private SerialPort sPort;
   private SerialParameters parameters;

   public boolean isConnected() {
         //retorna estado da conexão. Se true,porta já está conectada.
                  return connected;
         }

   public ServoController(SerialParameters parameters) throws IOException, InterruptedException,
SerialConnectionException {
         //seta parametros na "parameters" da classe.
         this.parameters = parameters;
   }

public void connect() throws IOException, InterruptedException, SerialConnectionException {
                  // Obtém um objeto do tipo CommPortIdentifier para a porta que você deseja abrir.
                  try {
                           //System.out.println("Porta: " + parameters.getPortName());
                     portId = CommPortIdentifier.getPortIdentifier(parameters.getPortName());
                  } catch (NoSuchPortException e) {
                     throw new SerialConnectionException(e.getMessage());
                  }
                  // Abre a porta representada pelo objeto CommPortIdentifier. Dá
                  // à chamada do open um tempo (timeout) de 30 seconds para permitir
                  // à uma aplicação diferente a requisitar a porta se o usuário
                  // assim desejar.
                  try {
                           sPort = (SerialPort)portId.open("AnimaLu", 0);
                  } catch (PortInUseException e) {
                     throw new SerialConnectionException(e.getMessage());
                  }
                  // Seta os parametros para a conexão. Se não setar, fecha
                  // a porta antes de lançar uma exceção.
                  try {
                     setConnectionParameters();
                  } catch (SerialConnectionException e) {
                           //fecha porta caso ocorra erro.
                     sPort.close();
                     throw e;
                  }
                  //notificação de erros: "Em parada" e "dados disponíveis"
```

```
                sPort.notifyOnBreakInterrupt(true);
                sPort.notifyOnDataAvailable(true);

                //instancia classe Output Stream.
                this.outputStream = sPort.getOutputStream();
                connected = true;
}

        public void movimentarServo(String dados, long espera) {
    try {
        if (!connected) //verifica se está conectado,senão conecta
                connect();

        //começa envio de dados pela serial. Repete enquanto houver dados.
                for (int i = 0; i < dados.length(); i++) {
                        outputStream.write(dados.charAt(i));
                        outputStream.flush();
                        System.out.println("PULSO: " + dados.charAt(i));
                                Thread.sleep(espera);//espera em milisegundos.
                }
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (SerialConnectionException e) {
                        e.printStackTrace();
                }
        }
        //método para desconectar a porta
        public void disconnect() {
    if (sPort != null)
        sPort.close();
    connected = false;
        }
/**
Seta os valores para a conexão de acordo com os dados da classe "TelaPrincipal".
*/
public void setConnectionParameters() throws SerialConnectionException {

        //Seta para os valores antigos no caso de ocorrer erro.
                int oldBaudRate = sPort.getBaudRate();
                int oldDatabits = sPort.getDataBits();
                int oldStopbits = sPort.getStopBits();
                int oldParity   = sPort.getParity();
                int oldFlowControl = sPort.getFlowControlMode();

                //Seta os novos parametros.
                try {
                   sPort.setSerialPortParams(parameters.getBaudRate(),
                                parameters.getDatabits(),
                                parameters.getStopbits(),
                                parameters.getParity());
                } catch (UnsupportedCommOperationException e) {
                        parameters.setBaudRate(oldBaudRate);
                        parameters.setDatabits(oldDatabits);
```

```
                    parameters.setStopbits(oldStopbits);
                    parameters.setParity(oldParity);
                throw new SerialConnectionException("Parametro nao Suportado");
            }

            // Seta o controle de fluxo.
            try {
                sPort.setFlowControlMode(parameters.getFlowControlIn()
                                        | parameters.getFlowControlOut());
            } catch (UnsupportedCommOperationException e) {
                throw new SerialConnectionException("Controle de Fluxo nao suportado");
            }
    }
}
```

- **SERIALPARAMETERS.JAVA**

É a classe que armazena os parâmetros para portas seriais. Ela é utilizada para configurar os parâmetros dentro de atributos de classe cujos atributos são lidos e armazenados na instância da classe que abre uma porta serial.

CÓDIGO-FONTE

```
import javax.comm.*;
/**
Classe que armazena parametros para portas seriais.
*/
public class SerialParameters {

    private String portName;
    private int baudRate;
    private int flowControlIn;
    private int flowControlOut;
    private int databits;
    private int stopbits;
    private int parity;

    /**
    Construtor da classe. Configura os parametros para nenhuma porta, velocidade 9600, sem controle de fluxo, 8 data bits, 1 stop bit, e sem paridade.
    */
    public SerialParameters () {
            this("",
                9600,
                SerialPort.FLOWCONTROL_NONE,
                SerialPort.FLOWCONTROL_NONE,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE );

    }
```

```
/**
Construtor parametrizado.

@param portName Nome da porta.
@param baudRate Taxa de transmissão.
@param flowControlIn Tipo de fluxo de controle para recebimento.
@param flowControlOut Tipo de fluxo de controle para envio.
@param databits Numero de dados.
@param stopbits Numero de dados de parada.
@param parity Tipo de paridade.
*/
public SerialParameters(String portName,
                        int baudRate,
                        int flowControlIn,
                        int flowControlOut,
                        int databits,
                        int stopbits,
                        int parity) {

        this.portName = portName;
        this.baudRate = baudRate;
        this.flowControlIn = flowControlIn;
        this.flowControlOut = flowControlOut;
        this.databits = databits;
        this.stopbits = stopbits;
        this.parity = parity;
}

/**
Configura o nome da porta.
@param portName Nome da porta.
*/
public void setPortName(String portName) {
        this.portName = portName;
}

/**
Busca o nome da porta.
@return Retorna o nome da porta atual.
*/
public String getPortName() {
        return portName;
}

/**
Configura a taxa de transmissao.
@param baudRate Nova taxa.
*/
public void setBaudRate(int baudRate) {
        this.baudRate = baudRate;
}

/**
Configura a taxa de transmissao.
@param baudRate Nova taxa formato String.
```

```
*/
public void setBaudRate(String baudRate) {
	this.baudRate = Integer.parseInt(baudRate);
}

/**
Retorna a taxa em <code>int</code>.
@return Taxa atual.
*/
public int getBaudRate() {
	return baudRate;
}

/**
Retorna a taxa em <code>String</code>.
@return Taxa atual.
*/
public String getBaudRateString() {
	return Integer.toString(baudRate);
}

/**
Seta o controle de fluxo para leitura.
@param flowControlIn Novo controle de fluxo para leitura.
*/
public void setFlowControlIn(int flowControlIn) {
	this.flowControlIn = flowControlIn;
}

/**
Seta o fluxo de controle para leitura.
@param flowControlIn Novo controle de fluxo para leitura.
*/
public void setFlowControlIn(String flowControlIn) {
	this.flowControlIn = stringToFlow(flowControlIn);
}

/**
Busca o valor do Controle de Fluxo para leitura como <code>int</code>.
@return Valor atual do controle de fluxo.
*/
public int getFlowControlIn() {
	return flowControlIn;
}

/**
Busca o valor do Controle de Fluxo para leitura como <code>String</code>.
@return Valor atual do controle de fluxo.
*/
public String getFlowControlInString() {
	return flowToString(flowControlIn);
}

/**
Configura o fluxo de controle para escrita.
```

```
@param flowControlIn Novo controle de fluxo para escrita.
*/
public void setFlowControlOut(int flowControlOut) {
	this.flowControlOut = flowControlOut;
}

/**
Configura o fluxo de controle para escrita.
@param flowControlIn Novo controle de fluxo para escrita.
*/
public void setFlowControlOut(String flowControlOut) {
	this.flowControlOut = stringToFlow(flowControlOut);
}

/**
Busca o valor do controle de fluxo para escrita como <code>int</code>.
@return Valor atual do controle de fluxo.
*/
public int getFlowControlOut() {
	return flowControlOut;
}

/**
Busca o valor do controle de fluxo para escrita como <code>String</code>.
@return Valor atual do controle de fluxo.
*/
public String getFlowControlOutString() {
	return flowToString(flowControlOut);
}

/**
Configura o Bits de Dados.
@param databits Novo Bits de Dados.
*/
public void setDatabits(int databits) {
	this.databits = databits;
}

/**
Configura o Bits de Dados.
@param databits Novo Bits de Dados.
*/
public void setDatabits(String databits) {
	if (databits.equals("5")) {
	   this.databits = SerialPort.DATABITS_5;
	}
	if (databits.equals("6")) {
	   this.databits = SerialPort.DATABITS_6;
	}
	if (databits.equals("7")) {
	   this.databits = SerialPort.DATABITS_7;
	}
	if (databits.equals("8")) {
	   this.databits = SerialPort.DATABITS_8;
	}
```

```
}

/**
Busca valor do Bits de Dados como <code>int</code>.
@return Valor atual para o Bits de Dados.
*/
public int getDatabits() {
	return databits;
}

/**
Busca valor do Bits de Dados como <code>String</code>.
@return Valor atual para o Bits de Dados.
*/
public String getDatabitsString() {
	switch(databits) {
	   case SerialPort.DATABITS_5:
		return "5";
	   case SerialPort.DATABITS_6:
		return "6";
	   case SerialPort.DATABITS_7:
		return "7";
	   case SerialPort.DATABITS_8:
		return "8";
	   default:
		return "8";
	}
}

/**
Configura Bits de Parada.
@param stopbits Novo Bits de Parada.
*/
public void setStopbits(int stopbits) {
	this.stopbits = stopbits;
}

/**
Configura Bits de Parada.
@param stopbits Novo Bits de Parada.
*/
public void setStopbits(String stopbits) {
		if (stopbits.equals("1")) {
		   this.stopbits = SerialPort.STOPBITS_1;
		}
		if (stopbits.equals("1.5")) {
		   this.stopbits = SerialPort.STOPBITS_1_5;
		}
		if (stopbits.equals("2")) {
		   this.stopbits = SerialPort.STOPBITS_2;
		}
}

/**
Busca o valor de Bits de Parada como <code>int</code>.
```

```
@return Atual valor de Bits de Parada.
*/
public int getStopbits() {
	return stopbits;
}

/**
Busca o valor de Bits de Parada como <code>String</code>.
@return Atual valor de Bits de Parada.
*/
public String getStopbitsString() {
	switch(stopbits) {
	   case SerialPort.STOPBITS_1:
		return "1";
	   case SerialPort.STOPBITS_1_5:
		return "1.5";
	   case SerialPort.STOPBITS_2:
		return "2";
	   default:
		return "1";
	}
}

/**
Configura Paridade.
@param parity Nova Paridade.
*/
public void setParity(int parity) {
	this.parity = parity;
}

/**
Configura Paridade.
@param parity Nova Paridade.
*/
public void setParity(String parity) {
	if (parity.equals("Nenhuma")) {
	   this.parity = SerialPort.PARITY_NONE;
	}
	if (parity.equals("Uniforme")) {
	   this.parity = SerialPort.PARITY_EVEN;
	}
	if (parity.equals("Impar")) {
	   this.parity = SerialPort.PARITY_ODD;
	}
}

/**
Busca o valor de paridade como <code>int</code>.
@return Valor atual da Paridade.
*/
public int getParity() {
	return parity;
}
```

```
/**
Busca o valor de paridade como <code>String</code>.
@return Valor atual da Paridade.
*/
public String getParityString() {
	switch(parity) {
	   case SerialPort.PARITY_NONE:
		return "Nenhuma";
	   case SerialPort.PARITY_EVEN:
		return "Uniforme";
	   case SerialPort.PARITY_ODD:
		return "Impar";
	   default:
		return "Nenhuma";
	}
}

/**
Converte uma <code>String</code> descrevendo um controle de fluxo para um tipo
<code>int</code> definido na classe <code>SerialPort</code>.
@param flowControl Uma <code>string</code> descrevendo um controle de fluxo.
@return Um tipo <code>int</code> descrevendo um controle de fluxo.
*/
private int stringToFlow(String flowControl) {
	if (flowControl.equals("Nenhum")) {
	   return SerialPort.FLOWCONTROL_NONE;
	}
	if (flowControl.equals("Xon/Xoff Out")) {
	   return SerialPort.FLOWCONTROL_XONXOFF_OUT;
	}
	if (flowControl.equals("Xon/Xoff In")) {
	   return SerialPort.FLOWCONTROL_XONXOFF_IN;
	}
	if (flowControl.equals("RTS/CTS In")) {
	   return SerialPort.FLOWCONTROL_RTSCTS_IN;
	}
	if (flowControl.equals("RTS/CTS Out")) {
	   return SerialPort.FLOWCONTROL_RTSCTS_OUT;
	}
	return SerialPort.FLOWCONTROL_NONE;
}

/**
Converte um tipo <code>int</code> descrevendo um controle de fluxo para uma
<code>String</code> descrevendo um controle de fluxo.
@param flowControl Um <code>int</code> descrevendo um controle de fluxo.
@return Uma <code>String</code> descrevendo um controle de fluxo.
*/
String flowToString(int flowControl) {
	switch(flowControl) {
	   case SerialPort.FLOWCONTROL_NONE:
		return "Nenhum";
	   case SerialPort.FLOWCONTROL_XONXOFF_OUT:
		return "Xon/Xoff Out";
	   case SerialPort.FLOWCONTROL_XONXOFF_IN:
```

```
                return "Xon/Xoff In";
            case SerialPort.FLOWCONTROL_RTSCTS_IN:
                return "RTS/CTS In";
            case SerialPort.FLOWCONTROL_RTSCTS_OUT:
                return "RTS/CTS Out";
            default:
                return "Nenhum";
        }
    }
}
```

- **SERIALCONNECTIONEXCEPTION.JAVA**

Classe que trata erros quando houver uma tentativa na conexão a uma porta serial.

CÓDIGO-FONTE

```
public class SerialConnectionException extends Exception {

    /**
     * Constroi um <code>SerialConnectionException</code>
     * com a mensagem específica.
     *
     * @param   s   A mensagem detalhada.
     */
    public SerialConnectionException(String str) {
        super(str);
    }

    /**
     * Constroi um <code>SerialConnectionException</code>
     * sem a mensagem.
     */
    public SerialConnectionException() {
        super();
    }
}
```

- **SERIALCONNECTION.JAVA**

Estabelece uma conexão a uma porta serial quando está sendo exibida a tela de terminal (TerminalSerial.java). Trata eventos do teclado quando qualquer tecla é pressionada dentro dos campos “TextArea” de entrada de dados.

Permite mostrar o retorno fornecido pela programação C do PIC no “TextArea” de saída de dados.

CÓDIGO-FONTE

```
import javax.comm.*;
import java.io.*;
import java.awt.TextArea;
import java.awt.event.*;
import java.util.TooManyListenersException;

/**
Classe que conecta o terminal a uma porta serial.
Lê uma entrada de um TextArea e escreve em um outro TextArea.
Mantém o estado de uma conexão.
*/

public class SerialConnection implements SerialPortEventListener,
                                         CommPortOwnershipListener {
   private TerminalSerial parent;

   private TextArea messageAreaOut;
   private TextArea messageAreaIn;
   private SerialParameters parameters;
   private OutputStream os;
   private InputStream is;
   private KeyHandler keyHandler;

   private CommPortIdentifier portId;
   private SerialPort sPort;

   private boolean open;

   /**
   Cria um objeto SerialConnection object e inicializa uma variável passada como parametro.

   @param parent Um objeto do tipo TerminalSerial.
   @param parameters O objeto SerialParameters.
   @param messageAreaOut O TextArea de saída (chega da serial).
   @param messageAreaIn O TextArea de entrada (entra na serial).
   */
   public SerialConnection(TerminalSerial parent,
                           SerialParameters parameters,
                           TextArea messageAreaOut,
                           TextArea messageAreaIn) {
         this.parent = parent;
         this.parameters = parameters;
         this.messageAreaOut = messageAreaOut;
         this.messageAreaIn = messageAreaIn;
         open = false;
   }

   /**
   Tenta conectar-se a uma porta serial
   Se não conseguir a conexão, retorna a porta a um estado de "fechada" (possibilita reconectar-se
novamente)
   e retorna um exceção.
   Dá um tempo de 30 segundos para a conexão, permitindo que outras aplicações
```

```
liberem a porta caso não estejam mais utilizando-as.
*/
public void openConnection() throws SerialConnectionException {

        // Obtém a porta que deseja conectar-se.
        try {
           portId =
                   CommPortIdentifier.getPortIdentifier(parameters.getPortName());
        } catch (NoSuchPortException e) {
           throw new SerialConnectionException(e.getMessage());
        }

        // Abre a porta representada pelo Objeto CommPortIdentifier.
        //Dá um tempo de 30 segundos para a conexão, permitindo que outras aplicações
        //liberem a porta caso não estejam mais utilizando-as.
        try {
           sPort = (SerialPort)portId.open("SerialDemo", 30000);
        } catch (PortInUseException e) {
           throw new SerialConnectionException(e.getMessage());
        }

        // Configura os parametros da conexão
        // Fecha a porta caso ocorra erro.
        try {
           setConnectionParameters();
        } catch (SerialConnectionException e) {
           sPort.close();
           throw e;
        }

        // Abre os dados de entrada e saída
        // se elas nao abrirem, fecha a porta antes de lançar uma exceção.
        try {
           os = sPort.getOutputStream();
           is = sPort.getInputStream();
        } catch (IOException e) {
           sPort.close();
           throw new SerialConnectionException("Error opening i/o streams");
        }

        // Cria um KeyHandler para responder cada tecla pressionada no TextArea de Saida
        keyHandler = new KeyHandler(os);
        messageAreaOut.addKeyListener(keyHandler);

        // Adiciona um listener para a porta serial.
        try {
           sPort.addEventListener(this);
        } catch (TooManyListenersException e) {
           sPort.close();
           throw new SerialConnectionException("too many listeners added");
        }

        // Abre um notifyOnDataAvailable para permitir eventos de entrada de dados.
        sPort.notifyOnDataAvailable(true);
```

```
        // Abre um notifyOnBreakInterrup para permitir o suporte de parada da porta.
        sPort.notifyOnBreakInterrupt(true);

        // Configura um timeout para receber dados da serial durante a entrada de dados.

        try {
           sPort.enableReceiveTimeout(30);
        } catch (UnsupportedCommOperationException e) {
        }

        // Dá o acesso proprietário a este usuário à porta.
        portId.addPortOwnershipListener(this);

        open = true;
}

/**
Configura os parametros da conexão nos parametros do objeto.
Se ocorrer problema, retorna aos valores padrões e lança uma exceção.
*/
public void setConnectionParameters() throws SerialConnectionException {

        // Salva o estado dos parametros antes de configurar novos parametros.
        int oldBaudRate = sPort.getBaudRate();
        int oldDatabits = sPort.getDataBits();
        int oldStopbits = sPort.getStopBits();
        int oldParity   = sPort.getParity();
        int oldFlowControl = sPort.getFlowControlMode();

        // Cconfigura os novos parametros. Em caso de erro, retorna às configurações anteriores.
        try {
           sPort.setSerialPortParams(parameters.getBaudRate(),
                                     parameters.getDatabits(),
                                     parameters.getStopbits(),
                                     parameters.getParity());
        } catch (UnsupportedCommOperationException e) {
           parameters.setBaudRate(oldBaudRate);
           parameters.setDatabits(oldDatabits);
           parameters.setStopbits(oldStopbits);
           parameters.setParity(oldParity);
           throw new SerialConnectionException("Unsupported parameter");
        }

        // Configura o controle de fluxo.
        try {
           sPort.setFlowControlMode(parameters.getFlowControlIn()
                                  | parameters.getFlowControlOut());
        } catch (UnsupportedCommOperationException e) {
           throw new SerialConnectionException("Unsupported flow control");
        }
}

/**
Fecha a porta e limpa elementos associados à conexão.
*/
```

```
    public void closeConnection() {
        // Se a porta já está fechada, apenas retorna ao método que a chamou.
        if (!open) {
            return;
        }

        // Remove listener.
        messageAreaOut.removeKeyListener(keyHandler);

        // Limpa objeto da conexão.
        if (sPort != null) {
            try {
                // Fecha a entrada e saída de dados.
                os.close();
                is.close();
            } catch (IOException e) {
                System.err.println(e);
            }

            // Fecha a porta.
            sPort.close();

            // remove o acesso proprietário à porta.
            portId.removePortOwnershipListener(this);
        }

        open = false;
    }

    /**
    Envia um sinal de parada.
    */
    public void sendBreak() {
        sPort.sendBreak(1000);
    }

    /**
    Retorna o estado da porta. Está aberta?
    @return true Se a porta está aberta retorna TRUE, se a porta está fechada retorna FALSE.
    */
    public boolean isOpen() {
        return open;
    }

    /**
Eventos do envio e recebimento de dados. Os dois tipo de SerialPortEvents que este
programa faz é ouvir um DATA_AVAILABLE e um BI. Durante o DATA_AVAILABLE a buffer de
porta é lido até que  buffer is read until it is não haja mais nada, quando não houver mais dados
disponíveis e 30 ms passaram-se, o método retorna. Quando um evento BI ocorre, as palavras "Parada
Recebida" são escritas ao messageAreaIn.
    */

    public void serialEvent(SerialPortEvent e) {
        // Cria um StringBuffer e um inteiro para receber dados.
        StringBuffer inputBuffer = new StringBuffer();
```

```
        int newData = 0;

        // Determina o tipo de evento.
        switch (e.getEventType()) {

           // Lê dados até que receba -1. Se \r é recebido, substitui-se
           // por \n para a correta formatação de nova linha.
           case SerialPortEvent.DATA_AVAILABLE:
                 while (newData != -1) {
                       try {
                          newData = is.read();
                          if (newData == -1) {
                                break;
                          }
                          if ('\r' == (char)newData) {
                                inputBuffer.append('\n');
                          } else {
                                inputBuffer.append((char)newData);
                          }
                       } catch (IOException ex) {
                          System.err.println(ex);
                          return;
                       }
                 }

                 // Insere os dados recebido ao messageAreaIn.
                 messageAreaIn.append(new String(inputBuffer));
                 break;

           // Se um evento de parada for recebido, envia mensagem .
           case SerialPortEvent.BI:
                 messageAreaIn.append("\n--- PARADA RECEBIDA ---\n");
        }

    }

    /**
Eventos de exclusividade da porta. Se um evento de PORT_OWNERSHIP_REQUESTED é recebido,
uma caixa de diálogo é criado perguntando ao usuário se deseja cancelar.
    */
    public void ownershipChange(int type) {
        if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED) {
           PortRequestedDialog prd = new PortRequestedDialog(parent);
        }
    }

    /**
    Classe que trata eventos de pressionamento de teclas
    Quando um char é recebido, o mesmo é lido, convertido
    para um inteiro e escrito para o <code>OutputStream</code> à porta.
    */
    class KeyHandler extends KeyAdapter {
        OutputStream os;

        /**
```

```
        Cria o KeyHandler.
        @param os O OutputStream para a porta.
        */
        public KeyHandler(OutputStream os) {
           super();
           this.os = os;
        }

        /**
        Suporte ao KeyEvent.
        */
     public void keyTyped(KeyEvent evt) {
        char newCharacter = evt.getKeyChar();
           try {
                 os.write((int)newCharacter);
           } catch (IOException e) {
                 System.err.println("Ocorreu erro ao enviar dados a porta: " + e);
           }
     }
  }
}
```

- **PORTREQUESTDIALOG.JAVA**

Informa ao usuário que há outra aplicação requisitando a porta usada, e então questiona o usuário se deseja selecionar outra porta. Se for selecionado “Sim”, a porta é fechada e a caixa de diálogo também. Se for selecionado “Não”, a caixa de diálogo é fechada e nenhuma outra ação acontece.

Esta classe é utilizada pelo Terminal quando ocorre erro na utilização da porta e é necessário informar o usuário deste erro.

CÓDIGO-FONTE

```
import java.awt.*;
import java.awt.event.*;

/**
Informa ao usuário que existe um outro aplicativo utilizando a porta serial.
Se resposta "SIM" então a porta é fechada e o diálogo fecha.
Se resposta "NÃO" então o diálogo fecha.
*/
public class PortRequestedDialog extends Dialog implements ActionListener {

   private TerminalSerial parent;

   /**
   Cria um Diálogo de mensagem com dois botões e uma mensagem questionando o usuário
```

```
    se ele deseja selecionar uma outra porta.

    @param parent Tela principal que a chama.
    */
    public PortRequestedDialog(TerminalSerial parent) {
        super(parent, "Port Requested!", true);
        this.parent = parent;

        String lineOne = "A porta seleciona já está sendo";
        String lineTwo = "utilizada por uma outra aplicação.";
        String lineThree = "Você deseja cancelar?";
        Panel labelPanel = new Panel();
        labelPanel.setLayout(new GridLayout(3, 1));
        labelPanel.add(new Label(lineOne, Label.CENTER));
        labelPanel.add(new Label(lineTwo, Label.CENTER));
        labelPanel.add(new Label(lineThree, Label.CENTER));
        add(labelPanel, "Center");

        Panel buttonPanel = new Panel();
        Button yesButton = new Button("Sim");
        yesButton.addActionListener(this);
        buttonPanel.add(yesButton);
        Button noButton = new Button("Não");
        noButton.addActionListener(this);
        buttonPanel.add(noButton);
        add(buttonPanel, "South");

        FontMetrics fm = getFontMetrics(getFont());
        int width = Math.max(fm.stringWidth(lineOne),
                Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));

        setSize(width + 40, 150);
        setLocation(parent.getLocationOnScreen().x + 30,
                parent.getLocationOnScreen().y + 30);
        setVisible(true);
    }

    /**
    Eventos dos botões
    */
    public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();

          if (cmd.equals("Yes")) {
              parent.portClosed();
          }

          setVisible(false);
          dispose();
    }
}
```

- **ALERTDIALOG.JAVA**

Classe padrão para utilizar como caixa de mensagens dentro do aplicativo. Necessário passar como parâmetro o texto do diálogo e o título da mensagem. Possui um botão “OK” para retornar a classe a quem a chamou.

É utilizado dentro da Classe responsável pelas funcionalidades do Terminal (TerminalSerial.java).

CÓDIGO-FONTE

```
import java.awt.*;
import java.awt.event.*;

/**
Mostra uma tela. A classe recebe uma mensagem e um título.
Pressiona-se o botão OK para sair da tela de aviso.
*/
public class AlertDialog extends Dialog implements ActionListener {

   /**

   @param parent Qualquer frame.
   @param title O título da tela.
   @param lineOne Primeira linha da mensagem.
   @param lineTwo Segunda linha da mensagem.
   @param lineThree Terceira linha da mensagem.
   */
   public AlertDialog(Frame parent,
                      String title,
                      String lineOne,
                      String lineTwo,
                      String lineThree) {
         super(parent, title, true);

         Panel labelPanel = new Panel();
         labelPanel.setLayout(new GridLayout(3, 1));
         labelPanel.add(new Label(lineOne, Label.CENTER));
         labelPanel.add(new Label(lineTwo, Label.CENTER));
         labelPanel.add(new Label(lineThree, Label.CENTER));
         add(labelPanel, "Center");

         Panel buttonPanel = new Panel();
         Button okButton = new Button("OK");
         okButton.addActionListener(this);
         buttonPanel.add(okButton);
         add(buttonPanel, "South");

         FontMetrics fm = getFontMetrics(getFont());
         int width = Math.max(fm.stringWidth(lineOne),
                  Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));
```

```
            setSize(width + 40, 150);
            setLocation(parent.getLocationOnScreen().x + 30,
                        parent.getLocationOnScreen().y + 30);
            setVisible(true);
        }

        /**
        Evento do botão OK
        */
        public void actionPerformed(ActionEvent e) {
            setVisible(false);
            dispose();
        }
    }
```