Centro Universitário de Brasília - UniCEUB
Faculdade de Ciências Exatas e Tecnologia - FAET
Curso de Engenharia da Computação
Projeto Final

AUTOMAÇÃO RESIDENCIAL POR COMANDO DE VOZ UTILIZANDO MICROCONTROLADOR

Por

Gustavo Gomes de Lucena

RA: 2011484-5

Professora Orientadora:

Prof[a]. M.C. Maria Marony Sousa Farias Nascimento

Brasília – DF

Dezembro de 2006

GUSTAVO GOMES DE LUCENA

AUTOMAÇÃO RESIDENCIAL POR COMANDO DE VOZ UTILIZANDO MICROCONTROLADOR

Monografia apresentada à banca examinadora para conclusão do curso e obtenção do título de bacharel em Engenharia da Computação do Centro Universitário de Brasília - UniCEUB.

Brasília – DF

Dezembro de 2006

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus pelo dom da vida e por todas as bênçãos derramadas.

Agradeço muito à minha família, por todo tipo de apoio, por acreditarem em mim e por entenderem minha ausência em diversos momentos durante esses anos. Em especial, agradeço aos meus pais, aos meus irmãos e à minha pequena Flavinha.

Aos amigos de longa data e aos novos amigos que conquistei durante o curso: vocês também fazem parte dessa vitória! Muito obrigado aos amigos que me deram uma força durante o desenvolvimento do projeto, Fabrício, Saulo, Ítalo, Glauber, Emmanuel e todos com quem compartilhei dúvidas.

Obrigado a todos os professores e funcionários do UniCEUB, principalmente ao professor coordenador Abiézer e à professora orientadora Maria Marony.

“Na natureza, nada se perde,

nada se cria, tudo se transforma!”

Lei da Conservação da Massa de Lavoisier

RESUMO

Visando beneficiar usuários com dificuldades de locomoção e usuários que buscam sofisticação e conforto, foi desenvolvida uma solução de automação residencial através do comando de voz. Neste projeto, o usuário controla o estado de uma lâmpada através de dois comandos: “aceso” e “apagado”. No computador, o Simulador de Entrada de Dados recebe o comando informado e transmite através da interface serial para o microcontrolador, que é responsável pelo acionamento do circuito elétrico. Esta monografia tem como objetivo documentar as atividades realizadas e o conhecimento teórico relacionado.

Palavras-chave: Automação Residencial, Comando de Voz, Microcontrolador, Linguagem de Programação Java, Linguagem de Programação C.

ABSTRACT

Aiming for benefit to users with locomotion difficulties and users who search sophistication and comfort, was developed a solution of residential automation through voice command. In this project, the user controls the state of a light bulb by two commands: “aceso” and “apagado”. In the computer, the Data Entry Simulator receives the command informed and transmits through the serial interface to the microcontroller, that is responsible for the drive of the electric circuit. This monograph has the objective of register the activities carried through and the related theoretical knowledge.

Key-words: Residential Automation, Voice Command, Microcontroller, Programming Language Java, Programming Language C.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES E SIGLAS

βcc	Ganho de corrente.
API	Applications Programming Interface (interface de programa aplicativo).
Aureside	Associação Brasileira de Automação Residencial.
CFTV	Circuito fechado de televisão.
COM1	Porta serial utilizada no projeto.
CW552	Kit de desenvolvimento da ControlWare Automação.
dB	Decibel. Unidade de medida do nível de intensidade do som.
DB-9	Conector serial, padrão físico RS-232, com nove pinos.
DB-25	Conector serial, padrão físico RS-232, com 25 pinos.
EPROM	Electrically Programmable Read-Only Memory (memória somente de leitura programável eletronicamente).
IDE	Integrated Development Environment (ambiente de desenvolvimento integrado).
Mbps	*Megabits* por segundo.
RAM	Random Access Memory (memória de acesso aleatório). Memória volátil, ou seja, perde os dados quando o computador é desligado.
RS-232	Padrão físico da interface serial que especifica a quantidade de conectores e a tensão nos pinos.
SDCC	Small Device C Compiler (compilador C para pequenos dispositivos).
SDK	Software Development Kit (kit de desenvolvimento de software).
UART	Universal Asynchronous Receiver and Transmitter (transmissor e receptor assíncrono universal).
USB	Universal Serial Bus (barramento serial universal).

LISTA DE TRECHOS DE CÓDIGO

CAPÍTULO 1. INTRODUÇÃO

Neste projeto foi desenvolvida uma solução de automação residencial através do comando de voz utilizando um microcontrolador que faz o elo entre o computador e a interface elétrica. Para isso, foram desenvolvidos softwares para o computador – responsável pela interface com o usuário – e para o microcontrolador – responsável por controlar o circuito de acionamento elétrico.

No Capítulo 1 é feita uma introdução ao projeto, onde são apresentados os objetivos e motivação. Na seção 1.1 são apresentadas as motivações do autor para o desenvolvimento do projeto. Na seção 1.2 é mostrada uma visão geral do projeto e a forma de organização da monografia.

1.1 Motivação

O fato de ainda não ter sido desenvolvido nenhum projeto de automação residencial através de comando de voz no UniCEUB e a crescente popularização do tema, foram as principais motivações para o desenvolvimento deste projeto. Todo o projeto foi desenvolvido buscando originalidade nas soluções apresentadas e no método de implementação.

1.1.1 Objetivos gerais

O objetivo deste projeto é proporcionar uma melhoria para a sociedade através do desenvolvimento de um tema voltado para a área de automação e controle. A melhoria ocorre através da praticidade proporcionada pelo comando de voz que controla a iluminação de uma residência.

1.1.2 Objetivos específicos

Os projetos de controle de equipamentos através de comando de voz estão em constante evolução devido, principalmente, ao aumento na capacidade de processamento dos computadores. Conforme dito anteriormente, este projeto tem o objetivo de proporcionar uma melhoria para a sociedade, mais especificamente para usuários com dificuldades de locomoção. Dessa forma, o usuário poderá controlar seus equipamentos sem a necessidade de se locomover ou solicitar a terceiros. Os benefícios advindos da automação residencial através de comandos de voz se estendem a todos os usuários, e não somente aos portadores de dificuldades de locomoção.

1.2 Visão geral do projeto

A solução desenvolvida é mostrada na Figura 1.1:

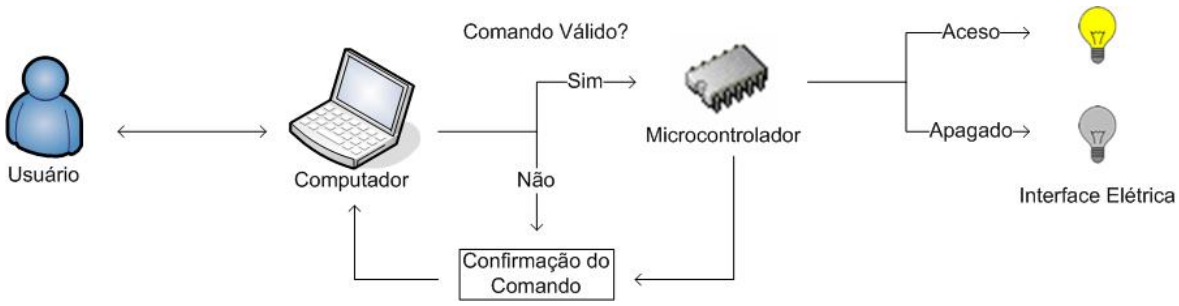

Figura 1.1 Diagrama geral do projeto.

O usuário informa o estado da iluminação (aceso ou apagado) no microfone. O software de reconhecimento de voz traduz a informação, preenchendo um campo específico no Simulador de Entrada de Dados.

Através do Simulador, o usuário recebe a confirmação dos estados solicitados, além de outras mensagens informativas. Se o estado solicitado for válido, é transmitido um comando através da interface serial do computador para o microcontrolador.

O microcontrolador é responsável pelo controle da iluminação através do comando solicitado. Caso o comando solicitado seja para apagar a iluminação, a tensão 0V será enviada para a interface elétrica. De forma semelhante, caso o comando solicitado seja

para acender a iluminação, será enviada a tensão 5V. A tensão de 5V fecha o circuito elétrico, permitindo a passagem de corrente e, conseqüentemente, o acendimento da lâmpada.

Cada uma dessas etapas será detalhada nos capítulos da monografia, conforme divisão abaixo:

- Capítulo 1: Introdução, com a motivação para a escolha do tema, os objetivos e visão geral do projeto.
- Capítulo 2: Referencial teórico sobre automação residencial e comando de voz.
- Capítulo 3: São mostrados o hardware e as interfaces utilizadas e desenvolvidas.
- Capítulo 4: Neste capítulo é apresentado o Simulador de Entrada de Dados. É mostrada a finalidade do Simulador, seu funcionamento e alguns trechos de código mais importantes.
- Capítulo 5: É apresentado o software desenvolvido para o microcontrolador, com a função ser o elo entre o computador e a interface elétrica. Também são apresentados alguns trechos de código mais importantes.
- Capítulo 6: Neste capítulo são apresentados os dados coletados durante a avaliação do desempenho e os resultados obtidos.
- Capítulo 7: São apresentadas as considerações finais do projeto, com as conclusões, dificuldades encontradas e sugestões para projetos futuros.
- Apêndices: São apresentados os códigos-fonte completos dos softwares do computador e do microcontrolador, além de uma fotografia do protótipo.

CAPÍTULO 2. REFERENCIAL TEÓRICO

A automação residencial através de comando de voz une duas vertentes que estão em constante crescimento no mercado brasileiro. Conforme já é previsto pelos projetistas de casas inteligentes, o conforto e a praticidade proporcionados por essa tecnologia serão, em um futuro próximo, indispensáveis nas residências (Aureside, 2006).

No Capítulo 2 é apresentada uma introdução teórica sobre os principais assuntos do projeto. Na seção 2.1 é abordada a automação residencial no Brasil com algumas características. Na seção 2.2 é apresentada a tecnologia de reconhecimento de voz e suas peculiaridades.

2.1 Automação Residencial

A automação residencial surgiu depois de seus similares nas áreas industrial e comercial por motivos de escala de produção e econômicos, pois essas áreas propiciariam rapidez no retorno dos investimentos. No Brasil, os primeiros sistemas industriais automatizados surgiram na década de setenta. Após consolidada a automação industrial, o comércio entrou na era automatizada e os avanços da informática propiciaram inovações constantes (Aureside, 2006).

Diferentemente da automação comercial, que deve ser mais generalizada para atender aos diversos tipos de clientes, as soluções de automação residencial devem ser pessoais, tornando-se soluções personalizadas de acordo com a realidade da residência. Por exemplo, um usuário mora em um local isolado que necessita reforço na segurança. Por isso, deseja a instalação de um CFTV (Circuito Fechado de Televisão). Um outro exemplo seria um usuário deficiente físico que tem dificuldade de locomoção e mora em um condomínio de luxo fechado bastante seguro. Nesse caso, a necessidade maior do usuário não é a segurança, mas sim a possibilidade de controlar seus equipamentos por comando de voz.

Existem diversas aplicações para a automação residencial, dependendo apenas da necessidade do morador e da criatividade do desenvolvedor. Monitoração da residência via Internet, monitoração através de CFTV, rede doméstica de computador, controle da climatização, da iluminação e de eletrodomésticos são algumas das soluções mais comuns.

Independente da solução de automação ser para a área residencial, comercial ou industrial, existem algumas características que devem ser levadas em consideração (Aureside, 2006):

- Relação custo-benefício;
- confiabilidade, ou seja, certeza de que a solução aplicada funciona nas condições locais de ruídos externos, baixa iluminação, dentre outros fatores;
- interatividade com o usuário, facilitando o uso da solução;
- atualização tecnológica simples, permitindo *upgrades*;
- conforto e conveniência;
- possibilidade de integração de diferentes soluções, simplificando a operação, aumentando a economia e a segurança.

Existem inúmeros projetos de casas inteligentes utilizando tudo o que é mais moderno na área de automação residencial. No artigo disponível no endereço *http://www.aureside.org.br/temastec/default.asp?file=concbasicos03.asp* é descrita a casa conceito desenvolvida pela Microsoft.

2.1.1 Comparação entre automação residencial e aparelhos celulares

Comparando a automação residencial com o surgimento dos aparelhos telefônicos celulares, em um primeiro momento, ambos são símbolos de status e de modernidade. Em seguida, o conforto e a praticidade proporcionados são fundamentais na decisão da aquisição dessas tecnologias. Depois de algum tempo, tanto os aparelhos celulares como a automação residencial tornam-se fatores de economia e de necessidade vital. Aparentemente pode parecer que avaliar a automação residencial como uma necessidade

vital é um exagero, mas se for levado em consideração que o morador dessa residência é um deficiente físico com dificuldades de locomoção ou um deficiente visual, a possibilidade de controlar os equipamentos através do comando de voz oferece um aumento significativo na qualidade de vida dessas pessoas.

2.2 Reconhecimento de Voz

Existem dois parâmetros que classificam os sistemas de reconhecimento de voz. O primeiro parâmetro é a dependência ou não com relação ao falante. O segundo parâmetro é a forma da fala, discreta ou contínua (Jerome, 1994).

Um sistema independente reconhece a voz de qualquer usuário e não requer treinamento. Os sistemas dependentes são utilizados para reconhecimento de um número maior de vocabulários previamente treinados pelo usuário. Os sistemas contínuos reconhecem a voz natural, enquanto nos sistemas discretos é necessário que as palavras sejam ditas pausadamente.

As primeiras soluções de automação residencial através de comandos de voz eram inovadoras e muito interessantes, mas faltava confiabilidade e desempenho para que fossem um método viável de controle. Com a evolução da informática, houve um aumento na capacidade de processamento associado à redução de custos, possibilitando dar continuidade aos projetos de automação residencial por comando de voz. Segundo a Associação Brasileira de Automação Residencial – Aureside – testes mostram que esses projetos funcionam razoavelmente bem, mas é necessário que o microfone esteja próximo ao usuário para permitir um reconhecimento confiável. A maioria dos produtos requer treinamento por parte do usuário para criação do padrão de voz e, ainda assim, os resultados não são confiáveis quando submetidos a ruídos de ambiente, como barulho de aparelho condicionador de ar ou o ruído emitido por pessoas conversando.

Existem alguns critérios que devem ser observados durante o desenvolvimento de um projeto de automação que utilize comandos de voz (Aureside, 2006):

- Confiabilidade;
- possibilidade de operação em ambientes ruidosos e silenciosos;
- não deve ser necessário que o usuário utilize nenhum tipo adicional de hardware;
- uso de microfones distribuídos pela residência que captam todo tipo de som e reconhece os comandos de voz;
- funcionamento paralelo com outros tipos de hardware, como interruptores e painéis de controle;
- confirmação do comando recebido com opção de emissão sonora.

Capítulo 3. HARDWARE E INTERFACES

Neste projeto existem três interfaces[1]. A interface de entrada permite ao usuário informar o comando desejado. É composta por dois softwares que trabalham em conjunto. O microcontrolador e o computador se comunicam através da interface serial. A interface elétrica recebe os *bit*s 0 ou 1 do microcontrolador e aciona o circuito da lâmpada.

Neste capítulo são apresentadas as interfaces do projeto. Na seção 3.1 são apresentados os softwares que compõem a interface de entrada. Na seção 3.2 são tratados os detalhes da interface serial. Na seção 3.3 é apresentada a interface elétrica do projeto.

3.1 Interface de Entrada

Conforme dito anteriormente, a interface de entrada é responsável pelo recebimento dos comandos do usuário, possibilitando o controle do acendimento da lâmpada. Neste projeto é utilizado um computador para receber os comandos de voz do usuário através do Simulador de Entrada de Dados (ver seção 4.3). Em resumo, o usuário pode alterar o estado de uma lâmpada – “aceso” ou “apagado” – através do comando de voz, não havendo necessidade de digitação desses comandos.

Dois softwares trabalham em conjunto para possibilitar a entrada de dados. Nos subitens 3.1.1 e 3.1.2 serão abordados o funcionamento do software de reconhecimento de voz e o funcionamento básico do Simulador, respectivamente.

3.1.1 Reconhecimento de voz - IBM Via Voice

A proposta do projeto é utilizar comando de voz para controlar a iluminação. O desenvolvimento de um sistema capaz de reconhecer padrões de voz está fora do escopo do projeto e é uma sugestão para projetos futuros.

[1] Interface é a superfície que separa duas faces de um sistema. (Novo Dicionário Aurélio da Língua Portuguesa).

Neste projeto, o reconhecimento de voz é feito por um software proprietário da IBM, o Via Voice Pro USB Edition versão 9.0. Esse software permite ao usuário ditar um texto e controlar diversas funções do computador, como abrir e fechar aplicativos.

O IBM Via Voice foi escolhido como o software de reconhecimento de voz devido à alta credibilidade do software entre os usuários em todo o mundo. O funcionamento interno do software não é conhecido. Entretanto, provavelmente são utilizadas redes neurais artificiais para possibilitar o aprendizado. (Neves, 2006).

Com relação ao tipo de sistema de reconhecimento de voz, o IBM Via Voice é dependente e discreto. Para que seja possível o reconhecimento da voz do usuário e para que os erros de interpretação por parte do software sejam minimizados, é necessária a criação de um padrão pessoal de voz, solicitada pelo software na primeira utilização ou na criação de um novo usuário. Na Figura 3.1 é mostrada a tela inicial do software para criação do padrão pessoal de voz.

Figura 3.1 Criação de um padrão pessoal de voz.

A criação do padrão pessoal de voz é feita em duas etapas. Na Figura 3.2 é mostrada a primeira etapa da criação do padrão pessoal de voz, que consiste do processamento de uma gravação curta da voz do usuário.

Figura 3.2 Primeira etapa da criação de um padrão pessoal de voz.

Na Figura 3.3 é mostrada a segunda etapa, onde o usuário lê uma história para que o IBM Via Voice aprenda como o usuário pronuncia as palavras.

Figura 3.3 Segunda etapa da criação de um padrão pessoal de voz.

Depois de concluída a segunda etapa, o IBM Via Voice analisa a gravação e cria o padrão pessoal de voz do usuário.

Ao ativar o IBM Via Voice, tudo o que o usuário disser no microfone será transcrito para o programa que estiver ativo no computador. No projeto, o programa ativo será o Simulador de Entrada de Dados.

Embora a versão utilizada neste projeto não seja gratuita, a IBM disponibilizou em setembro de 2004 sua tecnologia de reconhecimento de voz na forma de código aberto. A disponibilização de seu código-fonte ocorreu após um acordo firmado com os maiores desenvolvedores de aplicativos de reconhecimento de voz, que se comprometeram a aderir ao desenvolvimento aberto e abandonar as soluções proprietárias. Maiores informações no artigo disponível no endereço *http://www-03.ibm.com/press/us/en/pressrelease/7293.wss*.

3.1.1.1 Especificação do microfone

A entrada de dados é feita por um microfone ligado ao computador. O software reconhece a voz do usuário e digita na tela o que for falado no microfone.

Neste projeto, foi utilizado o microfone com pedestal da Goldship, modelo 1521. Esse é um microfone condensado de alta sensibilidade. A Tabela 3.1 é apresentada a especificação técnica desse microfone.

Tabela 3.1 Especificação técnica do microfone utilizado.

Parâmetro	Valor
Consumo máximo de corrente	350 nA
Voltagem padrão de operação	4,5 V
Freqüência de resposta	50 Hz – 13 kHz
Impedância	2200 Ohms +/- 15%
Relação sinal-ruído	40 dB ou mais
Sensibilidade	-58 dB +/- 3 dB
Tamanho da haste	30 cm
Tamanho do cabo	1,80 m +/- 5%
Peso	125 g

Fonte: Atera Informática, 2006.

3.1.2 Simulador de Entrada de Dados

O Simulador de Entrada de Dados trabalha em conjunto com o IBM Via Voice. O Simulador possui um *campo de comandos* com uma periodicidade de leitura automática.

Dessa forma, o Simulador recebe o comando ditado pelo usuário e transmite a informação para o microcontrolador. O funcionamento do Simulador é assunto do Capítulo 4.

3.2 Interface serial

As interfaces seriais estão nos computadores desde a década de oitenta. A principal característica é que podem transmitir ou receber um *bit* de cada vez. Na Tabela 3.2 é mostrado que a taxa de transferência é muito baixa se comparada com a interface USB. (HP, 2006).

Tabela 3.2 Comparação entre as taxas de transferência da interface serial e da interface USB.

Tipo de Conexão	Taxa de Transferência
Interface Serial	0,92 Mbps
USB 1.1	12 Mbps
USB 2.0	12 Mbps
USB 2.0 de alta velocidade	480 Mbps

Fonte: HP, 2006.

Devido à baixa taxa de transmissão, a interface serial geralmente é utilizada por dispositivos que não necessitam de alta velocidade, como mouse e teclado.

Existem diversos tipos de interfaces seriais. A interface serial padrão é assíncrona (UART – Universal Asynchronous Receiver and Transmitter) com meio físico no padrão RS-232. Muitas vezes é feita confusão entre os termos interface serial e interface RS-232. O padrão RS-232 especifica um conector de nove pinos (DB-9) para conexão com dispositivos seriais. Existe também uma versão com 25 pinos (DB-25). Na Figura 3.4 são mostradas as duas versões do conector padrão RS-232.

Figura 3.4 Conectores RS-232.

Além da versão do conector, o padrão RS-232 especifica os níveis de tensão para representar os valores 0 e 1. Geralmente são utilizados respectivamente os valores –12V e +12V, embora qualquer valor entre 6V e 15V possa ser utilizado. (Vasconcelos, 2002).

3.2.1 Formato dos dados transmitidos

Na Figura 3.5 é mostrada a transmissão de dados de uma interface serial padrão.

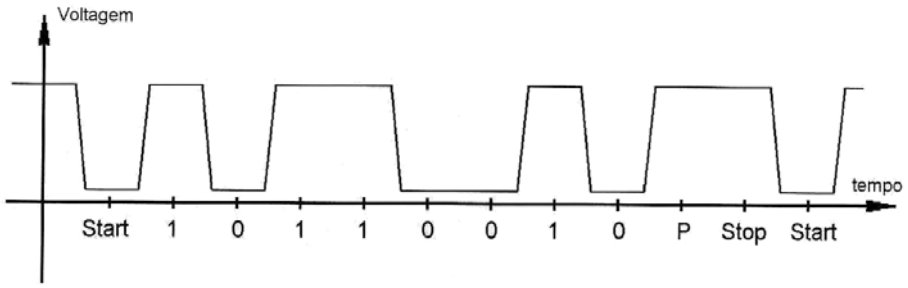

Figura 3.5 Transmissão em uma interface serial padrão.

Fonte: Vasconcelos, 2002.

Quando está em repouso, a interface serial fornece a tensão correspondente ao *bit* 1. O primeiro *bit* 0 – *start bit* – informa que será iniciada uma transmissão. A seguir, o dado é transmitido, um *bit* de cada vez. Podem ser transmitidos de cinco a oito *bits*, dependendo da configuração dos *data bits* na interface serial. Terminados os *bits* de dados, é enviado um *bit* opcional para detecção de erros conhecido como *bit de paridade*. A paridade pode ser par, ímpar ou nenhuma. No exemplo da Figura 3.5, o *bit de paridade* é ímpar, ou seja, a soma dos *bits* de dado com o *bit de paridade* deve ser ímpar. Após o *bit de paridade* vem o *stop bit* e, em seguida, é possível iniciar a transmissão de outro dado. (Vasconcelos, 2002).

Em resumo, a transmissão via interface serial é caracterizada por:

- *start bit* (sempre igual a 0). Informa que será iniciada uma transmissão;
- *data bits* (entre 5 e 8). Configura o número de *bits* que será enviado;
- *paridade* (par, ímpar ou nenhuma). Funciona como um dígito verificador;
- *stop bit* (pode ser 1, 1.5 ou 2). Informa o término da transmissão.

Os três últimos parâmetros definem o formato da transmissão. No projeto, a transmissão é 8N2, ou seja, oito *data bits*, nenhuma paridade e dois *stop bits*.

Os *bits* de dados sempre são transmitidos na ordem inversa, do *bit* menos significativo para o *bit* mais significativo. Portanto, a Figura 3.5 representa a transmissão do dado 01001101, o mesmo que 4D em hexadecimal ou 77 em decimal. (Vasconcelos, 2002).

3.2.2 Baud Rate

É uma medida com dimensão *bits* por segundo, mas não representa exatamente o número de *bits* transmitidos por segundo pela interface. A unidade é o *baud* e o seu valor é o inverso do período de transmissão de um *bit*. Fazendo uma analogia, o *baud rate* equivale à freqüência, assim como o período de transmissão equivale ao período.

O *baud rate* de uma interface serial padrão é gerado a partir da freqüência de um cristal externo e de um divisor interno. O divisor pode ser programado com qualquer número inteiro entre um e seu valor máximo. Dessa forma, é possível operar com diversos valores para o *baud rate*. Na prática, o Microsoft Windows permite o uso de 13 diferentes valores (Vasconcelos, 2002).

Tomando um *baud rate* de 9600 *bauds*, o mesmo valor utilizado no projeto, significa que cada *bit* é transmitido em um tempo de 1 s / 9600 = 1,04167 x 10^{-4} s, ou seja, aproximadamente 104,2 μs.

A taxa de transferência é obtida através da divisão do número de *bauds* pelo número total de *bits* – incluindo o *start bit*, o *bit de paridade* e o *stop bit*. No projeto, cada *byte* está utilizando 11 *bits* – um *start bit,* oito *data bits*, e dois *stop bits.* A taxa de transferência é de 9600 / 11 = 872,73 bytes por segundo (Vasconcelos, 2002).

O termo *baud* é uma homenagem ao engenheiro francês Jean Maurice-Emile Baudot. Ele utilizou essa grandeza para medir a velocidade de transmissão de telégrafos no século 19 (PCCOM, 2006).

3.3 Interface Elétrica

A interface elétrica do projeto é composta por três circuitos relacionados. Sua função é receber o *bit* 0 – comando “apagado” – ou o *bit* 1 – comando “aceso” – enviado pelo microcontrolador.

Na Figura 3.6 é apresentado o circuito completo, composto pelos circuitos do microcontrolador, do relé e da lâmpada.

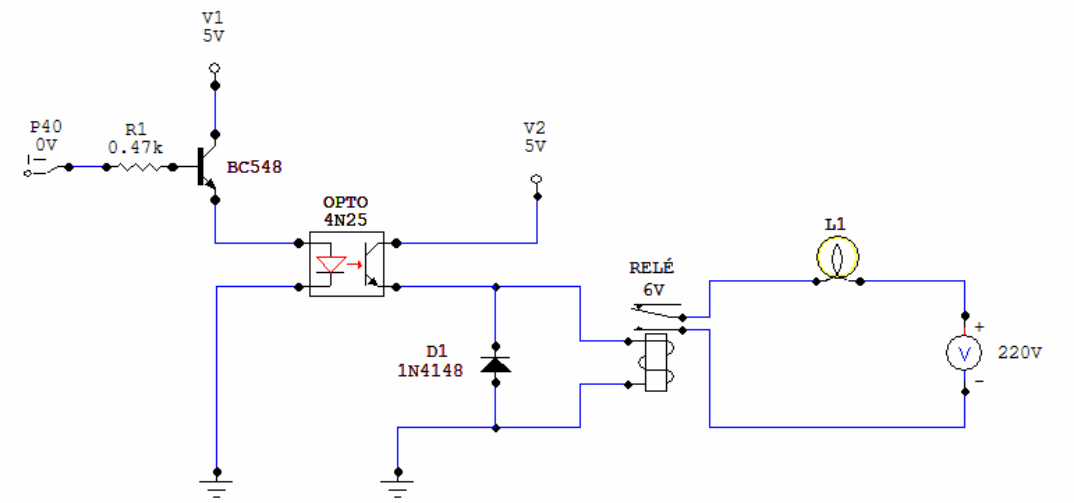

Figura 3.6 Circuito completo de controle da iluminação.

As figuras da seção 3.3 foram feitas utilizando o software Circuit Maker Student Version. Essa é uma versão gratuita para estudantes e possibilita a montagem e a simulação de circuitos elétricos e eletrônicos. É possível encontrar a versão de estudante para download no endereço *http://superdownloads.uol.com.br/download/i9609.htm*.

Nos subitens a seguir serão apresentados os circuitos individuais com suas características.

3.3.1 Circuito do microcontrolador

Na Figura 3.7 é apresentado o circuito do microcontrolador, responsável pela interface entre o microcontrolador e o relé que aciona a lâmpada.

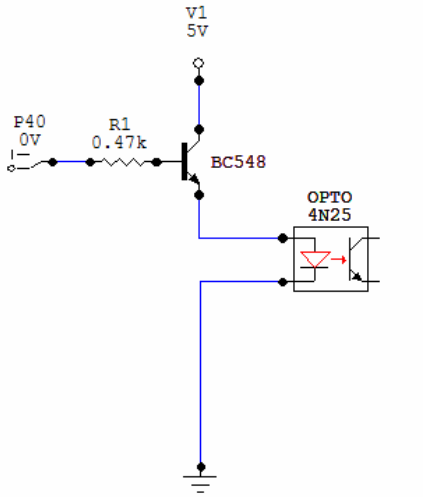

Figura 3.7 Circuito do microcontrolador.

O *bit* 0 ou 1 – 0V ou 5V, respectivamente – é enviado pelo pino P40 do microcontrolador. Enquanto a tensão no pino P40 for 0V, não passa corrente pelo led no interior do optoacoplador 4N25. Quando for enviado o *bit* 1 (ou seja, 5V), o led será acionado e o circuito do relé, representado na Figura 3.9, será fechado.

3.3.1.1 Funcionamento do transistor

Quando os transistores possuem a função de chaveamento, funcionam em corte e em saturação (aberto e fechado, respectivamente). A corrente na base do transistor controla o fechamento do contato emissor-coletor. Se a corrente na base for zero, a corrente no coletor será próxima de zero e o transistor estará em corte. Se a corrente na base for maior ou igual à corrente de saturação, a corrente no coletor será máxima e o transistor estará em saturação. (Bertoli, 2000).

Existem dois tipos de saturação. Saturação fraca significa que o transistor está levemente saturado, ou seja, a corrente na base é suficiente apenas para operar o transistor, não levando em conta as variações de ganho de corrente βcc [2]. Saturação forte significa que existe uma corrente na base suficiente para saturar o transistor para todas as variações

[2] A divisão do valor da corrente no coletor do transistor pelo valor da corrente na base do transistor é denominada variação de ganho de corrente (βcc).

de ganho de corrente βcc. Para garantir uma saturação forte, é necessário que a corrente na base seja um décimo da corrente no coletor (βcc = 10). (Bertoli, 2000).

No projeto, a corrente na base do transistor é aproximadamente 0,52 mA e a corrente no coletor é aproximadamente 2,1 mA, ou seja, o transistor está funcionando abaixo da saturação forte (βcc = 4).

3.3.1.2 Funcionamento do optoacoplador

Os optoacopladores, ou acopladores ópticos, são uma combinação de dispositivos ópticos eletrônicos – um diodo emissor de luz (led) e um fototransistor – montados em um mesmo invólucro. (Ahmed, 2000). A função do optoacoplador é o isolamento elétrico dos circuitos de entrada e de saída. Na Figura 3.8 é mostrado o invólucro (a), o esquema interno (b) e a descrição dos pinos (c) do optoacoplador 4N25, modelo utilizado no projeto.

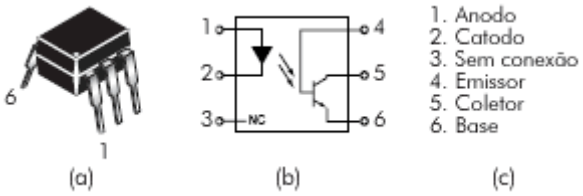

Figura 3.8 Optoacoplador 4N25.

Qualquer variação de tensão produz uma variação na corrente que passa pelo led, acendendo-o. O fototransistor percebe a iluminação produzida pelo led através da variação de corrente na sua base, gerando uma variação de tensão nos terminais coletor-emissor. Dessa forma, um sinal de tensão no circuito de entrada é transmitido para o circuito de saída. (Malvino, 1995).

No projeto, o optoacoplador tem a função de isolar o circuito do microcontrolador e o circuito do relé.

3.3.2 Circuito do relé

Na Figura 3.9 é mostrado o circuito do relé, responsável por fechar o circuito da lâmpada, acendendo-a.

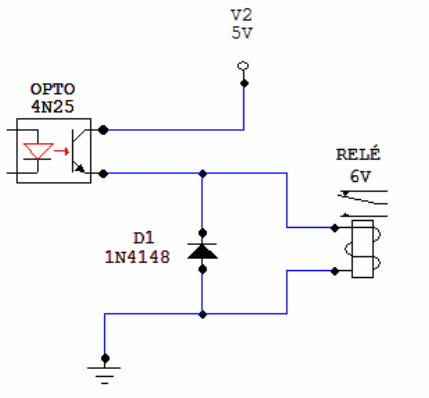

Figura 3.9 Circuito do relé.

Quando o led do optoacoplador é aceso, o fototransistor gera uma variação nos terminais coletor-emissor, fechando o circuito do relé. A corrente no coletor do transistor é 75 mA, valor superior à corrente de saturação para o optoacoplador 4N25 (segundo o *datasheet*, a corrente de saturação do coletor é 20mA). Portanto, o transistor do optoacoplador está funcionando em saturação forte.

Na Figura 3.10 é mostrado o diagrama do relé utilizado.

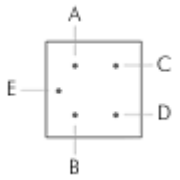

Figura 3.10 Diagrama do relé.

Os pinos A e B estão ligados no circuito do relé. Os pinos C e E estão ligados no circuito da lâmpada. O circuito da lâmpada fica normalmente aberto porque os contatos internos do relé estão nos pinos D e E. Quando o relé é energizado, o contato entre os pinos C e E é induzido magneticamente, fechando o circuito da lâmpada.

3.3.2.1 Função do diodo

O diodo 1N4148 está polarizado inversamente com relação ao relé. Na polarização inversa, os elétrons do material tipo N do diodo (anodo) são atraídos para o terminal positivo do relé, afastando-se da junção, e as lacunas do material tipo P (catodo) são atraídas pelo terminal negativo do relé. Dessa forma, a camada de depleção aumenta,

tornando praticamente impossível o deslocamento de elétrons de uma camada para outra. (Bertoli, 2000).

A função do diodo 1N4148 é proteger o optoacoplador da carga indutiva do relé. Toda vez que o relé é desligado, a energia armazenada é descarregada pelo diodo.

3.3.3 Circuito da lâmpada

Na Figura 3.11 é mostrado o circuito da lâmpada. Esse circuito é alimentado por uma fonte externa, que pode ser de 110V ou 220V.

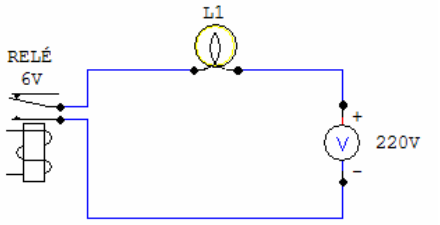

Figura 3.11 Circuito da lâmpada.

Conforme dito anteriormente, o circuito da lâmpada fica normalmente aberto e a lâmpada fica apagada. Quando o relé é energizado, o seu contato interno fecha o circuito e a lâmpada é acesa.

Capítulo 4. SOFTWARE DO COMPUTADOR

Conforme dito anteriormente, o projeto foi desenvolvido com a utilização de um computador e um microcontrolador. A função do computador é receber os comandos através de uma interface gráfica e enviar para o microcontrolador. O software do computador foi desenvolvido utilizando a linguagem de programação Java.

Neste capítulo é abordado o software do computador e suas funcionalidades. Na seção 4.1 é tratado sobre a linguagem de programação Java e as bibliotecas de classes. Na seção 4.2 é apresentado o programa editor utilizado. Na seção 4.3 é explicado o funcionamento do Simulador de Entrada de Dados. Na seção 4.4 é apresentada uma descrição de alguns trechos do código-fonte do software do computador.

4.1 Resumo histórico do Java

A linguagem de programação Java foi criada pela Sun Microsystems e liberada para o público em 1995. É uma linguagem baseada nas duas linguagens de implementação mais utilizadas pelos programadores, o C e o C++. Basicamente, a Sun removeu alguns dos recursos mais complexos do C e do C++, como ponteiros, herança múltipla e outros recursos mais propensos a erros.

A linguagem tornou-se apropriada para o ambiente de Internet em função da portabilidade, além dos novos recursos como interface gráfica, tratamento de exceções, multimídia, processamento de banco de dados, redes cliente-servidor e bibliotecas.

O Java foi disponibilizado gratuitamente para milhões de programadores em todo o mundo, o que consolidou seu processo de disseminação. Em 2003, existia cerca de 2,5 milhões de programadores Java, fato surpreendente pois, na época, a linguagem não havia completado dez anos de existência. (Deitel, 2003).

4.1.1 API – Applications Programming Interface

Os programas em Java consistem em partes chamadas de classes. As classes são compostas de outras partes chamadas métodos, responsáveis por realizar tarefas e retornar os resultados obtidos quando as tarefas são concluídas. É possível programar cada método que poderá ser utilizado para formar um programa Java, mas geralmente os programadores utilizam as classes existentes em bibliotecas de classes Java. Essas bibliotecas são também conhecidas como APIs, interfaces de programas aplicativos. (Deitel, 2003).

O software do computador foi desenvolvido utilizando algumas dessas bibliotecas de classes, especialmente a API para a comunicação com as portas serial e paralela.

4.1.2 Java Communications API

A API de comunicação do Java está dividida em três níveis de classes. Existem as classes de alto nível que gerenciam a conexão com as portas de comunicação. As classes de baixo nível, como `SerialPort` e `ParallelPort`, permitem a interface física com as portas seriais (padrão RS-232) e portas paralelas (padrão IEEE 1284), respectivamente. Existem também as classes de nível de driver, que fazem a interface entre as classes de baixo nível e o sistema operacional.

A API de comunicação do Java é fornecida gratuitamente pela Sun Microsystems no endereço *http://java.sun.com/products/javacomm/index.jsp*. Após baixar o arquivo e descompactá-lo, é necessário seguir alguns procedimentos para a instalação da API. Em primeiro lugar, é necessário copiar o arquivo “win32com.dll” para a pasta *C:\Program Files\Java\jre1.5.0_07\bin* (ou pasta equivalente). É importante verificar a versão do J2SE Runtime Environment (JRE). Em seguida, é necessário copiar os arquivos “comm.jar” e “javax.comm.properties” para a pasta *C:\Program Files\Java\jre1.5.0_07\lib*. Para finalizar, é necessário configurar o CLASSPATH para que o arquivo “comm.jar” seja reconhecido. O arquivo “Readme.html” contém o tutorial de instalação completo.

4.2 Eclipse SDK

O desenvolvimento de softwares utilizando a linguagem Java está dividido em algumas etapas. A primeira delas é a edição, que é realizada utilizando um programa editor. Existem diversos tipos de programas editores para o Windows. O Windows Notepad é um exemplo de editor simples, suficiente para a programação em Java. Entretanto, na maioria das vezes os programadores preferem utilizar ambientes de desenvolvimento integrado, os IDEs, pois esses ambientes possuem editores embutidos que são integrados ao ambiente de programação. (Deitel, 2003).

Dentre os vários IDEs disponíveis, no desenvolvimento do software do computador foi utilizado o Eclipse SDK, um software livre certificado pela OSI – Open Source Iniciative. Maiores detalhes e download gratuito estão disponíveis no endereço *http://www.eclipse.org*.

4.3 Simulador de Entrada de Dados

O Simulador de Entrada de Dados é a interface gráfica do projeto. Os comandos do usuário são recebidos pelo Simulador, convertidos em apenas um caractere[3] e enviados para o microcontrolador.

O Simulador está dividido em três classes: Principal.java, Interface.java e Serial.java.

Na Figura 4.1 é mostrada a tela inicial do Simulador de Entrada de Dados com suas funcionalidades.

[3] Para informar o status ao microcontrolador, o autor optou por mandar apenas um caractere. A escolha dos caracteres que representariam os dois estados possíveis foi baseada na segunda letra de cada um dos status, porque tanto “aceso” quanto “apagado” iniciam com a letra “a”. Portanto, o status “aceso” é representado pela letra “c” e o status “apagado” é representado pela letra “p”.

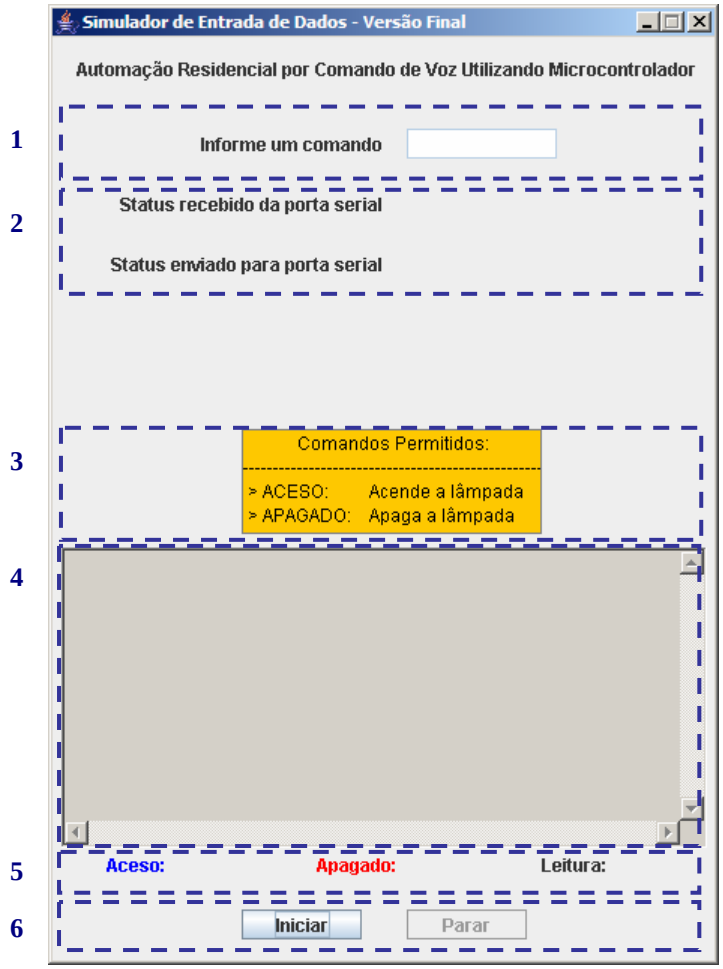

Figura 4.1 Tela inicial do Simulador de Entrada de Dados.

A Figura 4.1 foi dividida em trechos numerados. A seguir é feita uma análise de cada trecho:

1. O *campo de comandos* recebe os comandos de controle da iluminação.
2. Aparece o último status que foi recebido do microcontrolador e o último status que foi enviado, respectivamente.
3. O *box de comandos* mostra para o usuário os comandos permitidos.
4. Este é o *console* do Simulador. Nesse campo aparecerão diversas mensagens para o usuário de acordo com a ação executada.
5. Abaixo do console estão localizados três contadores. O contador da esquerda mostra o número de vezes que a lâmpada foi acesa. O contador do centro mostra o número de vezes que a lâmpada foi apagada. O contador da direita mostra o número de leituras executadas. A partir do momento que o Simulador

é iniciado, o contador de leituras é incrementado duas vezes por segundo (ver Trecho de Código 4.6).

6. Para iniciar a conexão, é necessário clicar no botão *Iniciar*.

Todos os campos e botões, com exceção do botão *Iniciar*, permanecem desabilitados enquanto a conexão não for estabelecida. Ao pressionar o botão *Iniciar*, será estabelecida a conexão do software do computador com a porta serial predefinida. Neste projeto, a porta serial utilizada é a porta “COM1”.

Na Figura 4.2 é mostrada a tela do simulador após estabelecer a conexão.

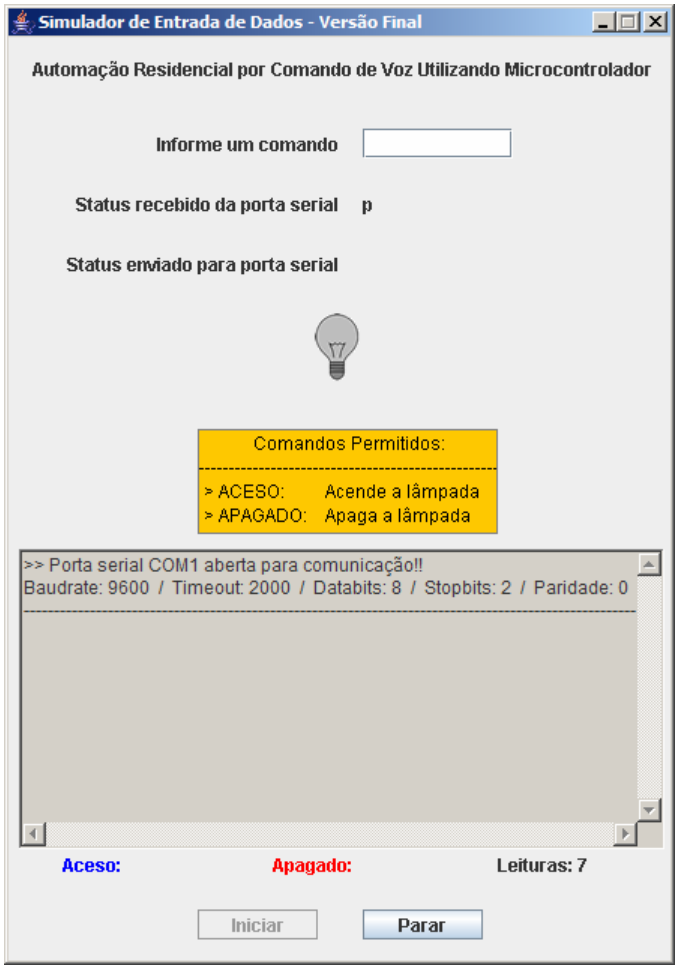

Figura 4.2 Conexão estabelecida.

- O botão *Iniciar* foi desabilitado;
- o botão *Parar* foi habilitado. Ao clicar nesse botão, a simulação é suspensa e o botão *Parar* é substituído pelo botão *Fechar*;
- o *campo de comando* também foi habilitado;

- no *console* aparece qual porta foi aberta e alguns parâmetros de configuração da conexão serial;
- o microcontrolador enviou a confirmação de recebimento do comando inicial “apagado” (ver subitem 4.4.1);
- acima do *box de comandos* apareceu a figura de uma lâmpada apagada. Da mesma forma, quando o Simulador receber o comando “aceso”, aparecerá a figura de uma lâmpada acesa, como é mostrado na Figura 4.3.

Após iniciar a simulação, o contador de leituras será incrementado devido à leitura automática do *campo de comandos* (ver subitem 4.4.5).

Na Figura 4.3 é apresentado o recebimento do comando “aceso”, com algumas diferenças com relação à Figura 4.2:

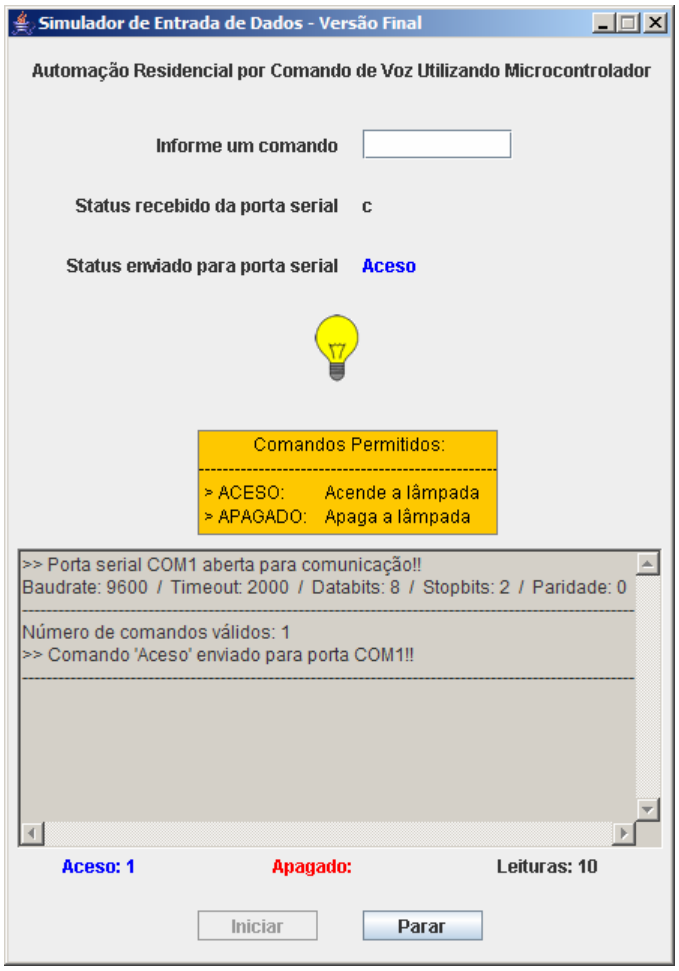

Figura 4.3 Recebimento de comando.

- A figura da lâmpada apagada foi substituída pela lâmpada acesa;
- apareceu o status enviado para a porta serial e a confirmação de recebimento enviada pelo microcontrolador;
- o contador de comandos foi incrementado;
- o console mostrou uma mensagem de confirmação de envio.

4.4 Descrição do código-fonte

Para facilitar o entendimento do funcionamento do software do computador, esta seção apresenta uma descrição de alguns trechos do código-fonte das classes do software do computador. Nos Apêndices A, B e C são apresentados os códigos-fonte completos.

4.4.1 Chamada para estabelecimento da comunicação serial – *Interface.java*

```
/* inicia conexão */
// Porta: definida anteriormente
// Baudrate: tempo de transmissão de cada bit (bauds)
// Timeout: tempo de espera para estabelecer da conexão (ms)
Serial serial = new Serial(porta, 9600, 2000);

/* método que abre a porta, inicia leitura dos dados
/* e inicia a simulação com o status 'Apagado' */
public void estabeleceConexao(){
  try {
    serial.AbrirPorta();

    // contador do número de acessos
    primeiroAcesso ++;

    // mostra configuração da porta serial no console
    if (primeiroAcesso == 1){
      String configuracao = serial.configuracao;

      areaConsole.setText(">> Porta serial " + porta +
        " aberta para comunicação!! \n" + configuracao +
        "\n---------------------------------------" +
        "---------------------------------------" +
        "--------------------\n");

      // inicia com o status 'Apagado'
      serial.EnviarString("p");
      lblConfirmaEnviado.setText(status);
      lblConfirmaEnviado.setForeground(Color.RED);
      lblIcon.setIcon(icons[0]);
    }

    // inicia leitura
    serial.LerDados();
```

```
        // leitura iniciada com sucesso
        start = true;
      }
      catch (Exception e){
        String console = areaConsole.getText();
        areaConsole.setText(console + serial.erroSerial);

        botaoIniciar.setVisible(false);
        botaoParar.setVisible(false);

        botaoFechar.setVisible(true);
        botaoFechar.requestFocus();
      }
  }
```

Trecho de Código 4.1 Chamada ao método que estabelece a comunicação serial.

No Trecho de Código 4.1 ocorre a chamada ao método que estabelece a comunicação serial. Em primeiro lugar, ocorre a declaração de um novo objeto `serial` com a passagem de três parâmetros de configuração da conexão. O parâmetro `Porta` informa qual porta serial será utilizada na comunicação. O parâmetro `Baudrate` define o tempo de transmissão de cada *bit*, medido em *bauds* (ver seção 3.2.2). Por último, o parâmetro `Timeout` define o tempo de espera da conexão em milissegundos.

Quando a simulação é iniciada, os métodos `AbrirPorta()` é chamado (ver Trecho de Código 4.2). Em seguida, o contador `primeiroAcesso` é incrementado. Se `primeiroAcesso` for igual a um, significa que é a primeira vez que o Simulador é iniciado. Nesse caso, aparecerá na tela a configuração da comunicação serial. Conforme dito anteriormente na seção 3.1, o status inicial da simulação é “apagado”. Desta forma, o método `public void estabeleceConexao()` envia ao microcontrolador o status `"p"` no primeiro acesso.

O método `LerDados()` é chamado em seguida (ver Trecho de Código 4.3). Após iniciada a leitura dos dados, os dados enviados pelo microcontrolador para a porta serial serão recebidos pelo software do computador. É importante ressaltar que o fato de manter a leitura sempre ativa não impede a escrita na porta serial.

Caso ocorra algum erro na abertura da porta serial ou na leitura dos dados, o console apresentará a mensagem de erro e aparecerá o botão *Fechar*.

4.4.2 Controle da porta serial – *Serial.java*

```
   /* recebe objeto da classe Interface.java
   /* com detalhes de configuração */
   public Serial(String p, int b, int t) {
      this.Porta = p;
      this.baudrate = b;
      this.timeout = t;
   }

   /* Abre a comunicação da porta */
   public void AbrirPorta() {
      // Obtém o ID da porta
      try {
         cp = CommPortIdentifier.getPortIdentifier(Porta);
         if (cp == null) {
            IDPortaOK = false;
         }
         IDPortaOK = true;
      } catch (Exception e) {
         IDPortaOK = false;
      }
      // Abre porta serial para comunicação
      try {
         porta = (SerialPort) cp.open("Simulador de Entrada de Dados",
timeout);
         PortaOK = true;

         // configuração dos parâmetros de porta
         porta.setSerialPortParams(baudrate, porta.DATABITS_8,
              porta.STOPBITS_2, porta.PARITY_NONE);

         // string configuração com informações da configuração
         configuracao =  "Baudrate: " + baudrate + "  /  " +
                    "Timeout: "  + timeout + "  /  " +
                    "Databits: " + porta.DATABITS_8 + "  /  " +
                    "Stopbits: " + porta.STOPBITS_2 + "  /  " +
                    "Paridade: " + porta.PARITY_NONE;

         PortaAberta = true;
      } catch (PortInUseException e) {
         PortaOK = false;
         erroSerial = ">> Erro ao abrir a porta serial!!" +
            "\nMotivo: A porta serial está sendo utilizada" +
            " por outro aplicativo" +
            "\n----------------------------------------" +
            "----------------------------------------" +
            "--------------------\n";
      } catch (NullPointerException e) {
         PortaOK = false;
         erroSerial = ">> Erro ao abrir a porta serial!!" +
            "\nMotivo: A porta serial não está disponível" +
            "\n----------------------------------------" +
            "----------------------------------------" +
            "--------------------\n";
      } catch (Exception e) {
         PortaOK = false;
         erroSerial = ">> Erro ao abrir a porta serial!!" +
            "\nMotivo: " + e +
            "\n----------------------------------------" +
            "----------------------------------------" +
            "--------------------\n";
      }
```

```
    // habilita escrita e leitura
    try {
        escrita = porta.getOutputStream();
        leitura = porta.getInputStream();
    } catch (IOException e) {
        porta.close();
       erroSerial = ">> Erro ao habilitar leitura e escrita!!" +
          "\nMotivo: " + e +
          "\n----------------------------------------" +
          "----------------------------------------" +
          "--------------------\n";
    }
}
```

Trecho de Código 4.2 Controle da porta serial.

O método `public Serial(String p, int b, int t)` recebe os parâmetros passados pelo objeto `serial` (ver Trecho de Código 4.1).

O método `public void AbrirPorta()` obtém o número de identificação da porta para verificar sua existência. Em seguida, abre a porta serial, informa os parâmetros de conexão e habilita a comunicação para envio e recebimento de dados.

4.4.3 Leitura de dados – *Serial.java*

```
/* habilita leitura de dados */
public void LerDados() {
   try {
      porta.addEventListener(this);
   } catch (Exception e) {
      erroSerial = ">> Erro ao criar listener!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
   porta.notifyOnDataAvailable(true);
   try {
      threadLeitura = new Thread(this);
      threadLeitura.start();
   } catch (Exception e) {
      erroSerial = ">> Erro ao iniciar leitura!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
}

/* gerenciador de eventos de leitura */
   case SerialPortEvent.DATA_AVAILABLE:
      byte[] bufferLeitura = new byte[1];
      try {
         while (leitura.available() > 0) {
            numBytes = leitura.read(bufferLeitura);
         }
```

```
            String Dadoslidos = new String(bufferLeitura);
            if (bufferLeitura.length == 0) {
            } else {
               recebido = Dadoslidos;
            }
         } catch (Exception e) {
            erroSerial = ">> Erro durante a leitura!!" +
               "\nMotivo: " + e +
               "\n----------------------------------------" +
               "----------------------------------------" +
               "--------------------\n";
         }
      break;
```

Trecho de Código 4.3 Leitura de dados

No Trecho de Código 4.3 é apresentado o método `public void LerDados()` e um trecho do método `public void serialEvent(SerialPortEvent ev)`, gerenciador de eventos da classe *Serial.java*. O primeiro método é chamado ao iniciar a simulação (ver Trecho de Código 4.1) e habilita o recebimento de dados. Quando algum dado é enviado pelo microcontrolador, o gerenciador de eventos analisa e armazena o dado em `recebido`.

4.4.4 Envio de dados – *Serial.java*

```
/* envia string para porta serial */
public void EnviarString(String msg) {
   try {
      escrita.write(msg.getBytes());
      Thread.sleep(100);
      escrita.flush();
   } catch (Exception e) {
      erroSerial = ">> Houve um erro durante o envio!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
}
```

Trecho de Código 4.4 Envio de dados.

O método `public void EnviarString(String msg)` recebe uma *string* e envia para `escrita`. O atributo `escrita` é responsável por enviar o dado para a porta serial. Após o envio, ocorre uma pausa de 100 milissegundos e, em seguida, `escrita` é esvaziado.

4.4.5 Leitura automática dos comandos – *Principal.java*

```
/* método principal */
public static void main(String args[]) {
   // chama thread leituraCampo
   Principal application = new Principal();
   application.leituraCampo();
}


/* monitora comandos recebidos (thread) */
public void leituraCampo(){
   try{
      verificaComando = new Thread(this);
      verificaComando.start();
   }
   catch (Exception e){
      e.printStackTrace();
   }

   // termina a simulação ao fechar a janela
   layout.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}
```

Trecho de Código 4.5 Leitura automática dos comandos.

No Trecho de Código 4.5 é apresentado o método principal `public static void main(String args[])` e o método de leitura automática `public void leituraCampo()`.

O método principal é executado ao iniciar o simulador. Dentro do método principal é feita a chamada ao método de leitura automática.

O método `public void leituraCampo()` é responsável pela leitura automática do *campo de comandos*. É declarada uma nova *thread* `verificaComando`. Em seguida, a *thread* é iniciada, habilitando a verificação automática do *campo de comandos*. A periodicidade da repetição é determinada no método `public void run()` (ver Trecho de Código 4.6).

Quando a tela do Simulador for fechada, a simulação é encerrada e, caso a porta serial esteja aberta, a mesma é fechada. Essa é uma ação padrão de saída definida pelo comando `layout.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`.

4.4.6 Execução da simulação – *Principal.java*

```
/* inicia simulação */
public void run(){
   try{
     while(true){ // loop

        Thread.sleep(500); // sleep em ms (1000 ms = 1 s)

        atualizaDados();

        if(layout.primeiroAcesso > 0 && layout.start == true){
          // chama método leituraSerial
          leituraSerial();
          // recebe comando
          layout.status = layout.txtComando.getText();

          // comandos inválidos
          if(layout.status.length() == 0){
            // campo comando vazio
            contTimer = 0;
          }
          if(layout.status.length() > 0){
            // campo comando preenchido e comando inválido

            // não apagará imediatamente um comando inválido
            // porque pode ser que a palavra esteja sendo escrita
            // no momento da leitura.
            contTimer ++;

            if (contTimer > 2){
              // após xx ciclos com comando inválido,
              // o campo comando será apagado e ciclo reiniciado
              layout.status = layout.txtComando.getText();
              layout.txtComando.setText("");
              contTimer = 0;
              contInvalido ++;

              console = layout.areaConsole.getText();
              layout.areaConsole.setText("" + console +
                   "Número de comandos inválidos: " + contInvalido +
                   "\n>> ATENÇÃO! Comando '" + layout.status +
                   "' inválido!!"+
                   "\n----------------------------------------" +
                   "----------------------------------------" +
                   "--------------------\n");
            }
          }

          // comandos válidos
          // caso 'Aceso'
          if (layout.status.equalsIgnoreCase("aceso")  ||
            layout.status.equalsIgnoreCase(" aceso") ||
            layout.status.equalsIgnoreCase("aceso ") ||
            layout.status.equalsIgnoreCase("acesa")  ||
            layout.status.equalsIgnoreCase(" acesa") ||
            layout.status.equalsIgnoreCase("acesa ") ||
            layout.status.equalsIgnoreCase("acesos") ||
            layout.status.equalsIgnoreCase(" acesos")||
            layout.status.equalsIgnoreCase("acesos ")||
            layout.status.equalsIgnoreCase("acesas") ||
            layout.status.equalsIgnoreCase(" acesas")||
            layout.status.equalsIgnoreCase("acesas ")
            ){
```

```
                    layout.status = "Aceso";

                    // chama método statusValido
                    statusValido(1); // 1: Aceso
                }
                // caso 'Apagado'
                if (layout.status.equalsIgnoreCase("apagado")  ||
                   layout.status.equalsIgnoreCase(" apagado") ||
                   layout.status.equalsIgnoreCase("apagado ") ||
                   layout.status.equalsIgnoreCase("apagada")  ||
                   layout.status.equalsIgnoreCase(" apagada") ||
                   layout.status.equalsIgnoreCase("apagada ") ||
                   layout.status.equalsIgnoreCase("apagados")  ||
                   layout.status.equalsIgnoreCase(" apagados") ||
                   layout.status.equalsIgnoreCase("apagados ") ||
                   layout.status.equalsIgnoreCase("apagadas")  ||
                   layout.status.equalsIgnoreCase(" apagadas") ||
                   layout.status.equalsIgnoreCase("apagadas ")
                   ){
                      layout.status = "Apagado";
                      // chama método statusValido
                      statusValido(0); // 0: Apagado
                }
              }
            }
          } catch (Exception e){
            e.printStackTrace();
          }
        }
```

Trecho de Código 4.6 Execução da simulação.

No Trecho de Código 4.6 é apresentado o método `public void run()` que é executado ao iniciar a simulação. A estrutura de repetição `while(true)` é responsável por manter a execução da função em *loop* infinito.

A periodicidade da repetição é definida em `Thread.sleep(500)`, onde o tempo de espera é definido em milissegundos. Portanto, antes de cada execução, o método faz uma pausa de 500 ms ou 0,5 s.

Os comandos válidos são “aceso” e “apagado”, além de suas variações em gênero e número.

CAPÍTULO 5. SOFTWARE DO MICROCONTROLADOR

Para o desenvolvimento do projeto, foi proposto o uso de um computador comunicando com um microcontrolador. A função do computador é receber os comandos através de uma interface gráfica (ver Capítulo 4). O microcontrolador tem a função de receber os dados enviados via interface serial pelo computador e controlar o acendimento da lâmpada.

Neste capítulo é abordado o software do microcontrolador e os detalhes relacionados. Na seção 5.1 é apresentado o kit de desenvolvimento CW552. Na seção 5.2 é apresentada uma descrição de alguns trechos do código-fonte do software do microcontrolador.

5.1 Kit de Desenvolvimento CW552

Para o desenvolvimento do projeto, foi escolhido o uso do kit de desenvolvimento CW552, o mesmo utilizado na disciplina Microcontroladores e Microprocessadores no curso de Engenharia da Computação. O kit CW552 é representado pela Figura 5.1.

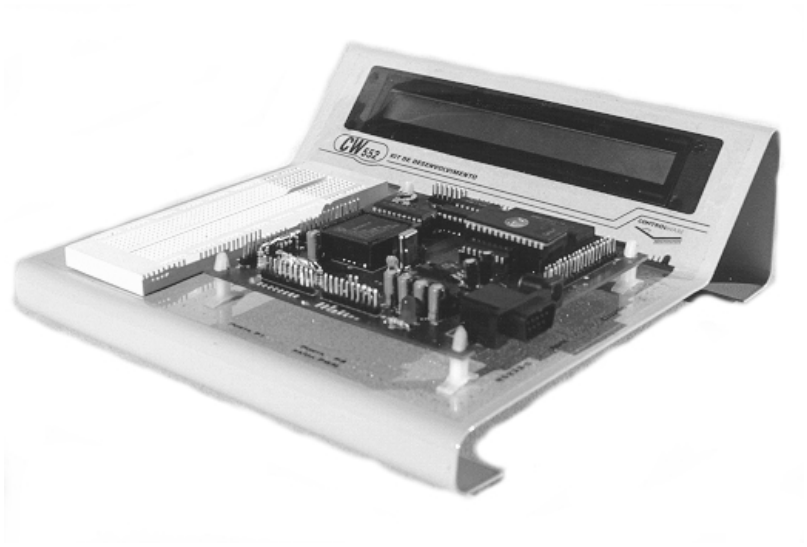

Figura 5.1 Kit de desenvolvimento CW552.

Fonte: ControlWare Automação, 2001.

A seção 5.1 foi escrita com base no manual do kit de desenvolvimento CW552, disponível no CD de instalação.

5.1.1 Apresentação

O microcontrolador utilizado no kit CW552 é o 80c552, pertencente à família do microcontrolador 8051. O kit CW552 é composto por uma base de suporte em acrílico, uma fonte de alimentação de 9V, um display de cristal líquido de duas linhas e 40 colunas, uma interface serial padrão RS-232, um *protoboard* e uma placa de circuito impresso com o microcontrolador.

Na Figura 5.2 é mostrado o diagrama de blocos do kit CW552.

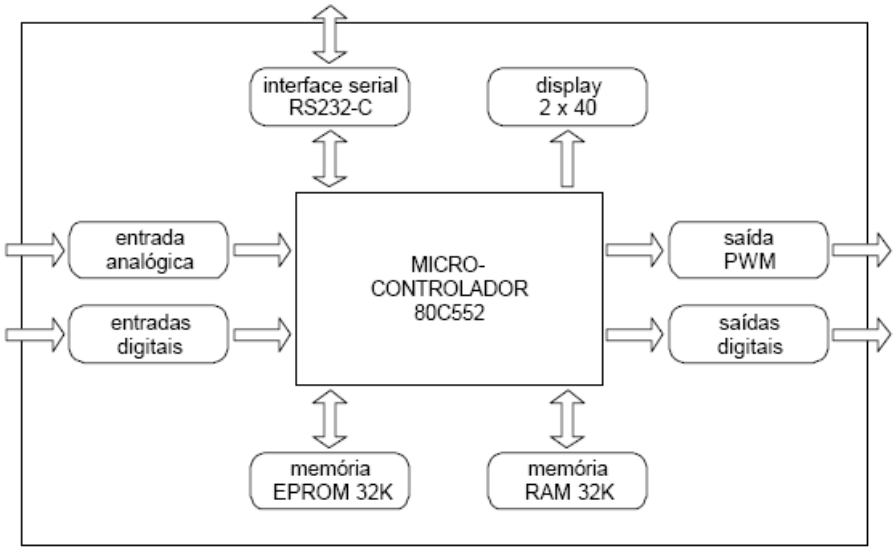

Figura 5.2 Diagrama de blocos do kit de desenvolvimento CW552.

Fonte: ControlWare Automação, 2001.

5.1.2 Comunicação do kit CW552 com o computador

A ligação do kit CW552 ao computador é feita através de um cabo serial no padrão RS-232C com a ligação cruzada, também conhecido por *cross-over*. Na Figura 5.3 é representada a pinagem do cabo de comunicação serial.

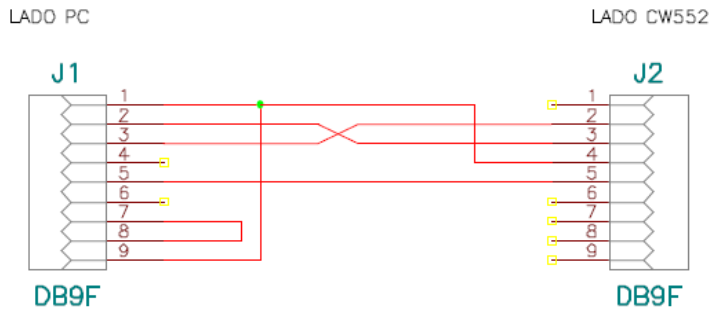

Figura 5.3 Pinagem do cabo de comunicação serial.

Fonte: ControlWare Automação, 2001.

5.1.3 Comunicação do kit CW552 com a interface elétrica

Os pinos de entrada e saída do kit CW552 possuem fácil acesso, conforme é mostrado na figura abaixo.

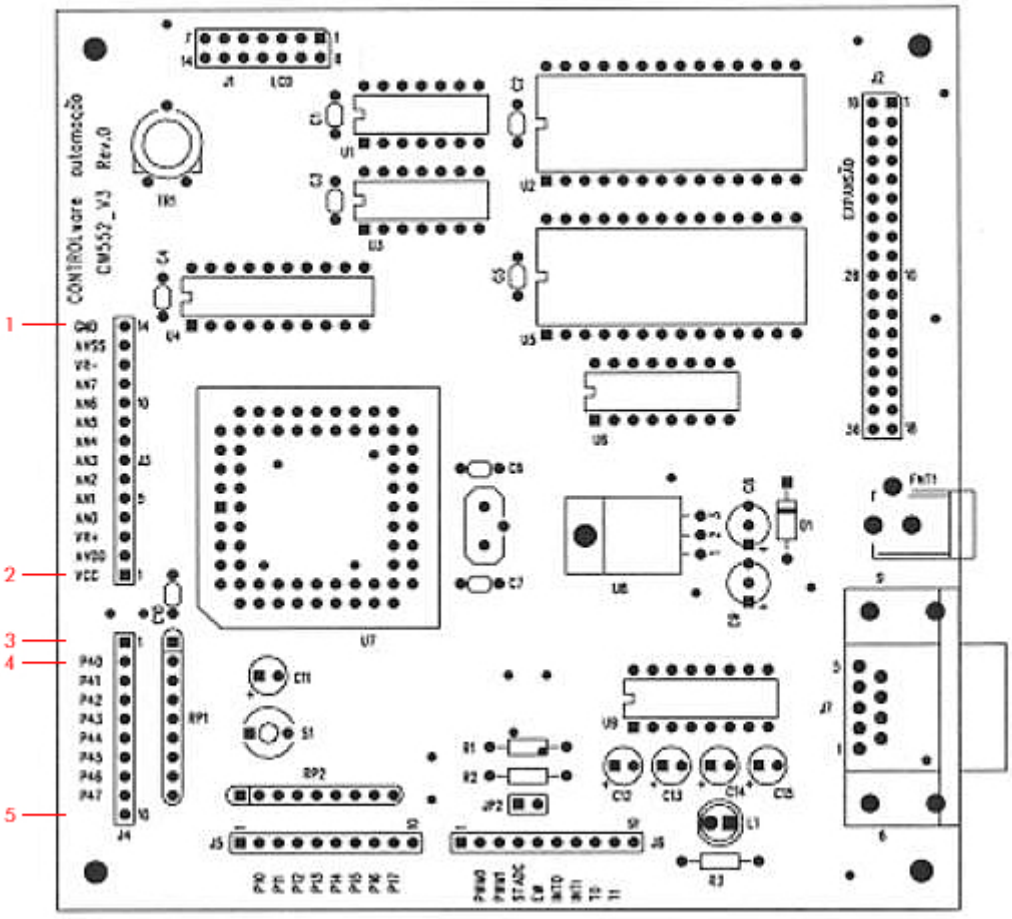

Figura 5.4 Layout da placa, destacando os pinos utilizados no projeto.

Fonte: ControlWare Automação, 2001.

Para alimentar o circuito, foram utilizados cinco pinos, sendo dois pinos com tensão 5V, dois pinos com tensão 0V (terra) e um pino de controle. Na Tabela 5.1 é mostrada a função dos pinos utilizados com a numeração apresentada na Figura 5.4.

Tabela 5.1 Pinos utilizados.

Número	Slot	Pino	Função
1	J3	GND	Aterramento do circuito do relé
2	J3	VCC	Alimentação (5V) do circuito do relé
3	J4	-	Alimentação (5V) do circuito do microcontrolador
4	J4	P40	Controle do acendimento (0V ou 5V)
5	J4	-	Aterramento do circuito do microcontrolador

5.1.4 Execução de um programa

A execução de um programa a partir do kit de desenvolvimento CW552 está dividida em quatro etapas. Após a escrita do software com o auxílio de um programa editor, é feita a compilação e geração de um programa executável no formato Intel-Hex. A compilação é feita pelo software monitor instalado na memória EPROM. Em seguida, o programa executável é transferido para a memória RAM do kit. Finalmente, o programa é executado.

5.1.5 Compiladores e Linguagem de Programação C

Qualquer computador ou microcontrolador é capaz de entender diretamente somente sua própria linguagem de máquina, que é sua linguagem natural. As linguagens de máquina são seqüências de números, o que dificulta o entendimento e torna essas linguagens incômodas para o ser humano. Com o passar do tempo, foram desenvolvidas abreviações semelhantes ao inglês, conhecidas como linguagens assembler, ou simplesmente Assembly. O uso do Assembly ainda exigia muitas instruções para executar simples comandos. Para acelerar o processo de programação, foram desenvolvidas linguagens de alto nível, como o C e o Java. Os compiladores são responsáveis pela conversão dos programas de alto nível em programas de máquina. (Deitel, 2003).

5.1.5.1 Compilador SDCC

Para facilitar o desenvolvimento do software para o kit CW552, o programador pode optar por utilizar o compilador SDCC – Small Device C Compiler. Esse é um compilador com código-fonte aberto e possui algumas características que facilitam o acesso aos recursos especiais do microcontrolador. Desta forma, com o uso da linguagem C e das extensões do SDCC, o programador tem acesso a todas as áreas de memória do microcontrolador.

Neste projeto foi utilizada a versão 2.6.0 do SDCC. O download gratuito está disponível no endereço *http://sdcc.sourceforge.net*.

5.1.5.2 Resumo histórico do C

A linguagem C foi desenvolvida em 1972 e se tornou conhecida como a linguagem de programação para o UNIX. Atualmente, grande parte dos novos sistemas operacionais estão escritos em C ou C++, uma extensão de C.

Nas últimas décadas, o C se tornou disponível para a maioria dos computadores porque não depende de hardware. A larga utilização do C em plataformas de hardware diferentes gerou muitas variações semelhantes, mas geralmente incompatíveis. A fim de solucionar esse problema e permitir a portabilidade dos programas escritos em C, foi criado um comitê técnico para a padronização da linguagem. Em 1989, foi aprovado um padrão para a linguagem C. No ano seguinte, ocorreu a padronização mundial de uma versão do C com a cooperação da ISO e foi estabelecido um documento conhecido como ANSI/ISO 9899:1990. Desde então, a versão ANSI C é a mais utilizada. (Deitel, 2003).

5.2 Descrição do código-fonte

O software do microcontrolador é responsável por receber os dados enviados pelo computador, confirmar o recebimento e controlar o acendimento da iluminação. O arquivo “Controle.c” é compilado previamente e instalado na memória RAM do microcontrolador.

Assim como na seção 4.4, esta seção apresenta uma descrição de alguns trechos do código-fonte do software do microcontrolador. No Apêndice D é apresentado o código-fonte completo.

5.2.1 Recebe o status

```
// Função que recebe os dados da serial
void recebeDado()
{
  while(!RI); // Quando RI = 1, terminou a recepção

  // Salva o que foi recebido na variável
  dadoRecebido=SBUF;

  // Limpa RI para permitir nova recepção
  RI = 0;

  // Envia confirmação de leitura
  statusEnviado = dadoRecebido;
  enviaStatus(&statusEnviado);

  // Incrementa contador de dados recebidos
  contRecebido ++;

}
```

Trecho de Código 5.1 Recebe o status enviado pelo computador.

A função `void recebeDado()` apresentada no Trecho de Código 5.1 é responsável por receber os dados enviados pelo computador. Em `while(!RI)`, a função está aguardando a chegada completa do dado. Em seguida, o dado recebido é salvo pela variável `dadoRecebido`. Ocorre a confirmação de leitura, onde o dado recebido é enviado novamente para o computador. Por fim, o contador de dados recebidos `contRecebido` é incrementado.

5.2.2 Envia o status

```
// Função que envia confirmação do dado recebido
void enviaStatus(int *dado)
{
  // Configura a comunicação serial
  confSerial();

  // Recebe o valor contido no endereço apontado pelo ponteiro
  SBUF=*dado;
```

```
    // Inicia a transmissão automaticamente
    while(!TI); // Quando TI = 1, terminou a transmissão

    // Limpa TI para permitir nova transmissão
    TI=0;

}
```

Trecho de Código 5.2 Envia status da iluminação.

No Trecho de Código 5.2 é apresentada a função que envia o status da lâmpada para o software do computador via porta serial. Em primeiro lugar, ocorre a configuração da comunicação serial. A função `confSerial()` foi obtida através de uma modificação no programa-exemplo “tst_ser.c”, contido no CD de configuração do kit CW 552.

A função `void enviaStatus(int *dado)` recebe o valor do ponteiro e transmite automaticamente.

CAPÍTULO 6. AVALIAÇÃO DE DESEMPENHO

A viabilidade do projeto depende de diversos fatores, dentre eles a comprovação do correto funcionamento do projeto. A confiabilidade do projeto é relação entre o que foi dito pelo usuário e o que o software interpretou, e depende essencialmente do software IBM Via Voice.

Neste capítulo são abordadas as formas de avaliação de desempenho e os resultados obtidos. Na seção 6.1 são apresentados os dados obtidos nas análises de confiabilidade. Na seção 6.2 é apresentado o software utilizado durante o desenvolvimento do Simulador de Entrada de Dados.

6.1 Resultados obtidos

Conforme descrito na seção 3.1, a entrada de dados é feita através de dois softwares. A confiabilidade do software IBM Via Voice foi medida através de diversas simulações de entrada de dados.

Para iniciar a simulação, foi necessário criar um padrão pessoal de voz no software IBM Via Voice. As etapas de criação do padrão pessoal de voz foram descritas anteriormente no subitem 3.1.1 e foram concluídas em, aproximadamente, 50 minutos.

6.1.1 Análise da confiabilidade do software IBM Via Voice

Após a criação do padrão pessoal de voz, o software já está preparado para o uso, porém apresenta algumas falhas no reconhecimento de voz. O motivo das falhas é que o software cria um padrão de voz através de algumas palavras que são lidas durante o treinamento e generaliza para as demais palavras. Por exemplo, algumas vezes, quando foi pronunciada a palavra “aceso”, o software interpretou a voz e escreveu a expressão “as seis”. Dessa forma, o correto entendimento de certas palavras por parte do software

depende de treinamento exclusivo dessas palavras. Portanto, é possível perceber que a confiabilidade do software é proporcional à quantidade de treinamento submetido.

6.1.1.1 Percentual de confiabilidade dos comandos válidos

Conforme é demonstrado na Figura 6.1, o percentual de confiabilidade para o comando “Aceso” foi 40%, ou seja, 30 entre os 50 comandos ditos foram mal interpretados pelo software. Na Figura 6.2 é demonstrado que ocorreu um fato semelhante para o comando “Apagado”, que obteve 54% de confiabilidade. A principal razão é a semelhança com outras expressões, como “as seis” e “apagar do”.

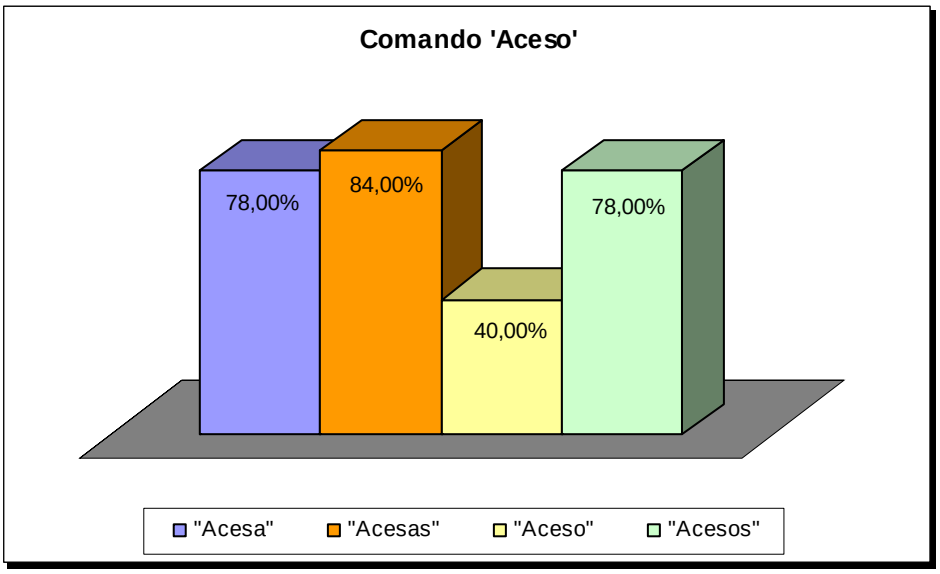

Figura 6.1 Análise do comando “Aceso”.

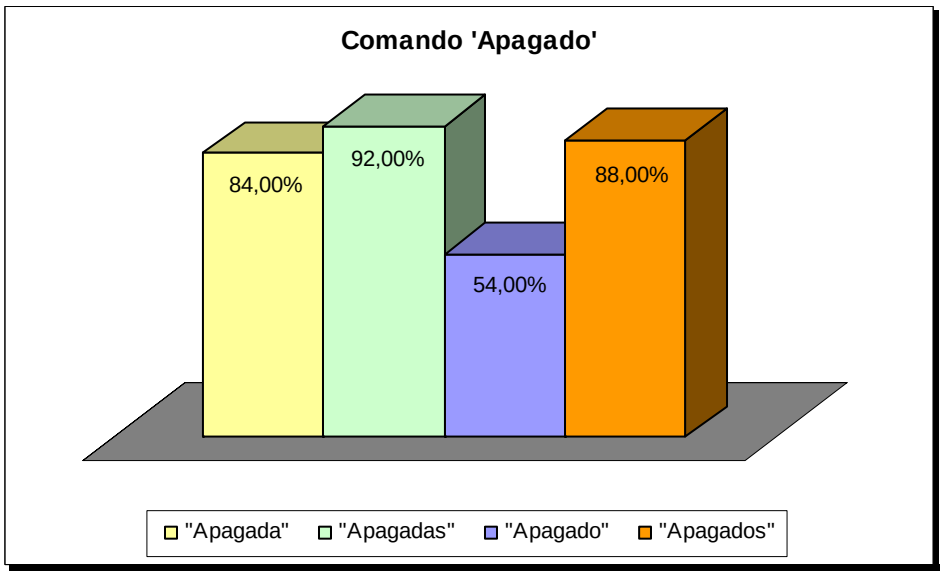

Figura 6.2 Análise do comando “Apagado”.

Devido à baixa confiabilidade para os comandos válidos, aliado ao fato de diversas vezes o software IBM Via Voice ter interpretado as palavras no gênero feminino, o Simulador de Entrada de Dados foi alterado para aceitar as variações de gênero e número dos comandos válidos. Dessa forma, foram ditos os comandos válidos e suas variações em gênero e número.

6.1.1.2 Análise de falsas detecções

Na Tabela 6.2 é apresentada uma comparação entre os comandos informados e o resultado obtido.

Tabela 6.1 Classificações com relação à análise de falsas detecções.

Classificação	Comando	Resultado
Verdadeiro Positivo	Válido	Válido
Verdadeiro Negativo	Inválido	Inválido
Falso Positivo	Inválido	Válido
Falso Negativo	Válido	Inválido

Quando o resultado é classificado como verdadeiro positivo, significa que foi informado um comando válido e o software interpretou corretamente. Caso o resultado seja verdadeiro negativo, o comando informado é inválido e o resultado também é inválido. Quando o resultado é classificado como falso positivo, um comando inválido é informado, mas o software interpreta como um comando válido. No caso do resultado falso negativo, é informado um comando válido e o software interpreta de forma errada.

Na Figura 6.4 é demonstrada uma comparação entre os resultados. Foram feitas 50 amostras de cada classificação.

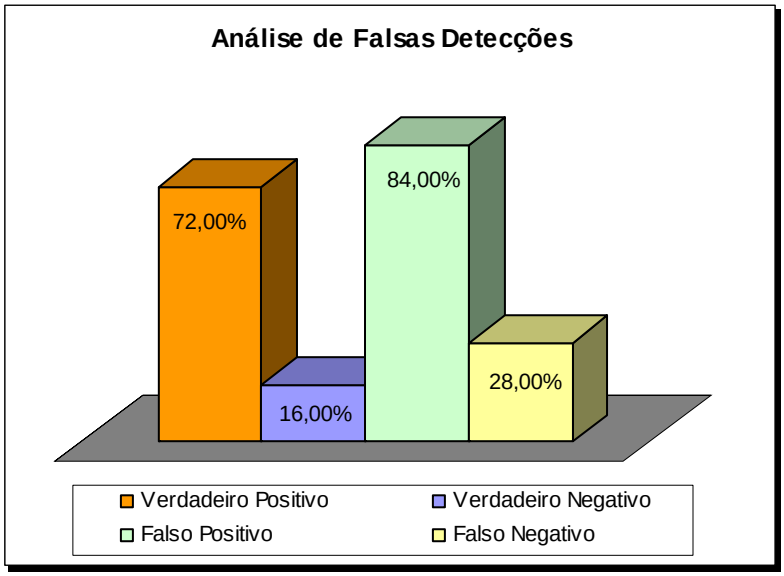

Figura 6.3 Análise de falsas detecções.

6.1.1.3 Nível de ruído no ambiente

Um fator importante na análise da confiabilidade é o nível de ruído no ambiente. Na Tabela 6.2 são apresentados valores aproximados do nível de ruído de alguns ambientes e objetos (Bertulani, 2006).

Tabela 6.2 Valores aproximados do nível de ruído.

Nível de Intensidade (dB)	Intensidade do som (W/m^2)	Exemplos típicos
130	10	Limiar da percepção
120	1,0	Grande avião a jato
110	0,1	Grande orquestra
100	0,01	Arrebitamento
90	10^{-3}	Trem
80	10^{-4}	Escritório ruidoso
70	10^{-5}	Motor de carro
60	10^{-6}	Discurso
50	10^{-7}	Escritório com ruído médio
40	10^{-8}	Escritório quieto
30	10^{-9}	Biblioteca
20	10^{-10}	Sussurro
10	10^{-11}	Sussurro bem baixo
0	10^{-12}	Limiar da audibilidade (a 1000 Hz)

Fonte: Bertulani, 2006.

A fim de analisar e comprovar que a confiabilidade é inversamente proporcional ao nível de ruído do ambiente, foram feitas simulações em um quarto de 9 m^2 baseado nos exemplos de escritório quieto e escritório com ruído médio da Tabela 6.2.

O ambiente quieto foi simulado quando não havia nenhum tipo de ruído dentro do quarto. De acordo com a Tabela 6.2, essa situação se assemelha ao escritório quieto e possui, aproximadamente, 40 dB de ruído. Na Figura 6.4 é mostrada a análise de confiabilidade em um ambiente quieto.

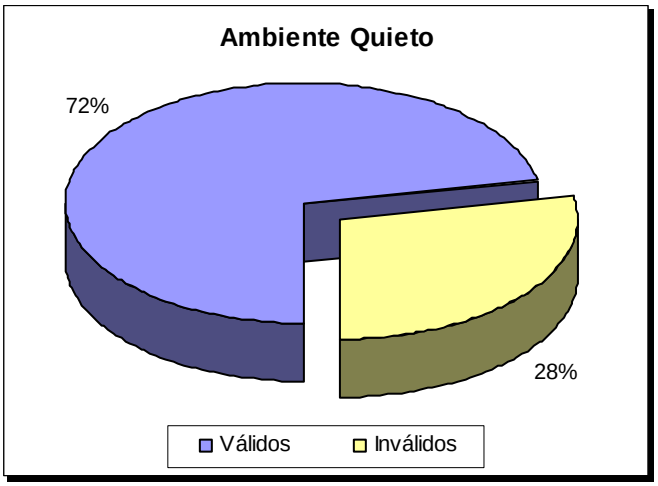

Figura 6.4 Confiabilidade em um ambiente quieto.

Para simular o ambiente com ruído médio, foi utilizado um aparelho de televisão, simulando um escritório com ruído médio. Essa situação possui, aproximadamente, 50 dB de ruído. Na Figura 6.5 é mostrado o resultado obtido em um ambiente com ruído médio.

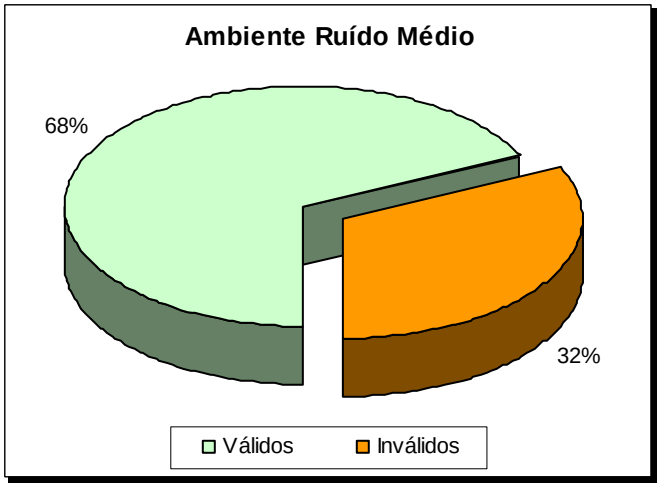

Figura 6.5 Confiabilidade em um ambiente com ruído médio.

Os valores obtidos nas simulações comprovam que o software de reconhecimento de voz possui um melhor desempenho em ambientes quietos.

6.1.1.4 Reconhecimento da voz de outro usuário

O software IBM Via Voice foi treinado por uma voz masculina. A análise do reconhecimento da voz de outro usuário foi feita por uma mulher utilizando o padrão pessoal de voz treinado e não foram obtidos resultados satisfatórios. De acordo com o software IBM Via Voice, “o padrão de voz refere-se a um arquivo de voz que contém dados de fala reunidos para cada usuário”. Cada pessoa tem pronuncia e modo de falar diferentes. Por isso, é necessário que cada usuário crie um padrão pessoal de voz.

6.2 Free Serial Port Monitor

Durante o desenvolvimento do Simulador de Entrada de Dados, foi utilizado o software de licença *freeware* Free Serial Port Monitor. Esse software tornou possível analisar os dados enviados para a porta serial. Na fase de desenvolvimento do software do microcontrolador, o Free Serial Port Monitor possibilitou a análise dos dados recebidos na porta serial.

O Free Serial Port Monitor monitora as portas seriais utilizadas por qualquer aplicação que rode nas versões NT 4.0, 2000, XP e 2003 Server do Microsoft Windows. Esse software intercepta, analisa e mostra todos os dados transferidos entre a aplicação e o dispositivo conectado na porta serial. É um mecanismo eficiente que ajuda a encontrar e eliminar erros de software ou hardware.

No projeto foi utilizada a versão 3.31 do Free Serial Port Monitor, disponibilizada em maio de 2005. Outras informações e download gratuito no endereço *http://www.serial-port-monitor.com*.

CAPÍTULO 7. CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as considerações finais relacionadas ao projeto. Na seção 7.1 são apresentadas as conclusões. Na seção 7.2 são apresentadas as dificuldades encontradas durante todas as etapas do projeto. Na seção 7.3 são apresentadas as sugestões para projetos futuros.

7.1 Conclusões

Neste projeto foi proposto o desenvolvimento de uma solução de automação residencial, através de comando de voz. Foi possível integrar diversos conhecimentos adquiridos durante todo o curso.

Foi observada uma praticidade proporcionada pelo comando de voz, o que oferece um aumento significativo na qualidade de vida do usuário, sobretudo nos casos de deficientes físicos. Todavia, a utilização do software IBM Via Voice é uma solução que demanda muito tempo dedicado ao treinamento do software para aperfeiçoamento do padrão pessoal de voz. O fato de ser um sistema de reconhecimento de voz dependente – depende do treinamento para criação de um padrão pessoal de voz – e discreto – as palavras devem ser ditas pausadamente – pode comprometer a confiabilidade do projeto. Após diversos treinamentos, foram obtidos resultados considerados satisfatórios com relação à confiabilidade.

Conforme os resultados obtidos na seção 6.1, o projeto é viável e atendeu aos requisitos propostos.

Alguns fatores que são requisitos básicos para o desenvolvimento de soluções de automação residencial e para a tecnologia de comandos de voz foram atendidos neste projeto:

- Relação custo-benefício, pois foram utilizados softwares *freeware*, com exceção do software IBM Via Voice;

- confiabilidade satisfatória;
- interatividade com o usuário, proporcionada pelo Simulador de Entrada de Dados, o que facilita o uso da solução;
- atualização tecnológica simples, uma vez que foram utilizadas linguagens de programação bastante difundidas;
- conforto e conveniência proporcionados pelo comando de voz;
- possibilidade de integração de diferentes soluções e outros aparelhos domésticos;
- confirmação do comando recebido através do Simulador de Entrada de Dados.

7.2 Dificuldades encontradas

Durante o desenvolvimento do projeto, houve algumas dificuldades. A primeira dificuldade foi relativa à transmissão e recebimento de dados via porta serial. Inicialmente, não havia definição sobre a linguagem de programação que seria utilizada para o desenvolvimento do software do computador. A existência da biblioteca Java Communications API foi o diferencial que definiu o uso do Java. O código-fonte da classe “Serial.class” foi desenvolvido baseado nos exemplos existentes no pacote dessa biblioteca.

Outra dificuldade encontrada foi o dimensionamento do circuito elétrico. Os primeiros testes não estavam apresentando os resultados esperados. Inicialmente, o motivo era o uso de alguns componentes desnecessários que estavam aumentando a resistência interna do circuito. A configuração final do circuito foi obtida após exaustivos testes. A falta de experiência em lidar com a placa de fenolite ilhada, onde é necessário soldar os componentes, gerou a necessidade de pesquisas e demandou um tempo maior não previsto.

O fato de ter sido utilizado o kit de desenvolvimento CW552 do UniCEUB trouxe um beneficio financeiro, uma vez que não foi necessário adquirir o microcontrolador.

Entretanto, se tivesse sido adquirido o microcontrolador, os testes não teriam sido realizados apenas no UniCEUB e teriam sido concluídos com maior rapidez.

7.3 Sugestões para projetos futuros

Como sugestão para projetos futuros, é possível dar continuidade ao projeto efetuando o controle de aparelhos eletrodomésticos, como televisores, aparelhos de microondas ou aparelhos de ar-condicionado. Pode ser analisada a possibilidade de controlar as portas ou janelas da residência.

Como melhoria para o projeto desenvolvido, é sugerida o funcionamento paralelo com outros tipos de hardwares no circuito elétrico, como um interruptor ou um painel de controle. O hardware deve atualizar o estado da iluminação no Simulador de Entrada de Dados.

O desenvolvimento de um sistema de reconhecimento de voz utilizando, por exemplo, redes neurais também é um tema muito interessante.

REFERÊNCIAS

AHMED, Ashfaq. **Eletrônica de Potência.** São Paulo: Prentice Hall, 2000.

Atera Informática **[Home Page].** 2006. Disponível em: <http://www.atera.com.br>. Acesso em: 14 dez. 2006.

Aureside – Associação Brasileira de Automação Residencial **[Home Page].** 2006. Disponível em: <http://www.aureside.org.br>. Acesso em: 29 out. 2006.

BERTOLI, Roberto Angelo. **Eletrônica.** São Paulo: Colégio Técnico de Campinas, 2000.

BERTULANI, Carlos A. **Ondas sonoras [Home Page]**. 2006. Disponível em: <http://www.if.ufrj.br/teaching/fis2/ondas2/ondas2.html>. Acesso em: 15 nov. 2006.

ControlWare Automação. **Manual do Usuário do Kit de Desenvolvimento CW552.** Out. 2001.

DEITEL, H. M. , DEITEL, P. J. **Java, Como Programar.** 4. ed. Porto Alegre: Bookman, 2003.

Eclipse.org **[Home Page].** 2006. Disponível em: <http://www.eclipse.org>. Acessado em: 12 jul. 2006.

Editora Saber **[Home Page].** 2006. Disponível em: <http://www.mecatronicafacil.com.br>. Acesso em: 22 out. 2006.

Fairchild Semiconductor Corporation. **Datashett Optoacoplador 4N25.** Jun. 2005.

Free Serial Port Monitor **[Home Page].** 2006. Disponível em: <http://www.serial-port-monitor.com>. Acesso em: 12 set. 2006.

GUJ – Grupo de Usuários Java **[Home Page].** 2006. Disponível em: <http://www.guj.com.br>. Acesso em: 04 set. 2006.

HP **[Home Page].** 2006. Disponível em: <http://www.hp.com>. Acesso em: 13 nov. 2006.

Java Technology **[Home Page].** 2006. Disponível em: <http://java.sun.com>. Acesso em: 04 set. 2006.

JEROME, Jeffrey. **Emerging Technologies for Independent Living – Report from a Working Conference.** Maryland, 1994

MALVINO, Albert Paul. **Eletrônica: volume 1.** 4. ed. São Paulo: Makron Books, 1995.

NEVES, Rogério Perino de Oliveira. **Introdução aos princípios conexionistas.** Disponível em: <http://www.lsi.usp.br/~rponeves/work/cognition>. Acessado em: 14 dez. 2006.

NICOLOSI, Denys Emílio Campion. **Microcontrolador 8051 Detalhado.** 5. ed. São Paulo: Érica, 2000.

PCCOM **[Home Page].** 2006. Disponível em: <http://www.pccompci.com>. Acesso em: 22 out. 2006.

Rogercom **[Home Page].** 2006. Disponível em: <http://www.rogercom.com>. Acesso em: 11 out. 2006.

SÁ, Maurício Cardoso de. **Programação C para Microcontroladores 8051.** 1. ed. São Paulo: Érica, 2005.

SDCC – Small Device C Compiler **[Home Page].** 2006. Disponível em: <http://sdcc.sourceforge.net>. Acesso em: 19 set. 2006.

Sun Microsystems **[Home Page].** 2006. Disponível em: <http://www.sun.com>. Acesso em: 04 set. 2006.

Superdownloads – Circuit Maker Student Version **[Home Page].** 2006. Disponível em: <http://superdownloads.uol.com.br/download/i9609.htm>. Acesso em: 14 out. 2006.

VASCONCELOS, Laércio. **Hardware Total.** São Paulo: Makron Books, 2002.

APÊNDICE A - CÓDIGO-FONTE – CLASSE

PRINCIPAL.JAVA

```
/** PROJETO FINAL DE CONCLUSÃO DE CURSO
 *  AUTOMAÇÃO RESIDENCIAL POR COMANDO DE VOZ UTILIZANDO MICROCONTROLADOR
 *
 *  Classe Principal.java
 *  Versão Final
 *
 *  >> Funcionalidades:
 *     - tratamento do comando recebido ('Aceso' ou 'Apagado')
 *     - leitura automática do campo comando
 *
 *  >> Data da revisão: 27/11/2006
 *
 *  >> Desenvolvido por Gustavo Gomes de Lucena
 *
 **/

package ProjetoFinal;

// pacotes do núcleo
import java.awt.*;

// pacotes de extensão
import javax.swing.*;


/* PROGRAMA PRINCIPAL */
public class Principal implements Runnable{

   // chama classe Interface
   public Interface layout = new Interface();

   // thread que monitora comandos
   private Thread verificaComando;

   // armazena mensagens do console
   String console = "";

   //contador de comandos
   int contValido   = 0; //comandos válidos
   int contInvalido = 0; //comandos inválidos
   int contAceso    = 0; //comando 'Aceso'
   int contApagado  = 0; //comando 'Apagado'
   int contTimer    = 0; //tempo de preenchimento do campo
   int contLeitura  = 0; //contador de leituras

   /* início dos métodos */
   // escreve na porta serial
   public void escritaSerial(){
      try{
         // caso 'Aceso'
         if (layout.status == "Aceso"){
            layout.serial.EnviarString("c");
         }
         // caso 'Apagado'
         if (layout.status == "Apagado"){
            layout.serial.EnviarString("p");
         }
```

```
      // escreve no console
      console = layout.areaConsole.getText();

      contValido ++;
      layout.areaConsole.setText("" + console +
                "Número de comandos válidos: " + contValido +
                "\n>> Comando '" + layout.status +
                "' enviado para porta " + layout.porta + "!!" +
                "\n---------------------------------------" +
                "---------------------------------------" +
                "--------------------\n");
   } catch (Exception e){
      layout.areaConsole.setText(layout.serial.erroSerial);
   }
}

// leitura na porta serial
public void leituraSerial(){
   // mostra dado recebido
   layout.lblConfirmaRecebido.setText(layout.serial.recebido);

   // contador de leituras
   contLeitura ++;
   layout.lblLeitura.setText("Leituras: " + contLeitura);
}


// monitora comandos recebidos (thread)
public void leituraCampo(){
   try{
      verificaComando = new Thread(this);
      verificaComando.start();
   } catch (Exception e){
      e.printStackTrace();
   }

   // termina a simulação ao fechar a janela
   layout.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

// configura status válido
public void statusValido(int i){

   // chama método que habilita a transmissão
   escritaSerial();

   // apaga campo comando
   layout.txtComando.setText("");
   layout.txtComando.requestFocus();

   switch(i){
      case 1: // aceso

         // mostra status enviado
         layout.lblConfirmaEnviado.setText(layout.status);
         layout.lblConfirmaEnviado.setForeground(Color.BLUE);

         // mostra ícone
         layout.lblIcon.setIcon(layout.icons[1]);

         // incrementa contador
         contAceso ++;
         layout.lblAceso.setText("Aceso: " + contAceso);

         break;
```

```
      case 0: // apagado

        // mostra status enviado
        layout.lblConfirmaEnviado.setText(layout.status);
        layout.lblConfirmaEnviado.setForeground(Color.RED);

        // mostra ícone
        layout.lblIcon.setIcon(layout.icons[0]);

        // incrementa contador
        contApagado ++;
        layout.lblApagado.setText("Apagado: " + contApagado);

        break;
    }
}

// dados estatísticos de confiabilidade
public void atualizaDados(){
   float totalDeComandos = (contAceso+contApagado+contInvalido);
   float confiabilidade = (((totalDeComandos - contInvalido) /
           totalDeComandos) * 100);

   if(totalDeComandos > 0){
      layout.estatistica = "ESTATÍSTICA DE CONFIABILIDADE" +
                "\nAceso = " + contAceso +
                "\nApagado = " + contApagado +
                "\nOutros comandos = " + contInvalido +
                "\n\nConfiabilidade = " + confiabilidade + "%";
   }
   else{
      layout.estatistica = "Nenhum comando recebido!!";
   }
}

/* inicia simulação */
public void run(){
   try{
     while(true){ // loop

        Thread.sleep(500); // sleep em ms (1000 ms = 1 s)

        atualizaDados();

        if(layout.primeiroAcesso > 0 && layout.start == true){
          // chama método leituraSerial
          leituraSerial();
          // recebe comando
          layout.status = layout.txtComando.getText();

          // comandos inválidos
          if(layout.status.length() == 0){
            // campo comando vazio
            contTimer = 0;
          }
          if(layout.status.length() > 0){
            // campo comando preenchido e comando inválido

            // não apagará imediatamente um comando inválido
            // porque pode ser que a palavra esteja sendo escrita
            // no momento da leitura.
            contTimer ++;
```

```
            if (contTimer > 2){
               // após xx ciclos com comando inválido,
               // o campo comando será apagado e ciclo reiniciado
               layout.status = layout.txtComando.getText();
               layout.txtComando.setText("");
               contTimer = 0;
               contInvalido ++;

               console = layout.areaConsole.getText();
               layout.areaConsole.setText("" + console +
                    "Número de comandos inválidos: " + contInvalido +
                    "\n>> ATENÇÃO! Comando '" + layout.status +
                    "' inválido!!"+
                    "\n---------------------------------------" +
                    "---------------------------------------" +
                    "--------------------\n");
            }
         }

         // comandos válidos
         // caso 'Aceso'
         if (layout.status.equalsIgnoreCase("aceso")  ||
            layout.status.equalsIgnoreCase(" aceso") ||
            layout.status.equalsIgnoreCase("aceso ") ||
            layout.status.equalsIgnoreCase("acesa")  ||
            layout.status.equalsIgnoreCase(" acesa") ||
            layout.status.equalsIgnoreCase("acesa ") ||
            layout.status.equalsIgnoreCase("acesos") ||
            layout.status.equalsIgnoreCase(" acesos")||
            layout.status.equalsIgnoreCase("acesos ")||
            layout.status.equalsIgnoreCase("acesas") ||
            layout.status.equalsIgnoreCase(" acesas")||
            layout.status.equalsIgnoreCase("acesas ")
            ){
               layout.status = "Aceso";

               // chama método statusValido
               statusValido(1); // 1: Aceso
         }
         // caso 'Apagado'
         if (layout.status.equalsIgnoreCase("apagado")  ||
            layout.status.equalsIgnoreCase(" apagado") ||
            layout.status.equalsIgnoreCase("apagado ") ||
            layout.status.equalsIgnoreCase("apagada")  ||
            layout.status.equalsIgnoreCase(" apagada") ||
            layout.status.equalsIgnoreCase("apagada ") ||
            layout.status.equalsIgnoreCase("apagados")  ||
            layout.status.equalsIgnoreCase(" apagados") ||
            layout.status.equalsIgnoreCase("apagados ") ||
            layout.status.equalsIgnoreCase("apagadas")  ||
            layout.status.equalsIgnoreCase(" apagadas") ||
            layout.status.equalsIgnoreCase("apagadas ")
            ){
               layout.status = "Apagado";
               // chama método statusValido
               statusValido(0); // 0: Apagado
         }
      }
   }
  } catch (Exception e){
    e.printStackTrace();
  }
}
```

```
   /* método principal */
   public static void main(String args[]) {
      // chama thread leituraCampo
      Principal application = new Principal();
      application.leituraCampo();
   }

}
```

APÊNDICE B - CÓDIGO-FONTE – CLASSE SERIAL.JAVA

```
/** PROJETO FINAL DE CONCLUSÃO DE CURSO
 *  AUTOMAÇÃO RESIDENCIAL POR COMANDO DE VOZ UTILIZANDO MICROCONTROLADOR
 *
 *  Classe Serial.java
 *  Versão Final
 *
 *  >> Funcionalidades:
 *     - configura comunicação serial
 *
 *  >> Data da revisão: 22/10/2006
 *
 *  >> Desenvolvido por Gustavo Gomes de Lucena
 *
 */

package ProjetoFinal;

// pacotes de comunicação serial
import javax.comm.*;

// pacotes que permitem entreda e saída
import java.io.*;


public class Serial implements Runnable, SerialPortEventListener {

   /* atributos de controle */
   private   String Porta;
   public    String Dadoslidos;
   public    String configuracao;
   public    String recebido = "";
   public   String erroSerial = "";

   protected int numBytes;
   private   int baudrate;
   private   int timeout;

   private   CommPortIdentifier cp;

   protected SerialPort porta;

   // método para realizar saída baseada em bytes
   private   OutputStream escrita;
   // método para realizar entrada baseada em bytes
   private   InputStream leitura;

   private   Thread threadLeitura;

   protected boolean IDPortaOK; // true = porta existe
   protected boolean PortaOK;// true = porta aberta
   public boolean PortaAberta;

   /* recebe objeto da classe Interface.java
   /* com detalhes de configuração */
   public Serial(String p, int b, int t) {
      this.Porta = p;
      this.baudrate = b;
      this.timeout = t;
   }
```

```
/* Abre a comunicação da porta */
public void AbrirPorta() {
   // Obtém o ID da porta
   try {
      cp = CommPortIdentifier.getPortIdentifier(Porta);
      if (cp == null) {
         IDPortaOK = false;
      }
      IDPortaOK = true;
   } catch (Exception e) {
      IDPortaOK = false;
   }
   // Abre porta serial para comunicação
   try {
      porta = (SerialPort) cp.open("Simulador de Entrada de Dados",
                                   timeout);
      PortaOK = true;

      // configuração dos parâmetros de porta
      porta.setSerialPortParams(baudrate, porta.DATABITS_8,
           porta.STOPBITS_2, porta.PARITY_NONE);

      // string configuração com informações da configuração
      configuracao =  "Baudrate: " + baudrate + "  /  " +
                "Timeout: "  + timeout + "  /  " +
                "Databits: " + porta.DATABITS_8 + "  /  " +
                "Stopbits: " + porta.STOPBITS_2 + "  /  " +
                "Paridade: " + porta.PARITY_NONE;
      PortaAberta = true;
   } catch (PortInUseException e) {
      PortaOK = false;
      erroSerial = ">> Erro ao abrir a porta serial!!" +
        "\nMotivo: A porta serial está sendo utilizada" +
        " por outro aplicativo" +
        "\n---------------------------------------" +
        "---------------------------------------" +
        "--------------------\n";
   } catch (NullPointerException e) {
      PortaOK = false;
      erroSerial = ">> Erro ao abrir a porta serial!!" +
        "\nMotivo: A porta serial não está disponível" +
        "\n---------------------------------------" +
        "---------------------------------------" +
        "--------------------\n";
   } catch (Exception e) {
      PortaOK = false;
      erroSerial = ">> Erro ao abrir a porta serial!!" +
        "\nMotivo: " + e +
        "\n---------------------------------------" +
        "---------------------------------------" +
        "--------------------\n";
   }
   // habilita escrita e leitura
   try {
       escrita = porta.getOutputStream();
       leitura = porta.getInputStream();
   } catch (IOException e) {
       porta.close();
      erroSerial = ">> Erro ao habilitar leitura e escrita!!" +
        "\nMotivo: " + e +
        "\n---------------------------------------" +
        "---------------------------------------" +
        "--------------------\n";
   }
}
```

```
/* habilita leitura de dados */
public void LerDados() {
   try {
      porta.addEventListener(this);
   } catch (Exception e) {
      erroSerial = ">> Erro ao criar listener!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
   porta.notifyOnDataAvailable(true);
   try {
      threadLeitura = new Thread(this);
      threadLeitura.start();
   } catch (Exception e) {
      erroSerial = ">> Erro ao iniciar leitura!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
}

/* envia string para porta serial */
public void EnviarString(String msg) {
   try {
      escrita.write(msg.getBytes());
      Thread.sleep(100);
      escrita.flush();
   } catch (Exception e) {
      erroSerial = ">> Houve um erro durante o envio!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
}

/* método RUN da thread de leitura */
public void run() {
   try {
      Thread.sleep(5000);
   } catch (Exception e) {
      erroSerial = ">> Erro!!" +
         "\nMotivo: " + e +
         "\n----------------------------------------" +
         "----------------------------------------" +
         "--------------------\n";
   }
}

/* gerenciador de eventos de leitura */
public void serialEvent(SerialPortEvent ev) {
   switch (ev.getEventType()) {
   case SerialPortEvent.BI:
   case SerialPortEvent.OE:
   case SerialPortEvent.FE:
   case SerialPortEvent.PE:
   case SerialPortEvent.CD:
   case SerialPortEvent.CTS:
   case SerialPortEvent.DSR:
   case SerialPortEvent.RI:
   case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
      break;
```

```
      case SerialPortEvent.DATA_AVAILABLE:
        byte[] bufferLeitura = new byte[1];
        try {
          while (leitura.available() > 0) {
            numBytes = leitura.read(bufferLeitura);
          }
          String Dadoslidos = new String(bufferLeitura);
          if (bufferLeitura.length == 0) {
          } else {
            recebido = Dadoslidos;
          }
        } catch (Exception e) {
          erroSerial = ">> Erro durante a leitura!!" +
            "\nMotivo: " + e +
            "\n----------------------------------------" +
            "----------------------------------------" +
            "--------------------\n";
        }
        break;
      }
    }

    /* fecha a conexão */
    public void FecharCom() {
      try {
        EnviarString("p");
        porta.close();
      } catch (Exception e) {
        erroSerial = ">> Erro ao fechar conexão!!" +
          "\nTipo de erro: " + e +
          "\n----------------------------------------" +
          "----------------------------------------" +
          "--------------------\n";
      }
    }
}
```

APÊNDICE C - CÓDIGO-FONTE – CLASSE LAYOUT.JAVA

```
/** PROJETO FINAL DE CONCLUSÃO DE CURSO
 *  AUTOMAÇÃO RESIDENCIAL POR COMANDO DE VOZ UTILIZANDO MICROCONTROLADOR
 *
 *  Classe Interface.java
 *  Versão Final
 *
 *  >> Funcionalidades:
 *     - cria interface gráfica
 *
 *  >> Data da revisão: 27/11/2006
 *
 *  >> Desenvolvido por Gustavo Gomes de Lucena
 *
 */

package ProjetoFinal;

// pacotes do núcleo
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

// pacotes de extensão
import javax.swing.*;
import javax.swing.border.LineBorder;

public class Interface extends JFrame implements ActionListener{

   /* atributos de controle */
   // porta serial utilizada
   String porta = "COM1";

   // informa se foi leitura foi iniciada com sucesso
   public boolean start;

   // no primeiro acesso é mostrada a configuração
   // da porta serial no console
   public int primeiroAcesso = 0;

   // armazena status do campo 'txtComando'
   public String status = "";

   // estatística de confiabilidade
   public String estatistica;

   // define atributos dos componentes e
   // anexa ao painel de comando
   public Container c = getContentPane();

   /* atributos da GUI (Interface Gráfica com o Usuário) */
   // campo que recebe o comando
   public JTextField txtComando;

   // box com comandos habilitados
   public JTextArea areaBox;

   // console
   public TextArea areaConsole;
```

```
// labels
public JLabel lblTitulo;
public JLabel lblComando;
public JLabel lblStatusEnviado;
public JLabel lblStatusRecebido;

// contadores
public JLabel lblAceso;
public JLabel lblApagado;
public JLabel lblLeitura;

// confirmação de status
public JLabel lblConfirmaEnviado;
public JLabel lblConfirmaRecebido;

// botões
public JButton botaoIniciar;
public JButton botaoParar;
public JButton botaoFechar;

// ícone status lâmpada
public JLabel lblIcon;
public String imagens [] = {"Imagens/lampada_acesa.gif",
                    "Imagens/lampada_apagada.gif"};
public Icon icons [] = {new ImageIcon (imagens [1]),
                 new ImageIcon (imagens [0])};

/* Configura a GUI */
public Interface() {
  // barra de título
  super("Simulador de Entrada de Dados - Versão Final");

  /* distâncias */
  // janela
  int xPopup = 447;
  int yPopup = 640;

  // colunas
  int col0 = 15, col1 = 35, col2 = 125,
    col3 = 175, col4 = 235, col5 = 325;
  int colIcone = xPopup/2 - 20;

  // linhas
  int ln1 = 10, ln2 = ln1+50, ln3 = ln2+40, ln4 = ln3+40,
    ln5 = ln4+40, ln6 = ln5+40, ln7 = ln6+40, ln8 = ln7+40,
    ln9 = ln8+40, ln10 = ln9+40, ln11 = ln10+40, ln12 = ln11+40,
    ln13 = ln12+40, ln14 = ln13+40, ln15 = ln14+40;

  // tamanho
  int xTexto = 100, xLabel = 190, xBox = 200, xTitulo = 420,
    xConsole = 430;

  // altura
  int yBox = 70, yConsole = 200;

  /* layout */
  // título
  lblTitulo = new JLabel("Automação Residencial por Comando de Voz " +
                 "Utilizando Microcontrolador");
  confLabel(c, lblTitulo, ln1, col0, xTitulo);

  // campo texto - comando
  lblComando = new JLabel("Informe um comando  ",
                       SwingConstants.RIGHT);
  confLabel(c, lblComando, ln2, col1, xLabel);
```

```
txtComando = new JTextField(SwingConstants.LEFT);
confTexto(c, txtComando, ln2, col4, xTexto);
txtComando.setEnabled(false);

// status recebido
lblStatusRecebido = new JLabel("Status recebido da porta serial  ",
                    SwingConstants.RIGHT);
confLabel(c, lblStatusRecebido, ln3, col1, xLabel);

lblConfirmaRecebido = new JLabel("", SwingConstants.LEFT);
confLabel(c, lblConfirmaRecebido, ln3, col4, xLabel);

// status enviado
lblStatusEnviado = new JLabel("Status enviado para porta serial  ",
                    SwingConstants.RIGHT);
confLabel(c, lblStatusEnviado, ln4, col1, xLabel);

lblConfirmaEnviado = new JLabel("", SwingConstants.LEFT);
confLabel(c, lblConfirmaEnviado, ln4, col4, xLabel);

// ícones
lblIcon = new JLabel();
c.add(lblIcon);
lblIcon.setBounds(colIcone, ln5, 200,50);

// box
areaBox = new JTextArea("            Comandos Permitidos:" +
       "\n-------------------------------------------------" +
       "\n > ACESO:         Acende a lâmpada" +
       "\n > APAGADO:    Apaga a lâmpada");
confArea(c, areaBox, ln7, col1+90, xBox, yBox);
areaBox.setBackground(Color.ORANGE);
areaBox.setForeground(Color.BLACK);
areaBox.setEditable( false );

// console
areaConsole = new TextArea("");
confTArea(c,areaConsole, ln9, 6, xConsole, yConsole);
areaConsole.setEditable(false);

// contadores
lblAceso = new JLabel("Aceso:", SwingConstants.LEFT);
confLabel(c, lblAceso, ln14, col1, xLabel);
lblAceso.setForeground(Color.BLUE);

lblApagado = new JLabel("Apagado:", SwingConstants.LEFT);
confLabel(c, lblApagado, ln14, col3, xLabel);
lblApagado.setForeground(Color.RED);

lblLeitura = new JLabel("Leitura:", SwingConstants.LEFT);
confLabel(c, lblLeitura, ln14, col5, xLabel);

// botão 'Iniciar'
botaoIniciar = new JButton ("Iniciar");
c.add(botaoIniciar);
botaoIniciar.setBounds(col2, ln15, 80, 20);
botaoIniciar.addActionListener(this);

// botão 'Parar'
botaoParar = new JButton ("Parar");
c.add(botaoParar);
botaoParar.setBounds(col4, ln15, 80, 20);
botaoParar.setEnabled(false);
botaoParar.addActionListener(this);
```

```
    // botão 'Fechar'
    botaoFechar = new JButton ("Fechar");
    c.add(botaoFechar);
    botaoFechar.setBounds(col3, ln15, 80, 20);
    botaoFechar.setVisible(false);
    botaoFechar.addActionListener(this);

    // tamanho da janela
    setSize(xPopup, yPopup);
    // seta visibilidade
    setVisible(true);
    // proibe redimensionar
    setResizable(false);
    // seta localização (horizontal, vertical)
    setLocation(400, 50);

    // foco inicial no botão 'Iniciar'
    botaoIniciar.requestFocus();

}
/* Fim da configuração da GUI */


/* Métodos de configuração do layout */
public void confArea(Container ctn, JTextArea area,
          int topo, int esq, int largura, int altura)     {
    // (distância esquerda, distância topo, largura, altura)
    area.setBounds(esq, topo, largura, altura);
    area.setBorder(LineBorder.createGrayLineBorder());
    ctn.setLayout(null);
    ctn.add(area);

    return;
}

public void confTArea(Container ctn, TextArea area,
          int topo, int esq, int largura, int altura)     {
    area.setBounds(esq, topo, largura, altura);
    ctn.setLayout(null);
    ctn.add(area);

    return;
}

public void confLabel(Container ctn, JLabel label,
          int topo, int esq, int largura)     {
    label.setBounds(esq, topo, largura, 20);
    ctn.setLayout(null);
    ctn.add(label);

    return;
}

public void confTexto(Container ctn, JTextField campo,
          int topo, int esq, int largura)     {
    campo.setBounds(esq, topo, largura, 20);
    ctn.add(campo);

    return;
}
/* Fim dos métodos de configuração do layout */
```

```
/* inicia conexão */
// Porta: definida anteriormente
// Baudrate: tempo de transmissão de cada bit (bauds)
// Timeout: tempo de espera para estabelecer da conexão (ms)
Serial serial = new Serial(porta, 9600, 2000);

/* método que abre a porta, inicia leitura dos dados
/* e inicia a simulação com o status 'Apagado' */
public void estabeleceConexao(){
   try {
     serial.AbrirPorta();

     // contador do número de acessos
     primeiroAcesso ++;

     // mostra configuração da porta serial no console
     if (primeiroAcesso == 1){
       String configuracao = serial.configuracao;

       areaConsole.setText(">> Porta serial " + porta +
         " aberta para comunicação!! \n" + configuracao +
         "\n---------------------------------------" +
         "---------------------------------------" +
         "--------------------\n");

       // inicia com o status 'Apagado'
       serial.EnviarString("p");
       lblConfirmaEnviado.setText(status);
       lblConfirmaEnviado.setForeground(Color.RED);
       lblIcon.setIcon(icons[0]);
     }

     // inicia leitura
     serial.LerDados();

     // leitura iniciada com sucesso
     start = true;
   }
   catch (Exception e){
     String console = areaConsole.getText();
     areaConsole.setText(console + serial.erroSerial);

     botaoIniciar.setVisible(false);
     botaoParar.setVisible(false);

     botaoFechar.setVisible(true);
     botaoFechar.requestFocus();
   }
}

/* executa a ação do botão pressionado */
public void actionPerformed (ActionEvent click){
   // botão 'Iniciar'
   if (click.getActionCommand() == "Iniciar"){
     // estabelece conexão
     estabeleceConexao();

     // caso tenha estabelecido o primeiro acesso
     if(primeiroAcesso > 0 && start == true){
       // desabilita botão 'Iniciar'
       botaoIniciar.setEnabled(false);

       // habilita botão 'Parar'
       botaoParar.setEnabled(true);
       botaoParar.setText("Parar");
```

```
            // habilita campo 'txtComando'
            txtComando.setEnabled(true);
            txtComando.requestFocus();
          }
        }
        // botão 'Parar'
        if (click.getActionCommand() == "Parar"){
          // fecha porta serial
          serial.FecharCom();

          // apaga lâmpada
          lblIcon.setIcon(icons[0]);
          lblConfirmaEnviado.setText("");

          // confirma lâmpada apagada
          lblConfirmaRecebido.setText(serial.recebido);

          // leitura parada
          start = false;

          // apaga e desabilita campo 'txtComando'
          txtComando.setText("");
          status = "";
          txtComando.setEnabled(false);

          // habilita botão 'Iniciar'
          botaoIniciar.setEnabled(true);
          botaoIniciar.requestFocus();

          // desabilita botão 'Parar'
          botaoParar.setText("Fechar");
        }
        // botão 'Fechar'
        if (click.getActionCommand() == "Fechar"){
          // mostra estatística de confiabilidade
          JOptionPane.showMessageDialog(null, estatistica);
          // fecha o simulador
          System.exit(1);
        }
      }

}
```

APÊNDICE D - CÓDIGO-FONTE – CONTROLE.C

```
/** PROJETO FINAL DE CONCLUSÃO DE CURSO
 *  AUTOMAÇÃO RESIDENCIAL POR COMANDO DE VOZ UTILIZANDO MICROCONTROLADOR
 *
 *  Controle.c
 *  Versão Final
 *
 *  >> Funcionalidades:
 *     - mostra no LCD;
 *     - aciona circuito do microcontrolador
 *
 *  >> Data da revisão: 27/11/2006
 *
 *  >> Desenvolvido por Gustavo Gomes de Lucena
 *
 */


#include "8051.h"
#include "at89s8252.h"


// linhas do LCD
#define linha1 0x80
#define linha2 0xc0

// ON = aceso ; OFF = apagado
#define ON  1
#define OFF 0

//pinos de controle - conector J4
sbit at 0xC0 P40; //alimenta circuito do microcontrolador


// início das rotinas do LCD
xdata at 0x3801 unsigned char Lcd_dado;
xdata at 0x3800 unsigned char Lcd_cont;

void wr_ctr_lcd(unsigned char a)
{
    int i;
    Lcd_cont = a;
    for (i=1;i!=1000;i++);
}

void wr_lcd(unsigned char a)
{
    int i;
    Lcd_dado = a;

    for (i=1;i!=100;i++);
}

void ini_lcd(void)
{
    wr_ctr_lcd(0x38);
    wr_ctr_lcd(0x06);
    wr_ctr_lcd(0x0E);
    wr_ctr_lcd(0x01);
}
```

```
void lcd_str(char *s)
{
    do wr_lcd(*s);
    while (*++s);
}

void lcd_hex(unsigned char i)
{
    char s;
    char ii;
    ii = (i >> 4) & 0x0F;
    s = ii < 0x0A ? (ii+'0') : (ii+'7');
    wr_lcd(s);
    ii = i & 0x0F;
    s = ii < 0x0A ? (ii+'0') : (ii+'7');
    wr_lcd(s);
}

void lcd_bcd(unsigned char a)
{
    a=a%100;
    wr_lcd('0'+a/10);
    wr_lcd('0'+a%10);
}
void goto_lcd(unsigned char l, unsigned char c)
{
    unsigned char a;
    if (l==1){
        a = linha1;
    }
    if (l==2){
        a = linha2;
    }
    wr_ctr_lcd(a+c-1);
}
// fim das rotinas LCD


// rotinas da comunicação serial
// armazena valor recebido
int dadoRecebido;

// armazena caractere a ser enviado
char statusEnviado;

// contador de comandos recebidos
unsigned char contRecebido = 0;

// Função para configurar a comunicação serial
void confSerial()
{
     // Modo 1 e Habilita Recepção de Dados
     SCON=0x50;

     // Seta bit de controle
     TCLK=1;

     // Seta bit de controle
     RCLK=1;

     // Habilita operação do periférico
     TR2=1;
}
```

```
// Função que envia confirmação do dado recebido
void enviaStatus(int *dado)
{
    // Configura a comunicação serial
    confSerial();

    // Recebe o valor contido no endereço apontado pelo ponteiro
    SBUF=*dado;

    // Inicia a transmissão automaticamente
    while(!TI); // Quando TI = 1, terminou a transmissão

    // Limpa TI para permitir nova transmissão
    TI=0;

}

// Função que recebe os dados da serial
void recebeDado()
{
    while(!RI); // Quando RI = 1, terminou a recepção

    // Salva o que foi recebido na variável
    dadoRecebido=SBUF;

    // Limpa RI para permitir nova recepção
    RI = 0;

    // Envia confirmação de leitura
    statusEnviado = dadoRecebido;
    enviaStatus(&statusEnviado);

    // Incrementa contador de dados recebidos
    contRecebido ++;

}


// programa principal
void main(){

    // Configura comunicação serial
    confSerial();

    // habilita a interrupção serial
    ES = 1;

    // inicialização do LCD
    ini_lcd();

    goto_lcd(1, 1);
    lcd_str("Automacao Residencial por Comando de Voz");
    goto_lcd(2, 1);
    lcd_str("Autor: Gustavo Gomes de Lucena");

    // circuito do microcontrolador
    P40 = OFF;

    while (1) { // Aguarda recebimento de dados

        // Chama função recebeDado
        recebeDado();

        // Apaga LCD
        ini_lcd();
```

```
            // escreve no LCD
            goto_lcd(1, 1);
            lcd_str("Contador de comandos recebidos:");
            goto_lcd(1, 33);
            lcd_bcd(contRecebido);
            goto_lcd(2, 1);
            lcd_str("Status:");

            if (dadoRecebido == 'c'){
                // c = aceso
                goto_lcd(2, 9);
                lcd_str("Aceso");

                P40 = ON;

            }
            if (dadoRecebido == 'p'){
                // p = apagado
                goto_lcd(2, 9);
                lcd_str("Apagado");

                P40 = OFF;

            }
        }
    }
```

APÊNDICE E - FOTO DO PROTÓTIPO

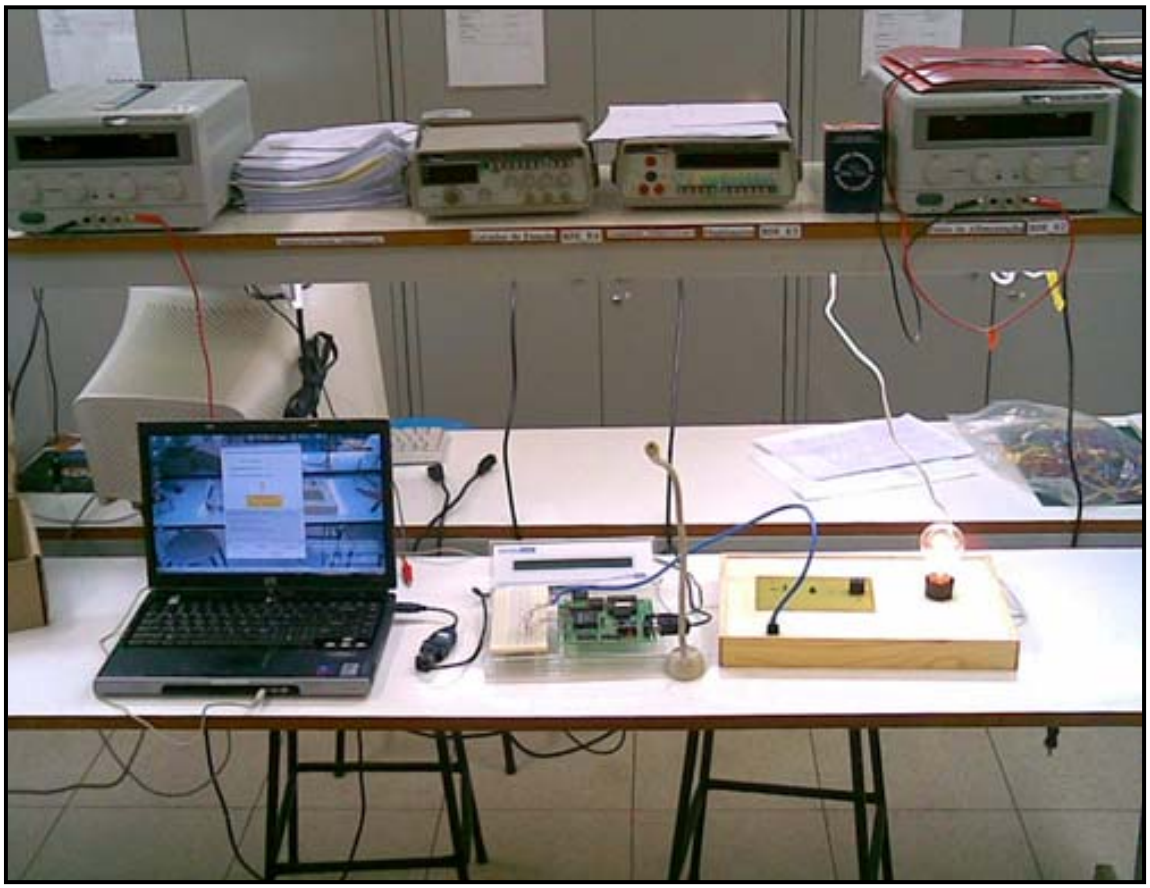

Figura E.1 Foto do protótipo.ótipo.