UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM COMPUTADOR DE BORDO AUTOMOTIVO BASEADO EM PC LINUX

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

CRISTIANO FREESE

BLUMENAU, JUNHO/2003

2003/1-12

PROTÓTIPO DE UM COMPUTADOR DE BORDO AUTOMOTIVO BASEADO EM PC LINUX

CRISTIANO FREESE

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Miguel A. Wisintainer — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Miguel A. Wisintainer

_____

Prof. Antonio Carlos Tavares

_____

Prof. Jomi Fred Hübner

AGRADECIMENTOS

Gostaria de agradecer aos meus pais Rubens Freese e Beatriz Bonatti Freese, pelo exemplo de determinação, pela sabedoria e incentivo aos estudos.

Agradeço também de forma especial ao meu orientador Miguel Alexandre Wisintainer, que com todo o seu conhecimento tecnológico e científico, contribuiu de forma decisiva e objetiva para que chegássemos ao objetivo final.

Devo também remeter um agradecimento ao professor Antonio Carlos Tavares pelos importantes esclarecimentos a cerca de comunicação inter-processos do Linux.

Faço também um agradecimento a todos os professores da academia, por terem dado suas significativas parcelas de conhecimento para a minha formação acadêmica, que no final deste trabalho se uniram e consolidaram o objetivo final.

RESUMO

Este trabalho demonstra a integração entre tipos diferentes de *hardware* e *software*, representando funcionalidades de um computador de bordo automotivo. Funções como indicação de temperatura ambiente, velocidade veicular, *status* de carga da bateria, interpretação de comandos via controle remoto e interface homem/máquina são desempenhadas por um *hardware* microcontrolado da família PIC denominado Interface, enquanto que funções como registro de excessos de velocidade e reprodução de músicas em formato MP3 são desempenhadas por um *hardware* padrão PC denominado *PC*, utilizando um novo formato de "placas-mãe" denominado Mini-ITX. O desenvolvimento deste sistema proporcionou a utilização das linguagens Basic para o *software* da Interface, linguagem C para o *software* do *PC* e ambiente de desenvolvimento Delphi 5.0 para o *software* de transferência de arquivos MP3 via FTP. Demonstrou também a potencialidade do sistema operacional Linux utilizando-se do seu mecanismo de comunicação inter-processos, através de *pipes*, bem como a comunicação serial entre *PC* e Interface.

ABSTRACT

This work demonstrates the integration between different types of hardware and software, representing funcionalities of a computer of automotive board.

Functions as indication of ambient temperature, vehicular speed, status of battery load, commands interpretation through remote control and interface man/machine are carried out by a micro controlled hardware of the PIC family denominated Interface, while functions as registration of speed excesses and reproduction of music in format MP3 are carried out by a PC standard hardware denominated PC, using a new “plate-mother” format denominated Mini-ITX.

The development of this system provided the use of the language Basic for the Interface software, language C for the PC software and development atmosphere Delphi 5.0 for the MP3 file transference software through FTP.

It also demonstrated the potentiality of the Linux operating system using its mechanism of inter-processes communication, through pipes, as well as the serial communication between PC and Interface.

LISTA DE ILUSTRAÇÕES/FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 Introdução

Pessoas que utilizam freqüentemente veículos para trabalhar, principalmente aqueles que realizam longas viagens, sentem-se muito desgastados ao final de suas viagens. Uma forma de amenizar este desgaste, seria com a obtenção de recursos de entretenimento e informações ambientais e de segurança no trânsito ao motorista, dados obtidos por um sistema computacional embarcado, conhecido como computador de bordo.

Segundo JcOnline (1999), os computadores de bordo funcionam por meio de sensores eletrônicos e vem sendo aplicado com cada vez mais freqüência, não só em veículos pequenos, mas também em veículos médios e grandes e tem como objetivo principal fornecer recursos que auxiliem na dirigibilidade e condução dos mesmos, através de consulta visual a informações específicas e gerais como temperatura, data e hora, velocidade, nível de óleo, nível de combustível, sensor de estacionamento e outros.

Atualmente, automóveis já dispõem de computadores de bordo capazes de controlar vários subsistemas do veículo e informar o estado geral de certos sistemas ao motorista (através de luzes no painel) e aos mecânicos através de conexões a computadores especiais. Estes sistemas possuem diversas particularidades, como exemplo, o fato de já virem instalados originalmente nos veículos, atendendo a necessidades específicas previamente mencionadas. Devido a automatização dos processos de fabricação, atualmente os veículos possuem uma padronização de forma que até versões intermediárias de veículos possuam suporte para os computadores de bordo originais, caso seja requerido.

Os avanços rápidos e contínuos na eletrônica e tecnologia da computação são um grande elemento impulsionador dos novos conceitos em controle de tráfego viário. “Veículos com computadores de bordo e comunicadores poderão receber do controle de trânsito central instruções sobre o melhor caminho até o destino final. O computador de bordo também poderá informar ao computador central o seu tempo de viagem e velocidade para ser usado como parte da informação a ser processada. Em sistemas ainda mais avançados, a temporização dos semáforos será coordenada instantaneamente pelas informações recebidas dos veículos próximos" (ABRAMCET, 2002).

Existem também os computadores de bordo comerciais, que focam a sua atuação principalmente em outras particularidades de um veículo, como por exemplo, o controle de uma frota de veículos de uma empresa, registrando excessos de velocidade, má condução, desgaste excessivo dos freios, etc ... com comunicação via satélite com a empresa proprietária do mesmo. Estes modelos de computador de bordo, possuem recursos de comunicação distintos, dentre os quais o GPS, tornou-se o mais popular. O novo desafio está sendo implementar um servidor *web* dentro do computador de bordo de forma totalmente embarcada.

Além de todos os detalhes abordados, acrescentam-se ainda funções de entretenimento com recursos multimídia, tais como, *players* de MP3, DVD, VCD interligados a *displays* LCD TFT de alta resolução.

O sistema proposto neste trabalho visa agregar funcionalidades dos computadores de bordo originais e comerciais com recursos multimídia, principalmente pelo fato do mesmo trabalhar com o conceito de compressão de áudio no formato MP3, substituindo os tradicionais CD *Players*, principalmente devido a capacidade de armazenamento e gerenciamento das músicas.

Além deste recurso o sistema possui a indicação de data e hora, temperatura ambiente e interna, controle de velocidade do veículo com registro de excessos (tacógrafo) e monitoramento de carga da bateria para evitar que o veículo fique sem ignição, devido principalmente á utilização da função MP3 com o veículo desligado. O sistema tem um funcionamento compartilhado por uma interface com *hardwar*e microcontrolado e um PC rodando o sistema operacional Linux. A utilização do sistema operacional Linux para o PC atribui-se pelo fato de que o mesmo arquiteturalmente, é bastante estável e seguro, segundo Anunciação (1999, p.34), além de ser *freeware*.

Este sistema nos permite utilizar diversos conceitos e áreas de conhecimento da computação, destacando-se arquitetura de *hardware* com microcontroladores e computadores pessoais (PC), comunicação de dados serial RS232, infravermelho Norma RC5 e FTP, sistema operacional Linux e integração das linguagens de programação Basic, C e ambiente de desenvolvimento Delphi.

Este trabalho visa também demonstrar uma nova tecnologia de placas-mãe denominado Mini-ITX, que representa um novo formato no segmento, criado pela Via

Technologies, que possui alta integração de periféricos, baixo consumo de energia, baixo custo, tamanho extremamente reduzido (170x170mm) e é ideal para aplicações embarcadas (VIA, 2002).

1.1 Objetivos

O objetivo principal consiste no desenvolvimento de um *hardware* microcontrolado que crie uma interface com o *hardware* do PC e com os diversos dispositivos envolvidos para a composição do sistema descrito.

Os objetivos específicos do trabalho são:

a) execução de arquivos formato MP3, ao invés do formato CDDA;

b) uso de transdutores para as mais diversas funcionalidades, tais como temperatura interna, externa, velocidade e comandos via infravermelho;

c) controle de velocidade, emissão de alertas e registro de excessos;

d) desenvolvimento de uma interface *microcontrolada*;

e) elaboração de compartimento acrílico ou metálico para o alojamento dos dispositivos envolvidos no sistema, em nível *comercial*.

1.2 Estrutura do trabalho

No capítulo 1 é apresentada a introdução, o objetivo principal e os objetivos secundários deste trabalho, assim como a presente estrutura do trabalho.

No capítulo 2 será explanado o *hardware* da interface apresentando os componentes eletrônicos utilizados e o compilador utilizado pelo microcontrolador.

No capítulo 3 será explanado o *hardware* do PC apresentando informações técnicas sobre a placa-mãe ITX e testes de desempenho de *hardware*.

No capítulo 4 será explanado o sistema operacional Linux, e seus recursos utilizados para o desenvolvimento do protótipo. Serão abordados processos UNIX, programas Linux para aplicações MP3, comunicação serial e outros.

No capítulo 5 será explanada a comunicação de dados, destinada a apresentação da comunicação serial RS232, comunicação infravermelho norma RC5 e protocolo FTP.

No capítulo 6 são apresentadas informações técnicas sobre o MP3, incluindo informações sobre os formatos existentes e ID3 *Tags*.

No capítulo 7 é descrito todo o protótipo do computador de bordo desenvolvido, com ênfase para a especificação e implementação do *software* da interface e do PC, e a integração física dos mesmos com diversas ilustrações do resultado final.

No capítulo 8 é apresentada a conclusão e sugestões para trabalhos futuros.

2 Hardware Interface

2.1 Microcontrolador PIC16F877

Este microcontrolador pertence a família PIC16C7XX/F87X da *Microchip*. Esta família possui microcontroladores em encapsulamentos de 18 a 44 pinos com uma vasta gama de opções de integração de periféricos. Possui também instruções de 14 bits, 4 a 8 canais de A/D de 8/10 bits, capacidade de gerenciamento de interrupções, várias interfaces seriais, módulos de captura, PWM e comparadores, detecção de *Brown-Out* e pilha com até 8 níveis. Os microcontroladores PIC16F87X possuem memória de programa FLASH, podem ser reprogramados com baixas tensões e são ideais para aplicações de segurança e sensoreamento remoto para comando de motores e aplicações automotivas de alta velocidade de processamento (MICROCHIP, 2003).

As principais características do microcontrolador PIC16F877 são:

a) CPU RISC de alta performance;

b) Set de 35 instruções *Assembly*;

c) Todas as instruções gastam um ciclo de *clock*, salvo os saltos de programa, que gastam dois ciclos;

d) Velocidade de processamento de até 20Mhz (200ns por ciclo de *clock*);

e) Memória de programa FLASH com 143368 bytes;

f) Memória de dados RAM com 368 bytes;

g) Memória de dados EEPROM com 256 bytes;

h) Pinagem compatível com o PIC16C73B/74B/76/77;

i) 14 fontes de interrupção;

j) 8 níveis de empilhamento (*STACK REGISTER*);

k) Modos de endereçamento relativo, direto e indireto;

l) *Power-On Reset* (POR);

m) Temporizador *Power-Up* (PWRT) e oscilador *Start-Up* (OST);

n) Temporizador *Watchdog* (WDT) com oscilador RC embutido;

o) Código de proteção programável;

p) Modo *SLEEP*;

q) Opções selecionáveis de oscilador;

r) Baixo consumo, alta velocidade de processamento com tecnologia CMOS FLASH/EEPROM;

s) Compatível com programação ISCP, via dois pinos;

t) Programação serial *In-Circuit* com apenas 5V;

u) Processador de escrita/leitura para memória de programa;

v) Ampla faixa de tensão de operação : 2.0V a 5.5V;

w) Alta corrente de *Sink* e *Source* : 25mA;

x) Faixas de temperatura de trabalho comercial, industrial e estendido;

y) Baixo consumo : < 0,6mA – 3V / 4Mhz, 20µA – 3V / 32Khz, < 1µA (stand by);

z) *Timer*0 – contador/temporizador de 8 bits com pré-escala de 8 bits;

aa) *Timer*1 – contador/temporizador de 16 bits com pré-escala, podendo ser incrementado durante *SLEEP* via *clock* externo;

bb) *Timer*2 – contador/temporizador de 8 bits com registrador de período, pré-escala e pós-escala;

cc) 2 módulos PWM (resolução máxima de 10 bits), comparador (máxima de 16 bits com resolução de 200ns), Captura (máxima de 16 bits com resolução de 12,5ns);

dd) 8 conversores A/D de 10 bits, Multi-Canal;

ee) Porta serial síncrona (SSP) com SPI (Mestre) e I2C (Mestre/Escravo);

ff) Receptor e Transmissor Universal Assíncrono e Síncrono com detecção do *bit* de paridade;

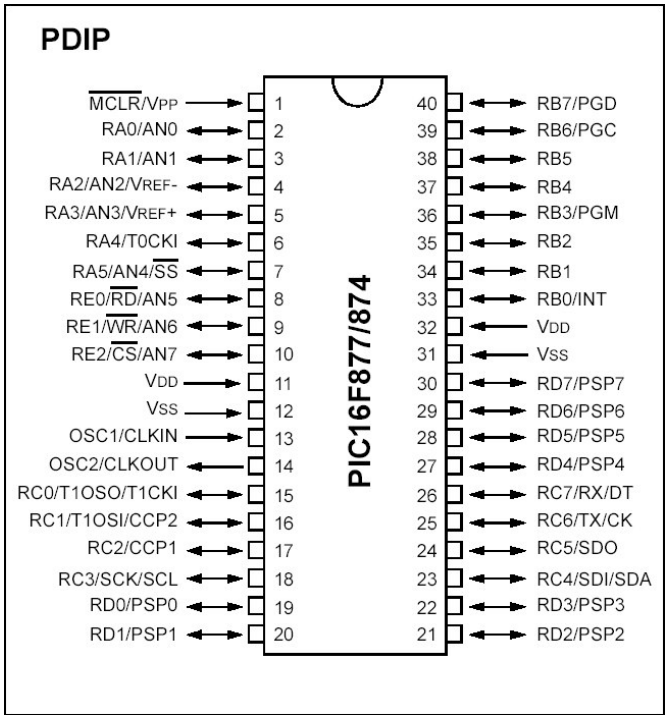
gg) Porta paralela escrava (PSP) de 8 bits, com controles de RD, WR e CS externos;

hh) Circuito de detecção *BROWN-OUT* para reset *BROWN-OUT* (BOR);

ii) 33 pinos de I/O.

A figura 1 mostra a pinagem do microcontrolador PIC16F877.

Figura 1: Pinagem microcontrolador PIC16F877.



2.2 Compilador PICBASIC PRO

O compilador PicBasic Pro foi desenvolvido pela *Micro Engineering Labs* e possui um conjunto de funções que tornam a programação dos microcontroladores PIC, notavelmente melhor estruturadas e definidas do que através do uso da linguagem *Assembly*, abstraindo detalhes internos de funcionamento destes microcontroladores, porém ainda com a possibilidade de programação em *Assembly*, caso seja necessário. Ele é totalmente compatível com o *Basic Stamp* II (MELABS, 2003).

Ele produz código que pode ser programado em uma vasta linha de microcontroladores PIC de 8 a 84 pinos, além de vários recursos como conversores A/D, temporizadores e portas seriais. A versão 2.4 do compilador PicBasic Pro, suporta todos os microcontroladores da família PIC, incluindo as séries de 12, 14 ,16 bits, PIC17CXXX e PIC18CXXX.

Para o desenvolvimento geral de programas utilizando o PicBasic Pro, os microcontroladores PIC16F628, 16F84, 16F876, 16F877, 18F252 e 18F452 são utilizados como referências pois utilizam memória de programa FLASH que possibilita uma rápida reprogramação de código. Outros microcontroladores, como o PIC12C5XX, 12C67X, 14C000, 16C4XX, 16C5X, 16C55X, 16C6XX, 16C7XX, 16C9XX, 17CXXX

e 18CXXX são desenvolvidos com tecnologia OTP (programável apenas uma vez) ou de janela que permite a remoção do programa anterior via luz ultravioleta após alguns minutos de exposição. A seguir são apresentadas algumas características do PicBasic Pro :

a) Propicia programas de rápida execução e maiores do que os gerados por outros interpretadores BASIC;

b) Acesso direto ou via funções de biblioteca a pinos e registradores;

c) Gerenciamento automático de páginas de memória de programa acima de 2K;

d) Vetores de *words*, *bytes* ou *bits*;

e) Estruturas de condição If...Then...Else...Endif;

f) Gerenciamento de expressões hierárquicas;

g) Tratamento de interrupções em BASIC ou *Assembly*;

h) Biblioteca *BASIC Stamp* I e II;

i) Suporte a *display* LCD;

j) Suporte a osciladores de 3.58MHz a 40MHz;

k) Instruções I2C para acesso a mais dispositivos externos incluindo EEPROMs seriais;

l) Possibilidade de programação em *Assembly* e suporte a chamadas de subrotinas com a instrução *Call*;

m) Compatibilidade com MPLAB/MPASM/ICE;

n) Pode ser executado em DOS ou *Windows*;

o) Suporta todos microcontroladores da família PIC;

p) Compatível com a maioria dos programadores para microcontroladores da família PIC.

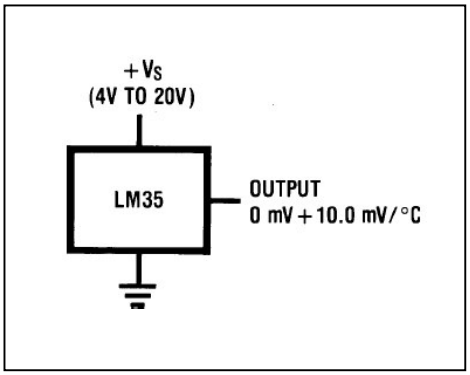
2.3 Componentes auxiliares

2.3.1 Sensor de Temperatura LM35

A série LM35, é formada por sensores de temperatura integrados com uma saída de tensão linearmente proporcional à temperatura em graus Celsius (°C). Sua saída linear fornece um fator de escala de +10mV/°C (NATIONAL, 2002).

Não requer qualquer tipo de calibração externa possuindo uma margem de erro de ± ¼ °C em temperatura ambiente e ± ¾ °C entre a faixa de leitura de -55°C a 150°C. Possui baixa impedância de saída (0,1Ω para 1mA de carga), saída linear, calibração precisa, tornando sua interface com circuitos externos, muito simples. Pode ser usado com fontes de alimentação simples, consumindo apenas 60μA de corrente com menos de 0,1°C de aquecimento interno. A tensão de alimentação pode ficar entre 4V e 30VDC. A figura 2 ilustra o sensor de temperatura LM35.

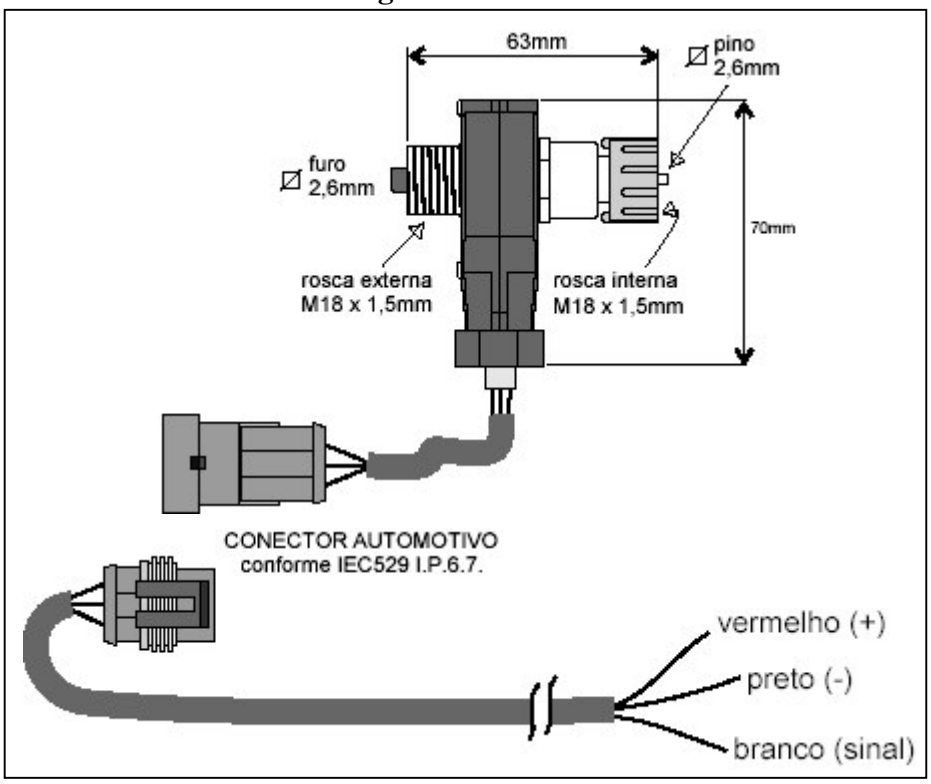
Figura 2: Representação do sensor LM35



2.3.2 Sensor de velocidade V305

Trata-se de um sensor efeito *Hall*. Este tipo de sensor tem seu princípio de funcionamento baseado na presença de um campo magnético, ou seja, quando existe a presença de um campo magnético o nível lógico do sensor será 0, e quando não houver a presença de um campo magnético será 1. Cada pulso gerado pelo sensor possui um período que pode ser transformado em valores de velocidade na ordem de Km/h. O sensor V305 gera 8 pulsos por volta completa (360°) e possui uma faixa de alimentação de 7V a 30VDC (FIP, 2003). A figura 3 ilustra o sensor de velocidade V305.

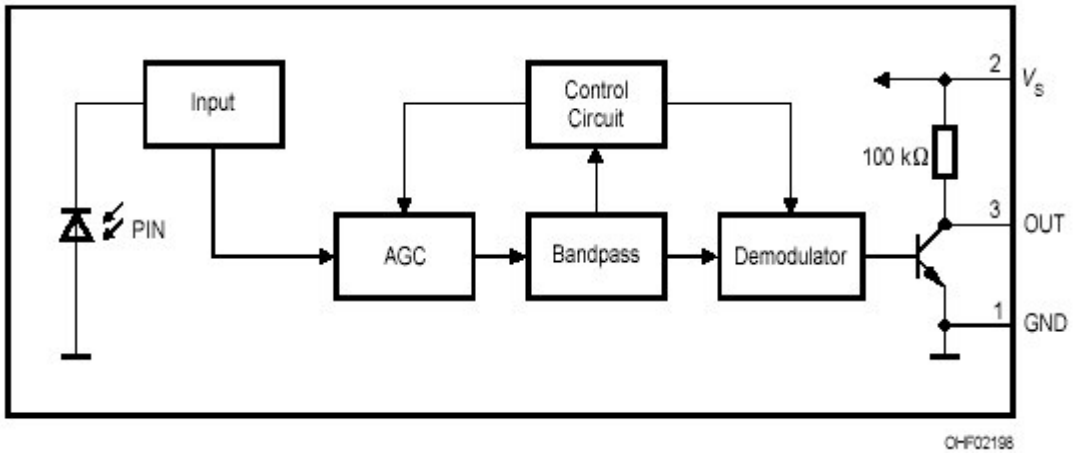
Figura 3: Sensor V305



2.3.3 Receptor infravermelho PHC38C

Trata-se de um foto-diodo com um circuito integrado híbrido, disponível para vários valores de freqüência entre 30 e 40KHz, neste caso 38KHz, que é encapsulado em um invólucro *Epoxy*. Possui alta imunidade a luz ambiente, baixo consumo, tensão de alimentação de 5VDC, alta sensibilidade e é compatível com a tecnologia CMOS e TTL. É largamente utilizado como módulo receptor de dados via infravermelho (SIEMENS, 2002).

Figura 4: Diagrama do receptor infravermelho PHC38C



2.3.4 Emissor infravermelho Systemlink 4

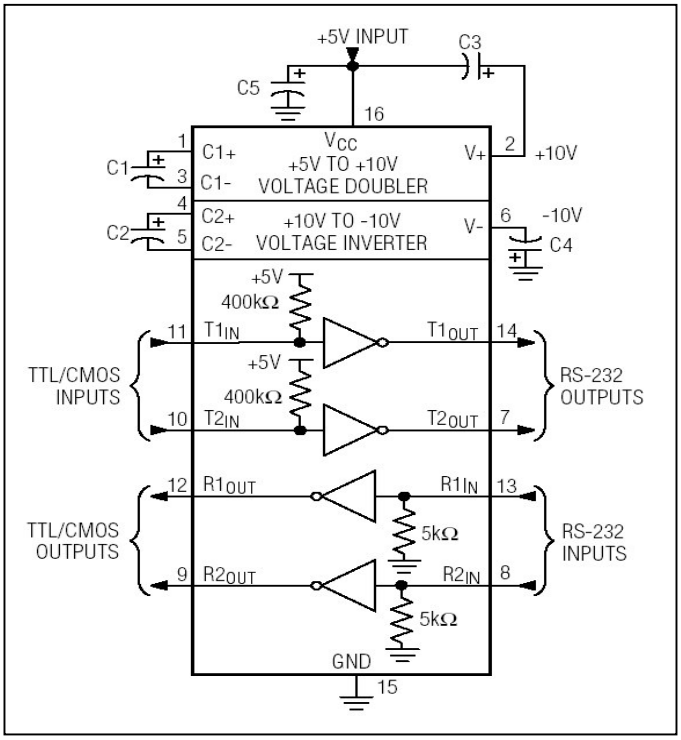
Trata-se de um controle remoto universal, fabricado pela RCA. Ele é destinado originalmente para comandos específicos para até dois aparelhos de TV e um aparelho VCR, ou um aparelho de TV e dois aparelhos VCR, 1 aparelho receptor de tv á cabo e 1 aparelho receptor de antena. Possui diversas configurações de normas de transmissão de pacotes via infravermelho, pré-armazenadas, para diversos fabricantes, incluindo a norma RC5 da Phillips, que será detalhada mais adiante (RCA, 1999).

2.3.5 Receptor/Emissor serial RS232 MAX232

A maioria dos equipamentos digitais utilizam níveis TTL ou CMOS. Portanto, o primeiro passo para conectar um equipamento digital a uma interface RS232 é transformar níveis TTL (0 a 5 Volts) em RS232 e vice-versa. Isto é feito por conversores de nível (MAXXIM, 2003).

Um circuito integrado mais popular para esta aplicação é o MAX232 (da Maxim). Ele inclui um circuito *charge pump* capaz de gerar tensões de +10V e –10V a partir de uma fonte de alimentação simples de +5V, bastando para isso alguns capacitores externos. Possui 2 receptores e 2 emissores no mesmo encapsulamento.

Figura 5: Conversor de nível MAX232



2.3.6 Display LCD WH2004-A

O baixo consumo, dimensões reduzidas e interfaceamento simples são algumas das características que fazem dos *displays* de cristal líquido (LCDs) os preferidos na maioria dos projetos em que uma informação deve ser apresentada na forma digital.

Partindo dos simples d*isplays* de 7 segmentos, que são a base de calculadoras e muitos instrumentos digitais, a eletrônica evoluiu rapidamente neste setor, criando novos dispositivos de maior complexidade, capazes de apresentar muitos dados de uma forma dinâmica a partir de informações obtidas de circuitos de grande complexidade, tais como microcontroladores, microprocessadores, etc...

Segundo Braga (1989), o *display* LCD já existe a bastante tempo, inclusive com versões extremamente gráficas, porém ainda hoje, os *displays* alfanuméricos LCD são largamente usados, devido principalmente ao custo e tamanho extremamente reduzido se comparado com os modelos mencionados. A tabela 1 mostra a especificação do *display* WH2004-A (WINSTAR, 2003).

Tabela 1: Especificação técnica *display* WH2004-A

Item	Dimensão	Unidade
Nr. de caracteres	20 caracteres x 4 linhas	□
Dimensões do módulo	98.0 x 60.0 x 13.6(MAX)	mm
Área de visão	77.0 x 25.2	mm
Área ativa	70.4 x 20.8	mm
Tamanho do pixel	0.55 x 0.55	mm
Tamanho do caracter	2.95 x 4.75	mm
Tipo de LCD	STN, Positivo, Transflectivo, Cinza	
Backlight	Não possui	

Existem duas áreas de memórias distintas muito utilizadas nestes modelos de *display* LCD que são chamadas de DDRAM (*display data* RAM) e *CGRAM (character generator RAM*). A DDRAM é a área onde os caracteres são endereçados e são visíveis no *display*, onde cada endereço de 8 bits corresponde a um caracter sempre apresentado no *display*. Os endereços de disposição são 80H para a 1ª linha, C0H para a 2ª linha, 94H para a 3ª linha e D4H para a 4ª linha.

Já a área de CGRAM é a área onde é possível personalizar até 8 caracteres de 5x8 *pixels* ou 4 caracteres de 5x10 *pixels* para o programa. Para isso deve-se escrever

nos endereços abaixo valores que indiquem quais pixels devem ficar acessos, formando o caracter desejado. Neste modelo de *display* estendem-se desde o endereço 40H até 7FH, sendo que cada caracter personalizado possui 8 bytes desta área caso ele seja de 5x8 *pixels*. A ligação elétrica deste *display* baseia-se na tabela 2.

Tabela 2: Pinagem *display* WH2004-A

Nr. Pino	Símbolo	Nível	Descrição
1	V_{SS}	0V	Terra (GND)
2	V_{DD}	5.0V	Positivo (5Vcc)
3	VO	(Variável)	Contraste LCD
4	RS	1/0	1: dados, 0: instruções
5	R/W	1/0	1: Leitura(MPU→Modulo) 0: Escrita(MPU→Modulo)
6	E	1,1→0	Sinal Chip Enable
7	DB0	1/0	Bit0 de dados
8	DB1	1,0	Bit1 de dados
9	DB2	1,0	Bit2 de dados
10	DB3	1,0	Bit3 de dados
11	DB4	1,0	Bit4 de dados
12	DB5	1,0	Bit5 de dados
13	DB6	1,0	Bit6 de dados
14	DB7	1,0	Bit7 de dados

2.3.7 Inversor de tensão DC/AC

O inversor de tensão DC/AC é um equipamento especial, que tem como objetivo converter uma tensão de entrada DC para AC. Ele tem uma vasta gama de aplicações, e as mais usuais são em *No Breaks* e sistemas de energia solar.

Existem várias especificações de inversores de tensão, com diferentes tensões de entrada e saída e diferentes potências, que produzem diferentes formas de onda em suas saídas, como por exemplo a onda quadrada e a onda senoidal pura. A tabela 3 apresenta um comparativo entre estes dois modelos de inversor DC/AC.

Tabela 3: Comparativo inversores onda quadrada X senoidal

	Onda Quadrada	Onda Senoidal
Custo	Baixo	Alto
Dimensão	elevada	reduzida
Peso (150VA)	±1Kg	±0,4Kg
Aquecimento	Excessivo	médio
Ruído	alto	baixo

Devido a baixa qualidade da saída gerada pelo inversor de onda quadrada, podem ocorrer problemas relacionados a ruídos, acarretando no mau funcionamento do sistema, assim como chiados, não previstos no sistema sonoro. Para amenizar estes efeitos, podem ser utilizados filtros, com o objetivo de amenizar o nível de ruído.

Na prática, os inversores de tensão não se apresentam tão eficientes eletricamente, produzindo muitas perdas, mas no caso de sistemas de baixo consumo, isto não tende a ser um fator tão agravante. Porém, como solução mais eficiente para este problema, existem as fontes DC/DC, que convertem uma tensão DC diretamente para uma ou mais tensões DC, porém estas fontes são extremamente caras e difíceis de projetar, pois os principais componentes utilizados, são de difícil aquisição.

Figura 6: Inversor de tensão

2.3.8 Bateria automotiva

A bateria automotiva, é responsável pela distribuição de energia elétrica para todo o veículo, sendo necessária para diversas funcionalidades de um automóvel, tais como, iluminação/alertas, ignição, som automotivo, sistemas de injeção eletrônica, alarmes, trio elétrico, etc... Esta bateria possui um circuito de recarga evitando que a mesma seja descarregada impossibilitando até mesmo a ignição do veículo.

Geralmente a bateria automotiva vem especificada em A/h ou MRC. *A/h* significa a máxima corrente que a bateria pode fornecer durante uma hora até se esgotar e *MCR* (minutos de capacidade de reserva) significa o tempo que ela consegue fornecer o máximo de energia.

Carros populares geralmente possuem bateria de 36 A/h e carros de médio porte, 55A/h e 60MCR. Outro dado é a *corrente de partida* (CCA = *cold cranking amps*) que indica a corrente máxima de pico (em frações de segundo) (MOTORCITY, 2003).

A *corrente máxima da bateria* é obtida através da aproximação com o valor MCR, portanto, uma bateria de 60 MCR pode fornecer no máximo 60A. A vida útil da bateria é especificada em ciclos de carga e descarga com valores médios de 2000 ciclos ou 2 a 3 anos de vida útil. A bateria é composta por seis células de chumbo/ácido sulfúrico (duas placas de chumbo mergulhadas em ácido) que produzem uma tensão de 2,4V cada, totalizando 14,4V na bateria com carga máxima, portanto, se a tensão cai abaixo de 10,5 V é porque existe alguma célula em curto.

Atualmente estão sendo fabricadas baterias com uma liga de prata/chumbo nos eletrodos positivos. Já outras baterias possuem *gel* ao invés do ácido líquido e utilizam placas enroladas como caracóis ao invés de simples placas colocadas paralelamente como nas baterias comuns, com o objetivo de aumentar a área de cada eletrodo aumentando assim a máxima corrente que ela pode fornecer.

Por maior que seja a capacidade de fornecimento de corrente de um bateria, ela sempre pode estar sujeita a ser descarregada, prejudicando de forma direta o funcionamento do veículo, pois a bateria é um componente vital para um sistema automotivo. Esta descarga pode ocorrer por várias formas, dentre elas: a bateria não aceita mais carga e precisa ser trocada, as luzes do veículo ficaram acesas por muito tempo, falha de isolamento nos dispositivos elétricos do veículo. Mas na prática uma das formas mais comuns de uma bateria ficar descarregada está diretamente ligada ao uso do som automotivo. Além da potência destes sistemas sonoros o tempo de consumo dos mesmos, acabam acarretando no descarregamento parcial e até total de uma bateria.

A referência inicial de tensão para uma bateria considerada descarregada é 10,6V. A referência inicial de tensão para uma bateria com carga completa é 14,4V.

Baseado nestas referências, ocorre o monitoramento destes valores de tensão e a identificação do status de carga da bateria, evitando que o veículo fique sem ignição e conseqüentemente possibilitando a recarga da bateria automotiva.

2.3.9 Acoplador 4N25

Trata-se de um acoplador ótico de uso geral encapsulado em formato DIP 6 pinos, que possui internamente um diodo emissor de luz infravermelha que satura um foto-transistor, isolando a entrada e a saída (NATIONAL, 2002).

Dentre suas aplicações principais destacam-se :

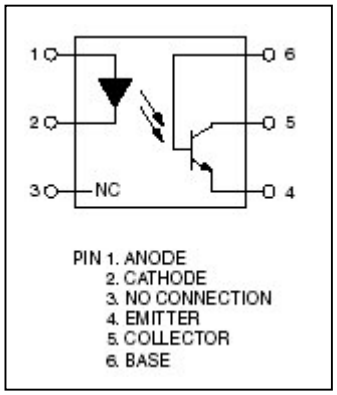
a) Fontes de alimentação reguladas;

b) Entradas lógicas digitais;

c) Entradas microprocessadas.

A figura 7 apresenta o diagrama do acoplador 4N25.

Figura 7: Diagrama do acoplador 4N25



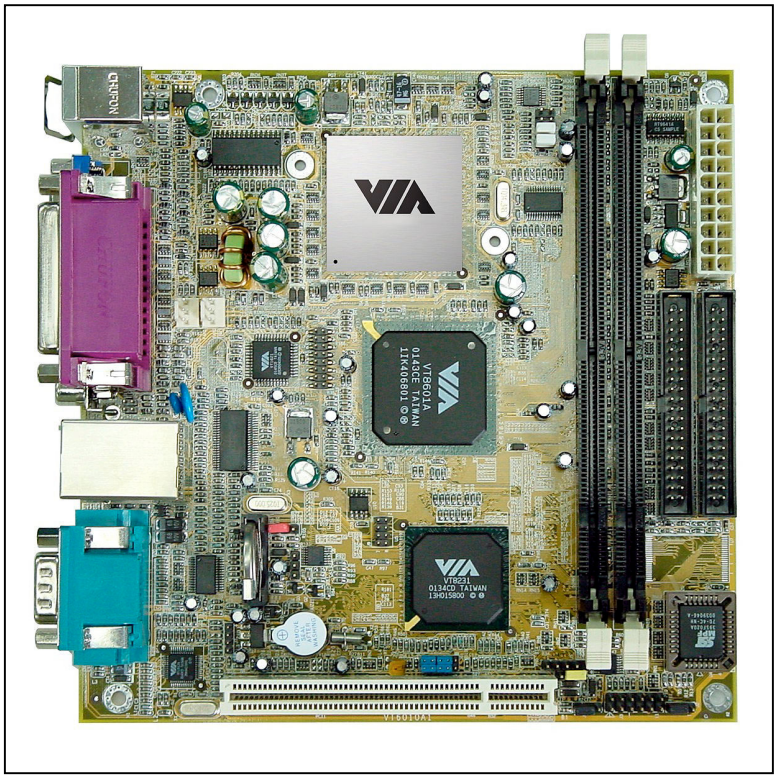
3 Hardware PC

3.1 Placa-Mãe Mini-ITX EPIA

Trata-se de uma placa-mãe para plataforma embarcada x86 ultra compacta, que possui uma alta integração de periféricos. Foi desenvolvida pela VIA Technologies, famoso fabricante de *chipsets* e fabricante também dos processadores Cyrix. Atualmente já existem três fabricantes deste padrão de placas-mãe : VIA, Lucky-Star e Jet-Way (MINI-ITX, 2003).

Este padrão de placas-mãe, denominado Mini-ITX abre as portas para uma nova geração de sistemas embarcados inovadores, ergonômicos e pequenos. Através do seu grande nível de integração de periféricos, ela ocupa 66% do espaço de uma placa-mãe padrão FlexATX e vem com um processador VIA soldado diretamente na placa, inclusive com opção de operação sem cooler, tornando sua operação muito silenciosa. A figura 8 apresenta a placa-mãe Mini-ITX EPIA (VIA, 2002).

Figura 8: Placa-Mãe Mini-ITX EPIA



3.1.1 Dados Técnicos

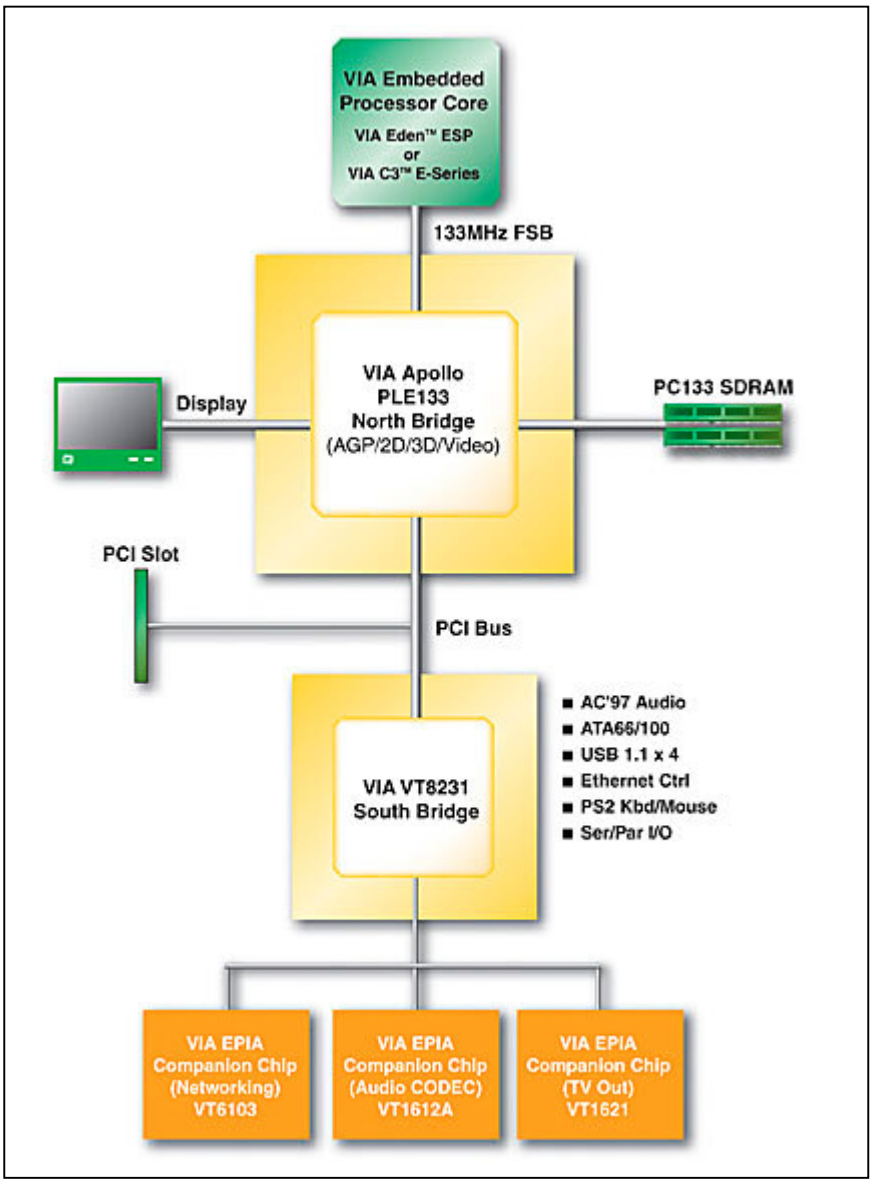
Na tabela 4 é apresentada a especificação desta placa-mãe.

Tabela 4: Especificação placa-mãe Mini-ITX EPIA

Formato	- 170mmx170mm - Mini-ITX
Processador	VIA Eden formato ESP - Freqüência do Barramento – 100/133MHz - Baixo consumo - Operação sem cooler - Freqüência do Processador – 533MHz VIA C3 formato EBGA - Freqüência do Barramento – 100/133MHz - Memória cache L1 128K, L2 64K - Freqüência do Processador - > 800MHz - Operação com cooler
Chipset	VIA Apollo PLE 133 - Ponte Norte VT8601A - Ponte Sul VT8231 - Possui interface AGP 2X integrada
Saída para TV	- Alta qualidade de escala e filtro - Saída de vídeo Composto ou S-Video - Suporta os formatos NTSC/PAL
Memória principal	- Dois soquetes para módulo de memória DIMM 168 pinos - Suporta SDRAM PC100/133
Rede	Rede Ethernet VIA 10/100 integrada
Vídeo	- Vídeo AGP2X integrado com aceleração gráfica 2D/3D - Compensação de movimento para reprodução de DVD - Porta de entrada de vídeo
Áudio	VIA VT1612A AC'97 integrada - Jaque de saída de áudio (Line-Out) - Jaque de entrada de áudio (Line-In) - Jaque de entrada de microfone (Microphone-In) - Compatível com Sound Blaster e Sound Blaster Pro - I/O digital compatível com S/PDIF
Slots de expansão	1 slot PCI
IDEs integradas	2 interfaces IDE ATA/100/66
Portas I/O	- 3 jaques de áudio (Line-Out, Line-In, Microphone-In) - 2 portas USB - 1 porta paralela EPP/ECP - 1 porta serial 16C550 - 2 conectores externos Teclado/Mouse PS/2 - 2 portas de saída de TV (S-Video e RCA) - 1 saída S/PDIF - 1 porta RJ45 para rede
Consumo elétrico	< 40W
Alimentação	Padrão ATX

Na figura 9 é apresentado o diagrama de *hardware* desta placa-mãe (VIA, 2002).

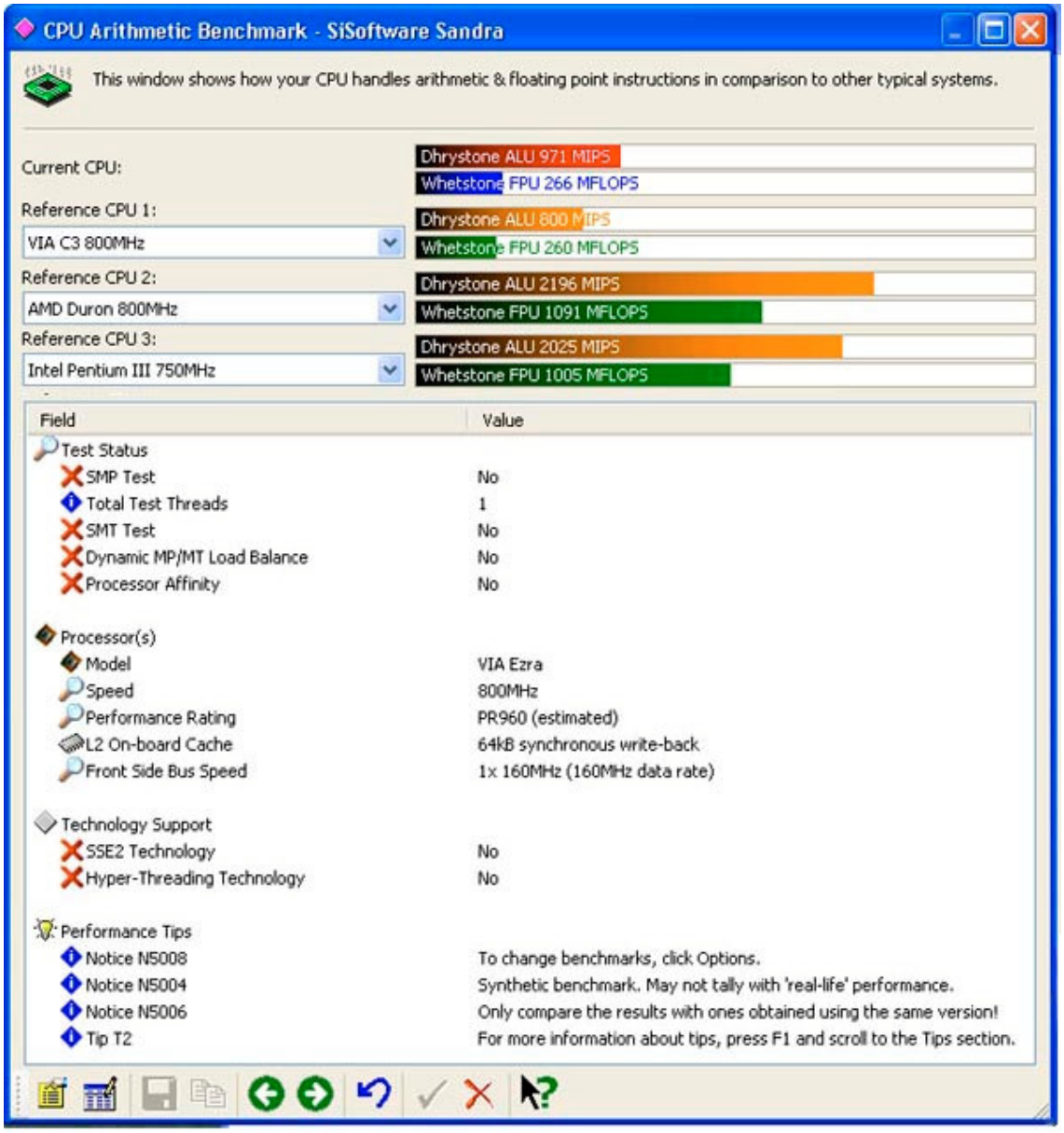
Figura 9: Diagrama de hardware da placa-mãe Mini-ITX



3.1.2 Testes de Desempenho de Hardware

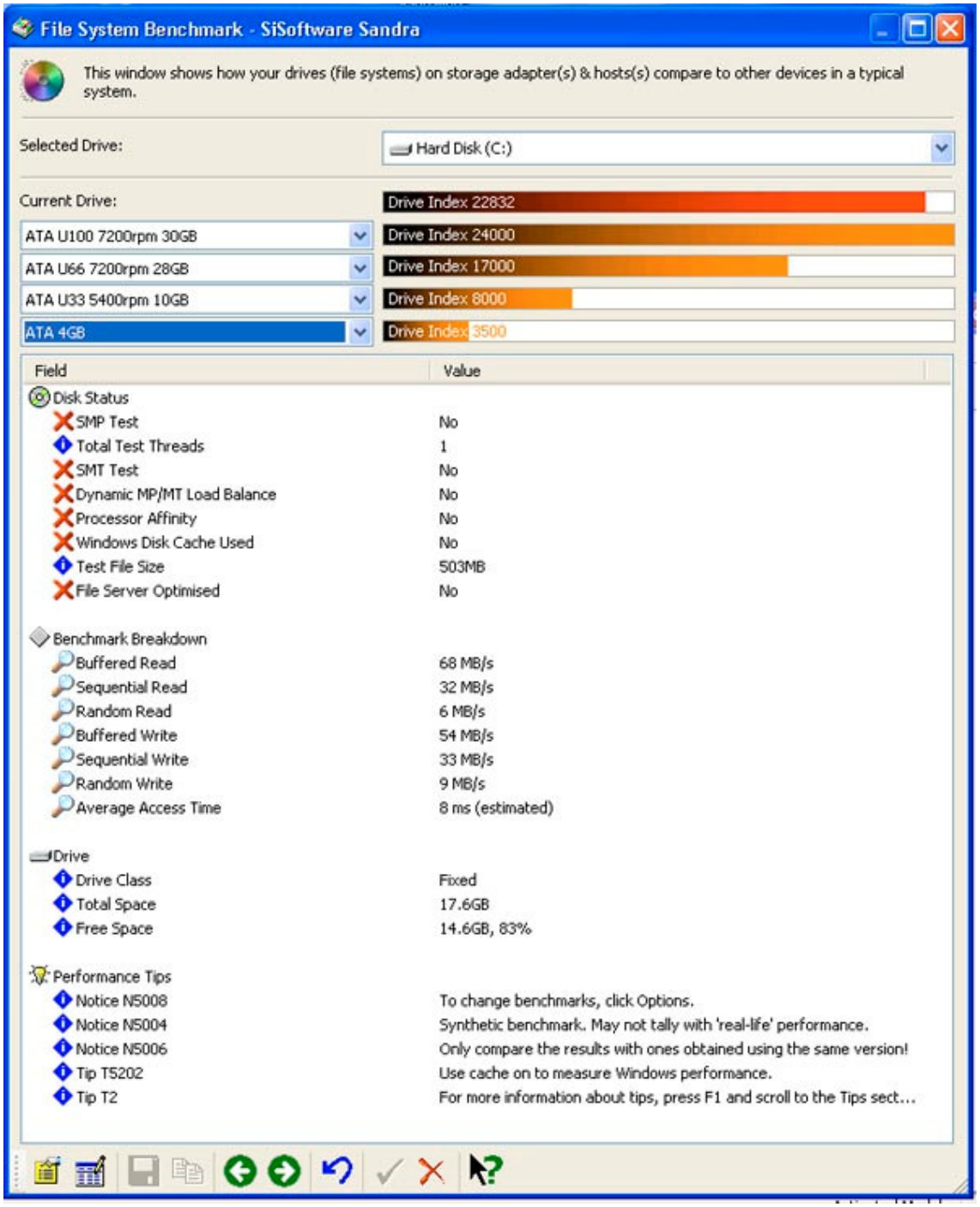
Foram realizados alguns testes de desempenho de *hardware* utilizando o *software* Sandra 2002. A placa-mãe Mini-ITX EPIA800 foi testada com a seguinte configuração : 256Mb de memória RAM PC133, HD 40Gb 7200rpm Western Digital, leitor de DVD 16X LG, fonte chaveada ATX 350W 110/220V, teclado e mouse. Como sistema operacional, foi escolhido o Windows Xp, cuja instalação apresentou-se extremamente rápida e eficiente, não apresentado qualquer tipo de conflito de *hardware* ou *software* (HITECHMODS, 2002). A figura 10 ilustra os resultados do teste de CPU.

Figura 10: Teste de Hardware CPU – Mini-ITX



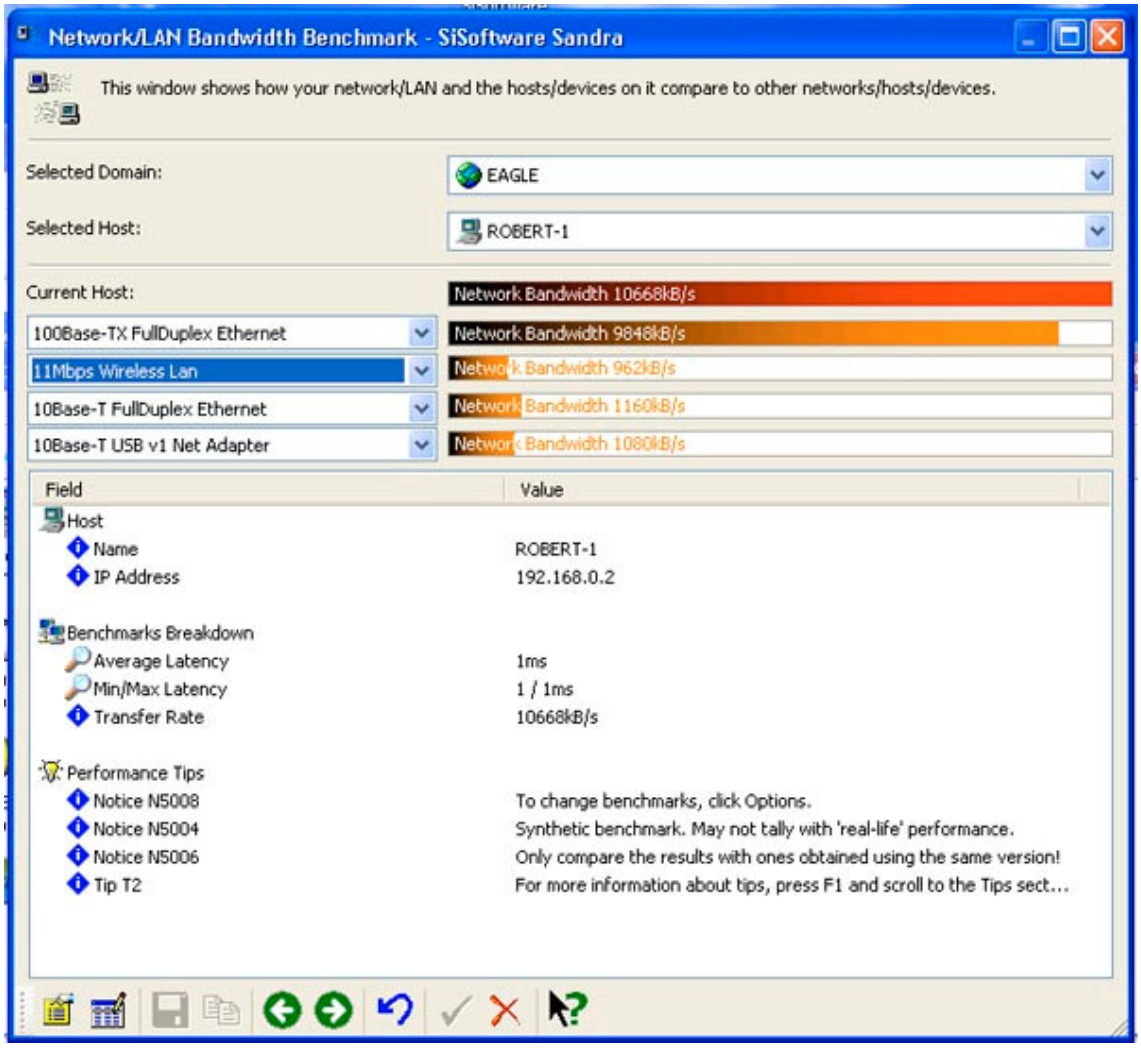
A figura 11 ilustra o resultado do teste de hardware do Hard Disk.

Figura 11: Teste de Hardware HD – Mini-ITX



A figura 12 ilustra o resultado do teste de hardware da rede *onboard* desta placa-mãe.

Figura 12: Teste de Hardware Rede – Mini-ITX



A tabela 5 apresenta as temperaturas lidas durante os testes.

Tabela 5: Temperaturas obtidas durante os testes – Mini-ITX

Sem carga	**Com carga**
Processador = 86ºF	Processador = 95ºF
Placa-mãe = 93ºF	Placa-mãe = 97ºF

Segundo Hitechmods (2002), o processador C3 não tem o mesmo poder de processamento dos fabricantes AMD ou Intel, mas atende perfeitamente a diversas soluções. O teste de desempenho do HD, mostrou que não existem perdas na taxa de transferência de dados.

Já o teste de rede indicou que a velocidade de transmissão de dados aproxima-se de 100Mbps. Estes três resultados confirmam que esta placa-mãe poderia ser utilizada com um servidor de arquivos e impressão (HITECHMODS, 2002).

Além destes testes foram feitos testes de *software*. Foram instalados *softwares* como Microsoft Office 2000, Adobe Photoshop, Word Perfect, Internet Explorer 6, Cute FTP, Outlook, WinZip, DIVX e todos apresentaram um bom funcionamento.

Através de todos os testes realizados chegou-se a conclusão de que esta placa-mãe não possui os recursos necessários para jogos ou estações gráficas pesadas, porém em muitas outras aplicações ele se encaixa perfeitamente como por exemplo : servidores de impressão, *players* MP3, estações de trabalho padrão (Internet, Processador de texto, sem jogos 3D...), servidores *proxy*, servidores WEB e aplicações restritamente embarcadas (HITECHMODS, 2002).

4 Sistema operacional Linux

O sistema operacional Linux é a versão mais popular do sistema operacional UNIX. Sua grande popularidade deu-se ao fato desse sistema operacional ser *freeware* (TORRES, 1999). Existem diversas versões de Linux disponíveis tais como, Red Hat, Slackware, Debian, Mandrake, Suse.

Segundo Anunciação (1999, p.34), a arquitetura do Linux é bastante estável e segura. Trabalha em modo protegido e oferece proteção de memória para os seus aplicativos bem como multitarefa preemptiva, além de ser um sistema operacional multi-usuário. Ele foi desenvolvido para empresas, universidades, computadores pessoais (PCs) e vem sendo utilizado em larga escala em aplicações embarcadas também.

O que torna o Linux diferente de outros sistemas operacionais é a sua implementação aberta. Ele foi e permanece sendo desenvolvido por um grupo de voluntários, inicialmente na *internet*, compartilhando código, identificando e resolvendo *bugs*. Inicialmente o sistema operacional Linux foi desenvolvido por Linus Torvalds, inspirando-se no mini sistema operacional UNIX, chamado MINIX e desenvolvido por Andy Tanenbaum (WELSH, 1995).

Algumas das principais características do sistema operacional Linux são:

a) Sistema multitarefa, multi-usuário;

b) Compatível com o maior número de sistemas UNIX;

c) Tem suporte a console virtual;

d) Emula instruções 387-FPU para operações matemáticas com ponto flutuante;

e) Suporta diversos sistemas de arquivo (ext2fs, MS-DOS, ISO9660 CDROM);

f) Possui recursos para comunicação via rede protocolos TCP-IP, SLIP, PLIP;

g) Núcleo desenvolvido em modo protegido para processadores Intel 80386 e 80486;

h) Suporta memória virtual (*swap*) alocada em disco rígido;

i) Os programas utilizam bibliotecas compartilhadas e ligadas dinamicamente, customizando o uso de memória RAM.

4.1 Comunicação serial

Assim como em outros sistemas operacionais como o Microsoft Windows, no Linux também existe uma seqüência de funções necessárias para uma comunicação serial. São elas :

1) Abrir a porta serial – através da chamada *open* a porta serial é aberta. Exemplo :

   fd = open(devicename, O_RDWR | O_NOCTTY | O_NONBLOCK);

   onde :

   "devicename" – caminho do device *driver* da porta serial

   O_RDWR – abrir a porta para leitura e escrita

   O_NOCTTY – não aloca o dispositivo como terminal de controle para este processo caso seja um dispositivo de terminal.

   O_NONBLOCK – Seta o modo não-bloqueado caso o dispositivo seja FIFO.

   Como retorno "fd" terá o descritor de arquivos utilizado para a porta serial.

   Se houver erro a chamada *open* retornará "-1".

2) Configurar a Porta Serial – Originalmente a porta serial possui uma configuração, porém nem sempre esta configuração é equivalente a necessária.

   newtio.c_cflag = BAUD | CRTSCTS | DATABITS | STOPBITS | PARITYON | PARITY | CLOCAL | CREAD;

   onde :

   BAUD - Define a taxa de baudrate para a porta serial (geralmente 9600 bauds)

   DATABITS – Define o nr de *bits* do dado (geralmente 8 bits)

   STOPBITS – Define o nr. de *stop bits* (geralmente 1)

   PARITY – Define a paridade (par ou ímpar)

   "Newtio" é um *struct termios* apontado para c_cflag onde serão armazenadas as novas configurações da porta serial. É interessante a criação de uma cópia das configurações atuais, antes de reconfiguração da porta serial. Exemplo :

   tcsetattr(fd,TCSANOW,&newtio);

Desta forma o descritor “fd” recebe a nova configuração para a porta serial.

3) Iniciar a leitura de dados – a leitura de dados é realizado com a chamada *read*

   res = read(fd,buf,nbytes);

   onde :

   fd – descritor utilizado para a porta serial

   buf – buffer de recepção dos dados via serial

   nbytes – quantidade de *bytes* a receber

   A variável “res” terá a quantidade de *bytes* recebido. Caso ocorra um erro “res” terá “-1”.

4) Escrita de dados – a escrita de dados é realizada com a chamada *write*.

   write(fd,buf,nbytes);

   onde :

   fd – descritor de arquivo para a porta serial

   buf – *buffer* a ser transmitido via serial

   nbytes – quantidade de *bytes* a transmitir.

   Caso a operação tenha sucesso irá retornar o número de bytes transmitidos, caso contrário irá retornar “-1”.

5) Finalizar comunicação – geralmente se restabelece a cópia original de configurações da porta serial através da linha de código :

   tcsetattr(fd,TCSANOW,&oldtio);

   onde : fd –descritor de arquivos da porta serial

   oldtio – *struct termios* contendo a cópia das configurações originais da porta serial.

6) Fechar porta – a chamada *close* fecha a porta serial. Exemplo :

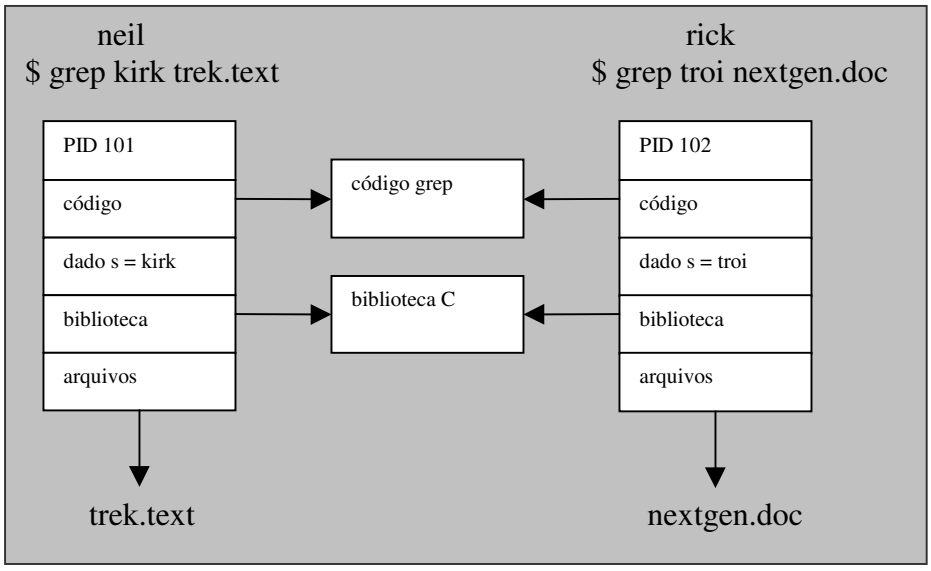
   close(fd);

4.2 Processos UNIX

Processos são fundamentais em sistemas operacionais UNIX, controlando quase todas as atividades de um computador UNIX. É possível criar, iniciar e parar processos dentro de outros programas, assim como enviar e receber mensagens. O processo consiste numa área de memória alocada e uma *thread* simples de controle que executa neste espaço seus recursos de sistema requeridos (VOLKERDING, 1998).

A figura 13 ilustra a estrutura de um processo. Cada processo possui um identificador PID, uma área de código e bibliotecas que podem ser compartilhadas aumentando a eficiência de utilização da memória física, uma área de variáveis que é exclusiva de cada processo e um descritor de arquivos. Possuem também uma pilha para variáveis locais e para chamadas *call* e *return*. Estes processos são controlados por uma tabela de processos indexadas pelo PID de cada processo. Cada processo filho é iniciado por outro conhecido como processo pai. O processo inicial do UNIX é o INIT que irá criar vários processos que irão criar outros vários processos e assim sucessivamente.

Figura 13: Estrutura de um processo UNIX



Segundo Matthew (1996, p. 21), seguindo a característica de que o sistema operacional UNIX é multitarefa preemptivo, os processos também possuem prioridade. O escalonador de processos de UNIX se baseia nestas prioridades para executar os processos, dando maiores fatias de tempo de execução para processos de maior prioridade. A execução de um processo pode ser realizada com o comando *system* (sem

interação) ou com os comandos da família *exec* (controle fino sobre as ações do processo).

4.2.1 Substituindo a imagem de um processo

A família *exec* de chamadas, troca o processo atual, por outro criado de acordo com os argumentos passados. O quadro 1 ilustra as suas variações disponíveis.

Quadro 1: Variações das chamadas execs

```
# include <unistd.h>
char **environ;
int execl(const char *path, const char *arg0, ...,   (char *)0);
int execlp(const char *path, const char *arg0, ...,  (char *)0);
int execle(const char *path, const char *arg0, ...,  (char *)0, const
char *envp[]);
int execv(const char *path, const char *argv[]);
int execvp(const char *path, const char *argv[]);
int execve(const char *path, const char *argv[], const char *envp[]);
```

O programa atribuído ao argumento *path* é usado com o código de programa, para executar no local no qual está atualmente executando. O novo programa passa argumentos “arg0”, “arg1”, ... até um argumento nulo. Alternativamente, os novos argumentos de programa podem ser passados na forma de um apontador para um vetor de *strings* “argv”. Em ambos casos, o novo programa inicia com os argumentos dados em *argv* passados para *main*.

As funções com nomes de sufixo “p” diferem daquelas que irão procurar a variável *path* para localizar o novo programa executável. Se o executável não está em *path*, um nome de arquivo absoluto, incluindo diretórios, precisará ser passado para a função, com um parâmetro. A variável global *envirom* é usada para passar o valor para o ambiente do novo programa. Alternativamente, um argumento adicional para as chamadas *execl* e *execve* está disponível para a passagem de um vetor de *strings* para ser usado como o ambiente do novo programa.

4.2.2 Duplicando uma imagem de processo

Segundo Matthew (1996), o uso de processos para mais de uma função, deve criar um processo inteiramente separado do programa principal, como faz o INIT, diferentemente de substituir a *thread* atual de execução, como o caso de chamadas *exec*.

Segundo Mitchell (2001), a chamada *fork* cria um novo processo. Esta chamada de sistema cria uma imagem do processo atual, criando uma nova entrada na tabela de processos com muitos dos mesmos atributos do processo atual. O novo processo é quase idêntico ao original, executando o mesmo código, mas com sua própria área de dados, ambientes e descritores de arquivo. A chamada *fork* cria um novo processo filho, idêntico ao processo pai que o chamou, salvo que o novo processo possui um PID próprio. O novo processo herda uma cópia da área de memória de dados do processo pai (variáveis), descritores de arquivo abertos e diretórios.

A chamada *fork* retorna o PID do novo processo filho criado. Este continua a executar como o original, com a exceção que no processo filho a chamada *fork* retorna "0". Isso permite pai e filho determinarem quem é quem. Se *fork* retornar "-1" é por que houve falha ao criar o processo filho (MATTHEW, 1996).

4.2.3 Espera por um Processo

A chamada *wait* pausa o processo pai até que um de seus processos filho termine ou pare. Ela retorna o PID do processo filho do qual a informação de *status* é disponível. Esta técnica pode ser utilizada para evitar a criação de processos que tornem-se obsoletos (*zombies* ou *defunct*).

4.2.4 Comunicação entre processos

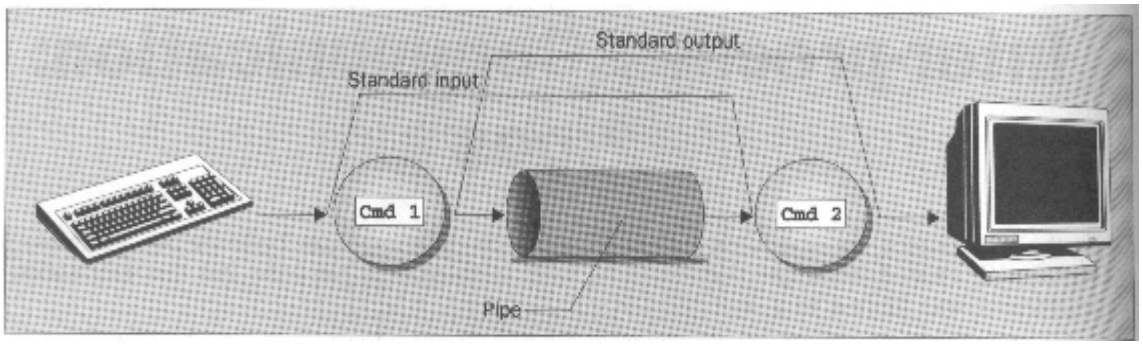
PIPE é o termo utilizado quando existe um fluxo de dados entre dois processos. Geralmente é ligada a saída de um processo a entrada do outro. Tal operação constitui a comunicação entre processos.

Por exemplo :

Cmd1 | Cmd2

O caracter | representa um pipe no sistema operacional UNIX. Desta forma o interpretador *shell* pode direcionar a entrada e a saída padrão de forma que os dados do teclado sejam passados ao vídeo através de dois comandos diferentes. A figura 14 ilustra a representação simbólica desta comunicação entre processos.

Figura 14: Representação de um pipe



4.2.5 Processos utilizando PIPES

A maneira mais simples de passagem de dados entre dois programas é com a utilização das chamadas *popen* e *pclose* (MATTHEW, 1996).

4.2.5.1 Popen

O quadro 2 apresenta a sintaxe da chamada *popen*.

Quadro 2: Sintaxe da chamada *popen*

```
#include <stdio.h>
FILE *popen(const char *command, const char *open_mode);
Int pclose(FILE *stream_to_close);
```

Permite invocar outro programa como se fosse um novo processo e ambos trocam dados. A *string* "command" é o nome do programa a ser executado com parâmetros adicionais se necessário. O parâmetro "open_mode" precisa ser "r" ou "w", porém, não é possível invocar outro programa e ambos lerem e escreverem para ele.

4.2.5.2 Pclose

Finaliza processos criados por *popen*. A chamada *popen* permite a execução de comandos complexos *shell*, porém o mesmo geralmente tende a necessitar de muitos recursos de sistema. Trata-se de uma função de alto nível.

4.2.6 A chamada PIPE

O quadro 3 apresenta a sintaxe da chamada *pipe*.

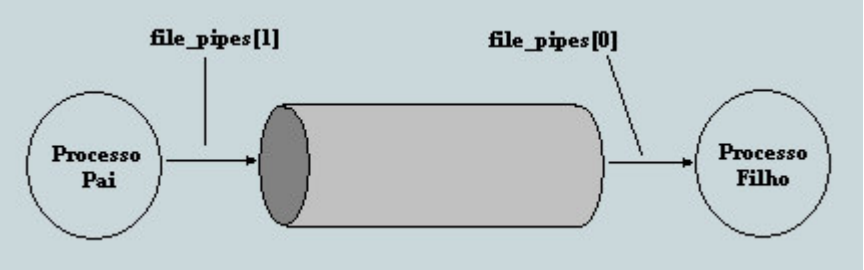
Quadro 3: Sintaxe da chamada *pipe*

```
#include <unistd.h>
int pipe(int file_descriptor[2]);
```

Não possui o *overhead* de utilização do interpretador de comandos *shell*, para o comando requisitado e fornece maior controle sobre leitura e escrita dos dados. O *pipe* aponta para um vetor de 2 inteiros. Ele preenche o vetor com 2 novos descritores de arquivo e retorna 0. Estes novos descritores são conectados em um caminho especial.

Qualquer dado escrito no descritor de arquivo[1], pode ser lido no descritor de arquivo[0], seguindo a lógica FIFO, *first in*, *first out*. Com a criação de um *pipe* no processo original e utilizando a chamada *fork* para criar um novo processo, torna-se possível passar dados de um processo para outro abaixo do *pipe*. A figura 15 ilustra esta representação (MATTHEW, 1996).

Figura 15: Comunicação de dois processos via *pipe*



4.2.7 Processos Pai e Filho

Com a utilização de uma chamada *exec* dentro de um *pipe,* um processo filho executa um programa diferente do processo pai. Porém deve-se saber qual descritor de arquivos esta chamada *exec* deve acessar, através da passagem do descritor de arquivos como parâmetro para o programa executado por *exec*.

4.2.8 Pipes usados como entrada e saída padrão

Com a chamada *dup* é possível invocar programas padrões, que não esperam um descritor de arquivo como parâmetro. O quadro 4 mostra a sintaxe da chamada *dup*.

Quadro 4: Sintaxe da chamada *dup*

```
#include <unistd.h>
int dup(int file_descriptor);
int dup2(int file_descriptor_one, int file_descriptor_two);
```

A função da chamada *dup* é abrir um novo descritor de arquivos, semelhante á chamada *open*. O novo descritor criado por *dup* refere-se ao mesmo *pipe* com um descritor de arquivo existente. No caso de *dup* ou *dup2*, o novo descritor de arquivo é sempre o menor número disponível (MATTHEW, 1996).

A entrada padrão de um descritor de arquivo é sempre 0. Ao fechar o descritor de arquivos[0] e chamar *dup*, o novo descritor de arquivos irá ter número 0. Como o novo descritor de arquivo é uma cópia do existente, a entrada padrão irá ser alterada para acessar o arquivo ou *pipe* do qual o descritor de arquivo foi passado por *dup*.

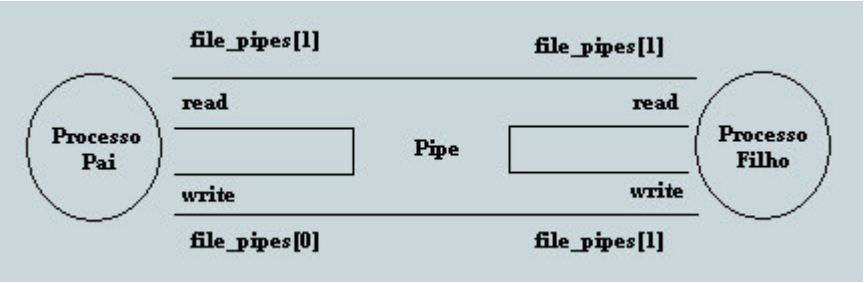
Segundo Matthew (1996), serão criados dois descritores de arquivo, que irão referenciar o mesmo pipe e um deles será a entrada padrão. A tabela 6 ilustra o status dos descritores de arquivos.

Tabela 6: Descritores de arquivo manipulados por *close* e *pipe*

Nr. descritor	Inicial	Após "close"	Após "dup"
0	Entrada padrão		Descritor pipe
1	Saída padrão	Saída padrão	Saída padrão
2	Erro padrão	Erro padrão	Erro padrão
3	Descritor pipe	Descritor pipe	Descritor pipe

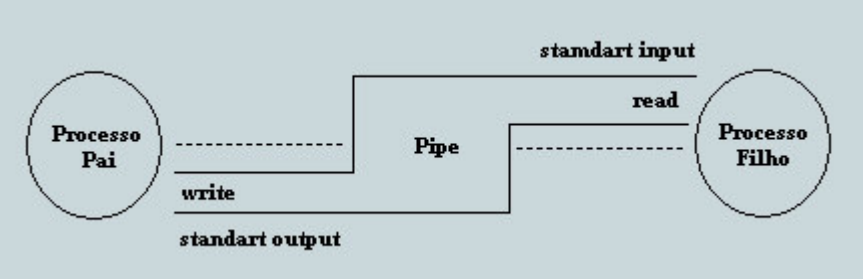
A figura 16 ilustra dois processos após uma chamada *fork*.

Figura 16: Processos após uma chamada *fork*.



A figura 17 ilustra os mesmos dois processos após a utilização da chamada *dup*, interligando a saída padrão do processo pai com a entrada padrão do processo filho.

Figura 17: Processos após a chamada *dup*



4.3 Compilador gcc

Trata-se do compilador C mais usado dos sistemas operacionais UNIX. Existem diversos compiladores C disponíveis para Linux, porém o compilador "gcc" apresenta-se como o mais completo (THE GCC TEAM, 2003). Ele foi criado e é atualmente mantido pela comunidade GNU. A sintaxe de utilização do gcc é a seguinte :

$ gcc teste.c

O resultado é a criação de um arquivo executável com o nome padrão a.out, como resultado do código compilado do arquivo "teste.c".

Existe a possibilidade de criação de um nome específico para o programa, ao invés de "a.out", procedendo com a seguinte sintaxe :

$ gcc teste.c -o tcc

Para compilar programas que estão em mais de um arquivo, cria-se inicialmente os objetos, para compilar o programa principal. Exemplo :

Arquivos : serial.c pipes.c tcc.c

Compilação dos três arquivos com :

$ gcc -c serial.c -o serial.o

$ gcc -c pipes.c -o pipes.o

$ gcc serial.o pipes.o tcc.c -o tcc

A opção "-c" cria os arquivos objeto.

Caso exista um volume muito grande de arquivos objeto a gerar, o utilitário *make* é usado, de forma a recompilar somente os arquivos que foram realmente

modificados e definindo uma seqüência de arquivos a serem compilados em um arquivo chamado *makefile*.

Existe também a possibilidade de utilização de *warnings*, que podem detectar problemas com ponteiros, ou até passagem de parâmetros. Entre eles destacam-se -*pedantic*, -*Wall*, -*W*, -*Wtraditional*, -*Wshadow*, -*Wpointer-arith*, -*Wbad-function*, -*cast*, -*Wcast-qual*, -*Wcast-align*, -*Wwrite-strings*, -*Wconversion*, -*Waggregate-return*, -*Wmissing-prototypes*, -*Wmissing-declarations*, -*Wnested-externs*, -*Winline,* -*Wwrite-strings*. Para um maior detalhamento das funcionalidades destes "warnings", deve-se consultar o manual e o arquivo info do compilador "gcc".

Uma particularidade interessante do compilador gcc é que o mesmo não possui a biblioteca <conio.h>, conhecida nos compiladores C para DOS. Esta biblioteca é substituída por uma biblioteca chamada <curses.h>.

4.4 Processo de Login

Segundo Henderson (2003), quando o sistema operacional Linux inicializa, o *kernel* cria o processo *init*. É o primeiro e último processo existente em qualquer sistema operacional Linux. Todos os processos restantes são criados pelo processo *init* ou por um outro que derive dele. Geralmente o processo *init* executa um programa chamado "Sysvinit" ou similar. Pode-se executar qualquer programa tal como o *init*, nomeando o executável nos parâmetros de boot do Linux. O "Sysvinit" recebe as instruções contidas no arquivo */etc/inittab*.

O arquivo */etc/inittab*, possui instruções para início de vários processos executando o programa *getty*, ou seja, um processo por console virtual. Por exemplo no terminal */dev/tty5*. A linha "c5:235:respawn:/sbin/agetty 38400 tty5" que diz ao *init* para iniciar um processo executando *getty* no console virtual . Neste caso, o programa *getty* é */sbin/agetty* ou */sbin/mingetty* (HENDERSON, 2003).

O *getty* abre o terminal específico como arquivo de entrada padrão, saída padrão e erro padrão para os seus processos. Também atribui o terminal como sendo o "terminal de controle" para os processos, e define o dono e as suas permissões no dispositivo do terminal para algo seguro (eliminando quaisquer definições anteriores).

Exemplo : *prompt* do *login* que aparece no console virtual **/**dev/tty5**.** O kernel cria o processo *init*, que executa “Sysvinit”, que segue as instruções no arquivo */etc/inittab* e inicia outro processo executando um programa *getty*, com parâmetros identificando o terminal /dev/tty5. O programa getty imprime "*login:*" em /dev/tty5 e espera que alguém digite alguma coisa.

Depois de responder ao *prompt* de *login* do *getty*, o *gett*y executa o programa *login* (*getty* executa qualquer programa, mas */bin/login* é o normal), ou seja, o *getty* é substituído pelo *login,* mas é o mesmo processo. Lembre-se que este processo foi criado pelo *init*, que pertence ao super usuário. Este processo que atualmente executa o *login* também pertence ao super usuário.

Inicialmente *login* pede a senha e após ser inserido ele determina se está correta ou não. Assumindo que é a senha está correta, o *login* procede então às seguintes tarefas:

a) Define o *user id* do processo para si mesmo.;

b) Define o *group id* do processo para o seu grupo;

c) Põe um registo na base de dados das contas de usuários (o arquivo utmp) indicando que está conectado (*logged in*). Isto é utilizado pelo comando *who*;

d) Define o processo como pertencente ao grupos restantes de que faz parte;

e) Define a diretório de trabalho atual do processo para o seu diretório (home);

f) Habilita o dono do terminal e define as devidas permissões.

A próxima tarefa do *login* é executar o interpretador de comandos (que pode ser um programa qualquer, mas normalmente é uma *shell*, ex.: */bin/bash*), ou seja, substitui-se pelo programa de *shell*. O *login* procura o nome de usuário no arquivo */etc/passwd* para encontrar toda as informações necessárias, tais como a sua senha, uid, e programa de shell. A *shell* executa o arquivo de perfil do sistema chamado */etc/profile* e o seu perfil pessoal (tipicamente o arquivo .profile no seu diretório *home*), e por fim imprime um p*rompt* de comandos (`$` ou `%`) no terminal. Somente a partir deste ponto é que houve uma conexão com o sistema operacional Linux.

Em determinadas situações especiais, pode-se automatizar o processo de *login*. Esta automação do *login*, consiste em abstrair as ações que o *init***,** *getty***,** *login*, e a *shell*

fazem sem que nos apareça o *prompt* do nome de “usuário” e da “senha”. Existem várias maneiras de realizar este processo e uma delas consiste na utilização do programa *qlogin*. O *qlogin* executa as funções do *getty* e do *login* e é chamado pelo *init*, tal como o *getty*. Sua última ação é chamar o programa de *shell*, como o *login*.

Então para possibilitar este efeito, deve-se substituir a linha no arquivo */etc/inittab* mostrada acima pela linha “c4:235:respawn:/sbin/qlogin /dev/tty5 user”. Isto faz com que o usuário *user* se conecte ao console virtual /dev/tty5 no momento do *boot* em vez de apresentar o *prompt* para o nome do usuário e da senha.

Segundo Henderson (2003), o processo *respawn* na linha acima indica que quando o processo termina, o *init* vai criar um novo processo para o substituir. Num sistema tradicional UNIX, significa que quando ocorre *logout* da *shell*, o que acarreta no fim do processo, um novo getty é executado e o terminal recebe um *prompt* de *login* para o próximo usuário.

Neste caso *qlogin*, significa que quando ocorre o *logout* da *shell*, uma nova shell aparece imediatamente para o substituir. As ações do programa *qlogin* são determinadas através de alguns parâmetros na linha de comandos, tais como:

--command - define o comando a ser executado depois de *qlogin***.** */bin/bash* é o normal.

--arg0 - é o argumento zero que o programa que executa depois do *qlogin* recebe, que é o que aparece no listagem do comando *ps*.

--uid - o *user id* numérico para o processo.

--gid - o *group id* numérico para o processo.

--homedir - o diretório *home* e diretório inicial e atual para o processo.

--utmp/--noutmp - determina se o *qlogin* usa a base de dados dos usuários para fazer o *login* (arquivo “utmp”).

4.5 SERVIÇOS TELNET E FTP

Segundo Redhat (2003), serviço é um programa ou conjunto de programas que devem ser executados para iniciar uma atividade em um servidor. Existem serviços que são considerados essenciais em um sistema operacional Linux, pois sem eles, o sistema perderia um pouco da sua flexibilidade. Por exemplo, sem um serviço para captar e registrar as mensagens geradas pelos vários outros programas em funcionamento haveria mais dificuldades em detectar problemas no sistema.

O serviço TELNET é utilizado para criar um console virtual acessível remotamente via rede e o serviço FTP permite a transferência e cópia de arquivos via rede. Estes serviços precisam ser configurados, para que possam ser inicializados pelo *xinet*

O xinet é uma alternativa segura e eficiente para o *inet,* o gerenciador de serviços de rede do Linux, e possui vários recursos que o *inet* não implementa tais como controle de acesso baseado em destino e hora, redirecionamentos, grande capacidade de *logs*, ligar serviços a interfaces específicas e limitar o número de máquinas simultâneas em um determinado serviço. Toda a configuração do *xinet* é feita através de um único arquivo, o */etc/xinetd.conf*.

O serviço *telnet* pode ser habilitado, caso exista a necessidade de criação de um console virtual remoto via rede. O quadro 5 apresenta a área de configuração para o serviço TELNET.

Quadro 5 : Configuração do serviço TELNET

```
# default: on
# description: The telnet server serves telnet sessions; it uses \
#     unencrypted username/password pairs for authentication.
service telnet
{
      flags             = REUSE
      socket_type       = stream
      wait              = no
      user              = root
      server            = /usr/sbin/in.telnetd
      log_on_failure    += USERID
      disable           = no
}
```

A configuração do serviço FTP é necessária, caso exista a necessidade de transferência de arquivos via rede através do protocolo FTP. O quadro 6 apresenta a configuração para o serviço de FTP.

Quadro 6 : Configuração do serviço FTP

```
# default: on
# description: The wu-ftpd FTP server serves FTP connections.It uses \
```

```
#     normal, unencrypted usernames and passwords for authentication.
service ftp
{     socket_type       = stream
      wait              = no
      user              = root
      server            = /usr/sbin/in.ftpd
      server_args       = -l -a
      log_on_success    += DURATION
      nice              = 10
      disable           = no
}
```

4.6 Programa mpg321

O programa mpg321 é o substituto versão *freeware* do mpg123, um *player* MP3 de linha de comando muito popular. Ele é largamente usado para *frontends*, bem como *players* de MP3 ou então como conversor MP3 para *Wave* (para gravação de CDs de áudio padrão). A grande característica deste *player* está no fato do mesmo utilizar a biblioteca MAD (*Mpeg Audio Decoder*), que é livre e possui uma saída com amostras de 24 bits. Foi desenvolvida por Rob Leslie (LESLIE, 2003).

O *player* mpg321 possui uma interface para controle remoto, através da opção -R, que é útil se exite a necessidade de desenvolvimento um *frontend* para o mpg321, com controle de playback. Para ativar esta interface existe a seguinte sintaxe :

mpg123 -R abcd (onde, 'abcd' pode ser qualquer *string*. Completa a sintaxe da opção -R).

Todos estes comandos podem ser ativados também através da primeira letra dos mesmos.

Os comandos disponíveis nesta interface são vistos no quadro 7.

Quadro 7: Comandos de interface do mpg321

```
LOAD <file> -- Carrega e inicia a reprodução do arquivo mp3 <file>.
JUMP [+-]<frames> -- Se os argumentos '+' ou '-' são passados, então
salta <frames> frames no arquivo mp3 avançando ou retrocedendo,
respectivamente. Se nada é especificado, então salta para o frame
absoluto <frames>.
PAUSE -- pausa a reprodução do arquivo mp3. Se já está pausado então
reinicia a reprodução.
STOP -- para a reprodução do arquivo mp3.
QUIT -- Abandona o mpg321.
```

Ele também gera informações de status identificados por @n. O quadro 8 apresenta o status de interface do mpg321.

Quadro 8: Status de interface do mpg321

```
@R MPG123 -- Indica inicio de operação da interface do mpg321.
@I mp3-filename -- Indica o nome do arquivo sem a extensão, após o
arquivo mp3 ser carregado.
@S <a> <b> <c> <d> <e> <f> <g> <h> <i> <j> <k> <l> -- Indica
informações de saída do arquivo mp3, após o arquivo mp3 ser carregado.
Abaixo é apresentada cada informação em detalhes.
<a>: versão do arquivo mp3. Atualmente 1.0 (madlib).
<b>: layer: 1, 2, ou 3. Inteiro.
<c>: Samplerate. Inteiro.
<d>: Modo string. String.
<e>: Modo extension. Inteiro.
<f>: Bytes por frame (aproximado). Inteiro.
<g>: Número de canais (1 ou 2, usualmente). Inteiro.
<h>: Copyright? (1 ou 0). Inteiro.
<i>: CRC ? (1 ou 0). Inteiro.
<j>: Enfase. Inteiro.
<k>: Bitrate, em kbps (por exemplo 128) Inteiro.
<l>: Extensão. Inteiro.
@F <current-frame> <frames-remaining> <current-time> <time-remaining>
-- Indica o status de atualização dos frames decodificados. Current-
frame e frames-remaining são inteiros e current-time e time-remaining
são de ponto flutuante com duas casas decimais.
@S {0, 1, 2} -- Indica status Stop/pause.
0 - reprodução parada. Ocorre após 'STOP' ou no final do arquivo MP3.
1 - reprodução está pausada. É necessário 'PAUSE' para continuar.
2 - reprodução iniciou novamente.
```

4.7 Programa aumix

O programa *aumix* serve para ajustar os parâmetros de um dispositivo de áudio. Ele pode ser usado de uma linha de comando, em *scripts* ou interativamente através de teclado ou *mouse*. Pode controlar diversos dispositivos, tais como, volume principal, *bass*, CD, *line in*, microfone, *line out*, sintetizador, *treble*, PCM, *imix*. O quadro 9 ilustra algumas das opções possíveis para o programa aumix.

Quadro 9: Opções de execução de aumix

```
-v ajusta o volume principal
-b ajusta o nível de bass
-c ajusta o volume do CD
-i ajusta o volume de line in
-m ajusta o volume do microfone
-o ajusta o volume de line out
-s ajusta o volume do sintetizador
-t ajusta o nível de treble
-x ajusta o nível de imix
-q apresenta os ajustes atuais de todos os dispositivos
```

Exemplo : aumix -q -v75 -m q

Este comando imprime na tela o ajuste atual de todos dispositivos, ajusta o volume final para 75%, e imprime-o novamente com o novo valor conforme abaixo :

Vol 75, 75, R

4.8 Programa mp3info

O programa mp3info é um pequeno utilitário utilizado para ler e modificar ID3 *Tags* dos arquivos MP3 no sistema operacional Linux. Ele pode mostrar também vários aspectos técnicos dos mesmos, tais como tempo, *bit rate*, freqüência e outros atributos.

Possui dois modos de execução, sendo eles, o modo interativo com a utilização da biblioteca *curses*, ou o modo de linha de comando. O quadro 10 ilustra algumas opções possíveis para o modo linha de comando. Os argumentos precedidos por "%" são usados para consultar informações diversas dos arquivos MP3 (IBIBLIO, 2001).

Quadro 10: Opções de execução do mp3info

```
-a artista        Especifica o nome do artista presente no ID3
-c comentário     Especifica o comentário presente no ID3
-g gênero         Especifica o gênero presente no ID3
-l álbum          Especifica o nome do álbum presente no ID3
-i    Edita o ID3 Tag interativamente
-d    Deleta o ID3 Tag (se existir)
%f    Nome do arquivo sem o path (string)
%a    Exibe o nome do artista (string)
%c    Exibe o comentário (string)
%g    Exibe o gênero (string)
%l    Exibe o nome do álbum (string)
%n    Exibe o número da faixa (integer)
%t    Exibe o nome da música (string)
%y    Exibe o ano da música (string)
%S    Exibe o tempo total da música em segundos (integer)
```

Exemplo: mp3info –p %a /Tcc/MP3/cpm.mp3

Irá consultar e exibir a informação artista do arquivo cpm.mp3 e irá apresentar a *string* CPM22 como resultado.

5 Comunicação de dados

5.1 Comunicação infravermelho utilizando a Norma RC5

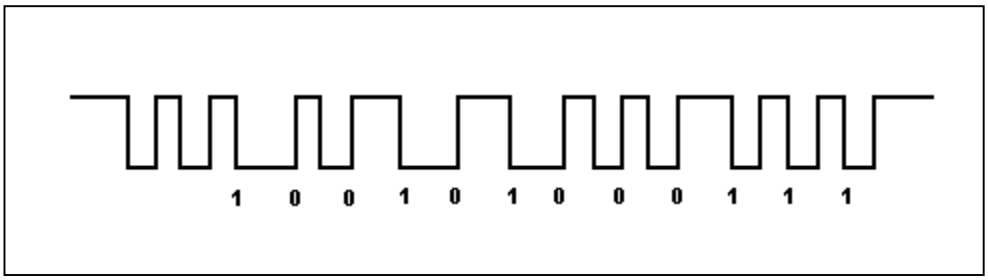
Segundo Kainka (2002), RC5 é uma norma universal para comandos a distância por infravermelho utilizada principalmente em equipamentos de áudio, televisores, videocassetes e outros aparelhos domésticos, com uma área de alcance de aproximadamente 10m.

O conjunto de códigos da norma RC5 foi desenvolvido pela Phillips e possui 2048 comandos divididos em 32 grupos endereçáveis de 64 comandos cada. O código transmitido consiste de uma palavra de 14 bits, sendo eles :

- 2 bits para ajuste do nível AGC do receptor (2 start bits). O primeiro é sempre 1 e o segundo corresponde a 1 se o código de comando está entre 0-63 e 0 se está entre 64-127;
- 1 bit para controle (*check bit*) que muda de estado lógico cada vez que um botão é pressionado na unidade de comando a distância. Isto serve para indicar se o botão foi pressionado uma vez ou se continua sendo pressionado;
- 5 bits de endereço do sistema para seleção de 1 dos 32 sistemas possíveis listados na tabela 7. Isso define o tipo de aparelho que se pretende controlar.;
- 6 bits de comando representando 1 dos 128 comandos possíveis. Estes comandos são listados na tabela 8. Isso define a ação que se pretende executar em um determinado aparelho (sistema) selecionado;

Tanto no endereço do sistema, quanto no comando o *bit* menos significativo é transmitido primeiro. A figura 18 mostra um pacote de dados norma RC5.

Figura 18: Pacote de dados Norma RC5



Os primeiros 6 bits da direita para a esquerda, correspondem ao comando (1+2+4+0+0+0), os próximos 5 bits correspondem ao sistema (1+0+4+0+0). A decodificação do pacote da figura 18 apresenta os seguintes dados :

Controle = 1 Sistema = 5 Comando = 7

Tabela 7: Endereços de sistema Norma RC5

Número	**Sistema (aparelho)**
0	Televisor 1

1	Televisor 2
2	Teletexto
3	Televisor 1 e 2 (comandos de 0-61 ou comandos de 64-127)
5	Videocassete 1
6	Videocassete 2
7	Experimental
16	Pré-amplificador de áudio
17	Rádio
18	Tape
19	Experimental
23	DAT

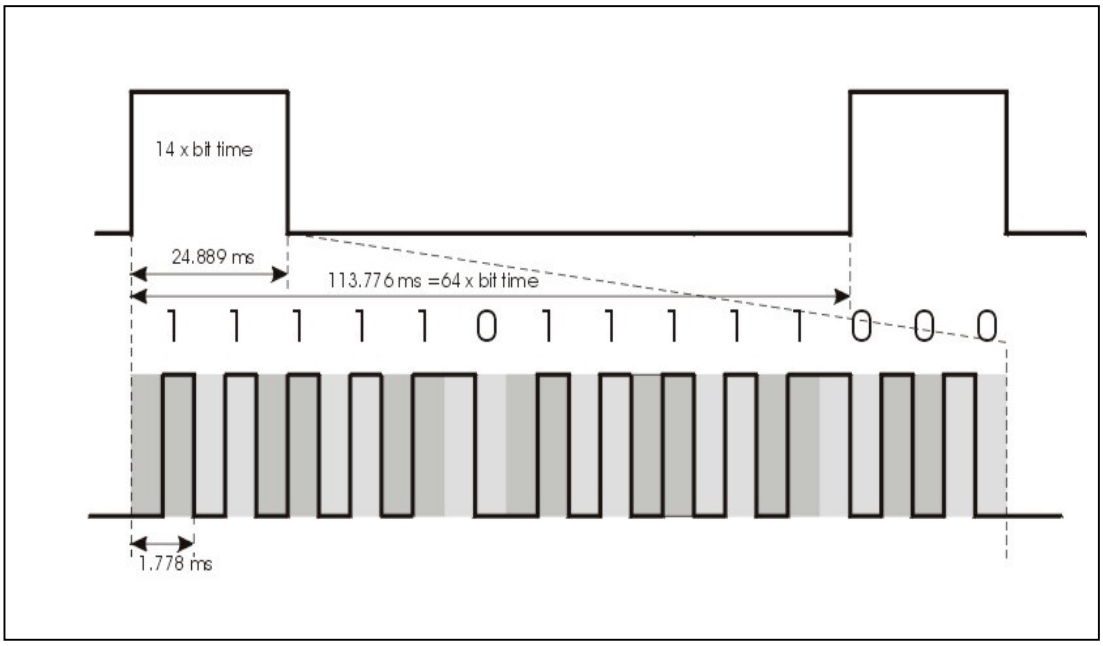
Tabela 8: Identificação de comandos Norma RC5

Número	**Função**
0-9	0-9
12	Standby
13	Mute
14	Presets
16	Volume +
17	Volume -
18	Brilho +
19	Brilho -
20	Cor +
21	Cor -
22	Bass +
23	Bass -
24	Treble +
25	Treble -
26	Balanço direita
27	Balanço esquerda
48	Pause
50	Reverse rápido
52	Avanço rápido
53	Play
54	Stop
55	Record

Na norma RC5, os dados são modulados numa freqüência portadora de 30 a 40KHz. O transmissor emite por rajadas (salvas) onde estão contidos os pacotes de dados. Cada *bit* transmitido tem 1,778ms de duração, enquanto que cada pulso curto tem 6,9444µs de duração e 20,8332µs de intervalo. Para uma freqüência portadora de 36Khz, cada salva curta é formada por 32 impulsos e cada salva longa, por 64 impulsos.

A palavra completa dura 24,889ms, e é sempre transmitida completamente. Se um botão do comando a distância é mantido pressionado, então o código é repetido em intervalos de 64 impulsos(113,778ms), conforme ilustra a figura 19.

Figura 19: Intervalo de transmissão do pacotes de dados Norma RC5



A norma RC5 utiliza sinais bifásicos, em que a informação está contida nas alterações de fase. A amplitude do sinal muda de estado pelo menos a cada 1,776ms e o receptor pode manter-se sincronizado com o transmissor mediante essas alterações de amplitude. Os sinais começam sempre com a mesma seqüência. Depois, seguem-se três regiões de dados, em que as alterações de amplitude espaçadas de 1,776ms representam os bits de dados. Os sinais descendentes representam estados lógicos altos 1 e os sinais ascendentes representam estados lógicos baixos 0. A seguir a cada alteração de amplitude, o integrado receptor espera um pouco mais de 0,888ms e esquece qualquer alteração de amplitude que possa ocorrer nesse intervalo de tempo. A próxima alteração de amplitude (sinal) representa um bit de dados e serve também para a sincronização do sinal de relógio (KAINKA, 2002).

Uma unidade de comando a distância que possui o protocolo Norma RC5 é o controle remoto universal RCA Systemlink 4 já comentado no item 2.3.4. As tabelas 9 e 10 apresentam as funções disponíveis para os sistemas de videocassete e televisor.

Tabela 9: Videocassete 1 - Funções disponíveis – RCA Systemlink 4

Botão	**Número**

0..9	0..9
CH+	32
CH-	33
CH Prev	49
On-Off	12
Enter	40
TV-VCR	62
Rew	50
Play	53
FF	52
Rec	55
Pause	41
Stop	54

Tabela 10: Televisor 1 – Funções disponíveis – RCA Systemlink 4

Botão	**Número**
0..9	0..9
Mute	13
CH+	32
CH-	33
Vol+	16
Vol-	17
On-Off	12
Enter	40
CH Prev	49

5.2 Comunicação serial RS232

RS é uma abreviação de "*Recommended Standard*" e descreve a padronização de uma interface comum para comunicação de dados entre equipamentos, criada no início dos anos 60, por um comitê conhecido atualmente como "*Electronic Industries Association*" (EIA). Naquele tempo, a comunicação de dados compreendia a troca de dados digitais entre um computador central (*mainframe*) e terminais de computador remotos, ou entre dois terminais sem o envolvimento do computador. Estes dispositivos poderiam ser conectados através de linha telefônica, e conseqüentemente necessitavam de um *modem* em cada lado para fazer a decodificação dos sinais (CANZIAN, 2002).

Este foi o ponto de partida para o padrão RS232. Ele especifica as tensões, temporizações e funções dos sinais, um protocolo para troca de informações, e as conexões mecânicas. Desde que essa padronização foi desenvolvida, a mais de 30 anos,

a EIA publicou três modificações, sendo que a mais recente, EIA232E, foi introduzida em 1991, com a mudança de nome de RS232 para EIA232 e algumas linhas de sinais renomeadas e várias linhas novas definidas.

Embora tenha sofrido poucas alterações, muitos fabricantes adotaram diversas soluções mais simplificadas que tornaram impossível a simplificação da padronização proposta. As maiores dificuldades encontradas pelos usuários na utilização da interface RS232 incluem pelo menos um dos seguintes fatores:

- A ausência ou conexão errada de sinais de controle que resultam em estouro do buffer (*overflow*) ou travamento da comunicação;
- Função incorreta de comunicação para o cabo em uso que resultam em inversão das linhas de transmissão e recepção, bem como a inversão de uma ou mais linhas de controle (*handshaking*).

Se a norma EIA232 completa for implementada, o equipamento que faz o processamento dos sinais é chamado DTE (*Data Terminal Equipment* – usualmente um computador ou terminal) que possui um conector DB25 macho, e utiliza 22 dos 25 pinos disponíveis para sinais ou terra. Já o equipamento que faz a conexão (normalmente uma interface com a linha telefônica) é denominado de DCE (*Data Circuit-terminating Equipment* – usualmente um modem) e tem um conector DB25 fêmea utilizando os mesmos 22 pinos disponíveis para sinais e terra. Um cabo de conexão entre dispositivos DTE e DCE contém ligações em paralelo, não necessitando mudanças na conexão de pinos. Se todos os dispositivos seguissem essa norma, todos os cabos seriam idênticos, e não haveria chances de haver conexões incorretas.

Um cabo *null modem* é utilizado para conectar dois DTEs juntos. Isto é comumente usado como um meio barato para transferir arquivos entre computadores utilizando protocolos Zmodem, Xmodem, etc. Ele também pode ser utilizado em diversos sistemas de desenvolvimento. Na tabela 11 é apresentado um método de conexão de um cabo *null modem*. Apenas 3 fios são necessários (TxD, RxD e GND).

Tabela 11: Ligação cabo *Null modem*

DB9	DB25
3	2
2	3
5	7

4	20
6	6
1	8
7	4
8	5

A teoria de operação é razoavelmente simples. O princípio é fazer o DTE pensar que está falando com um *modem*. Qualquer dado transmitido do DTE deve ser recebido no outro extremo e vice-versa. O sinal de terra (SG) também deve ser conectado ao terra comum dos dois DTEs.

O sinal DTR é conectado com os sinais DSR e CD nos dois extremos. Quando o sinal DTR for ativado (indicando que o canal de comunicação está aberto), imediatamente os sinais DSR e CD são ativados. Nessa hora o DTE pensa que o modem virtual ao qual está conectado está pronto e que foi detectado uma portadora no outro *modem*. O DTE precisa se preocupar agora com os sinais RTS e CTS. Como os 2 DTEs se comunicam à mesma velocidade, o fluxo de controle não é necessário e conseqüentemente essas 2 linhas são conectadas juntas em cada DTE. Quando o computador quer transmitir um dado, ele ativa as linhas RTS como estão conectadas juntas, imediatamente recebe a resposta que o outro DTE está pronto pela linha CTS.

Note que o sinal RI não está conectado em nenhum extremo. Esta linha é utilizada apenas para informar ao DTE que existe um sinal de chamada telefônica presente. Como não existe modem conectado a linha telefônica ela pode permanecer desconectada.

Existem outros trabalhos que podem esclarecer maiores detalhes sobre o protocolo RS232, visto que o sistema proposto, apresenta pontos de estudo mais relevantes do que a comunicação serial RS232, que já vem sendo bastante esclarecida devido ao seu grau de utilidade absoluto na área de comunicação de dados.

5.3 Protocolo FTP

Transferência de arquivos significa copiar arquivos de um computador para outro.

O protocolo FTP é o principal método de transferência de arquivos na *internet* ou rede local. Em princípio, a transferência de arquivos entre dois computadores, pressupõe que exista uma permissão de acesso (*username)* nos dois sistemas (TCHE, 2003).

O FTP é usado, geralmente, quando você possui três informações:

- o endereço Internet ou da rede local onde o arquivo que você está interessado se encontra;
- o diretório no qual o arquivo está armazenado neste sistema;
- o nome do arquivo.

6 MP3

MP3 é a abreviação de MPEG *Audio Layer* III, o formato de arquivo que comprime faixas de áudio padrão em tamanhos extremamente reduzidos sem comprometer a qualidade do som. Consiste num formato de arquivos de áudio que vem se tornando cada vez mais popular. Estas músicas tem qualidade de CD com taxas de

compressão 12:1. A compressão usada no MP3 contém significativamente mais capacidade de armazenamento por *megabyte* do que a maioria dos outros algoritmos de compressão, porém algoritmos de compressão/descompressão de MP3 são exaustivamente complexos (MPEG ORG, 2003).

O *encoder* MPEG usa um modelo psico-acústico quando executando uma forma de onda, garantindo descartar partes do sinal que não são detectáveis para o sistema auditivo humano. Estas partes descartadas são geralmente referidas como irrelevantes. O modelo psico-acústico quebra o sinal em blocos que são aplicados a uma transformação de freqüência. Ele então modela o sistema auditivo humano dentro do sinal e estima o nível de ruído. Em paralelo, o *coder* MPEG determina o conteúdo spectral do sinal e então codifica este dado com a parte descartada para criar uma *stream* de saída. Contudo a análise MPEG é muito complexa, e a reconstrução é muito mais simples, mas continua requerendo uma grande quantidade de processamento.

Esta análise permite que o *decoder* seja alojado em um processador de sinais digitais (DSP) que pode ser incorporado em uma aplicação de *decoder* MPEG portátil, ou então utilizar os *decoders* instalados em PCs, tais como Winamp, RXAudio, Sonique, Mpg123, etc.. Esta segunda opção é a mais popular devido a popularidade dos PCs, na qual o *decoder* de MP3 torna-se um entre vários *softwares* disponíveis, independente do sistema operacional utilizado (MPEG ORG, 2003).

O áudio MPEG é composto de fases e camadas. Correntemente existem quatro camadas definidas, sendo elas :

- MPEG-1 – define um canal simples mono e um canal duplo *stereo*, com taxas de amostra de 32,44.1 e 48KHz. A faixa de *Bitrate* é de 32 a 448Kbps para a camada 1, 32 a 384Kbps para camada 2 e 32 a 320Kbps para a camada 3;
- MPEG-2 BC – define uma extensão Multi-Canal do MPEG-1. Ele possui acima de 5 canais e um canal de baixa freqüência (tipicamente chamado 5.1) capaz de *bitrates* acima de 1Mbps. Também estende taxas de amostra inferiores a 16, 22.05 e 24KHz para *Bitrates* de 32 a 256Kbps na camada 1 e de 8 a 160Kbps para camada 2 e 3;

- MPEG-2 ACC – contém um *coder* padrão de alta qualidade de 1 a 48 canais, taxas de amostra de 8 a 96KHz e *Bitrates* de 8 a 160Kbps. Esta fase não é compatível com MPEG-1 ou MPEG-2 BC;
- MPEG-4 – será usado código natural e objetos de áudio em uma extensa faixa de *Bitrates*.

A camada do áudio MPEG define basicamente a complexidade, o tempo e a eficiência do *encoder* e *decoder*. A camada 1 corresponde a uma compressão 1:4, camada 2 de 1:6 a 1:8, e camada 3 de 1:10 a 1:12. Faixas de áudio digitais normais consistem de amostras de 16 *bits* gravadas em taxas de amostras acima de 44.1KHz (CD). Isto indica um *Bitrate* de 1.4Mbps de música *stereo* com qualidade de CD. Usando um *coder* de áudio MPEG, o *Bitrate* pode ser reduzido para uma faixa de 32 a 384Kbps, mantendo a qualidade de som original. A camada 3 possui as melhores alternativas : para uma determinada qualidade de som, ela fornece o menor *Bitrate* ou para determinado *Bitrate*, ela fornece a melhor qualidade de som.

O arquivo de áudio MPEG é composto de muitos *frames*, e em muitos casos, de um *Audio Tag*. Cada *frame* é geralmente independente de todos os outros *frames* e contém um cabeçalho de frames especificando informações sobre o arquivo como *Bitrate*, fase MPEG, camada MPEG, etc. Isto indica que cada *frame* pode ter um *Bitrate* diferente.

Desde que não haja nenhum cabeçalho, qualquer arquivo de áudio camada MPEG 1 ou 2 pode estar corrompido em qualquer *frame*, mas pode continuar decodificando normalmente. A camada 3 utiliza algumas técnicas apuradas para manter o *Bitrate* e qualidade e alguns frames são dependentes de outros. Os quatro primeiros bytes de cada frame constituem o cabeçalho do frame. O formato básico do frame é mostrado na tabela 12. Estas definições seguem a sintaxe :

AAAAAAAAAAABBCCDEEEEFFGHIIJJKLMM

Tabela 12: Identificação de informações de frames MP3

<table>
<tr><th>Field</th><th>Bits</th><th>Description</th></tr>
<tr><td>A</td><td>31-21</td><td>Frame Sync (all bits set)</td></tr>
<tr><td>B</td><td>20-19</td><td>MPEG Audio Version
MPEG Version 2.5 is not an official standard, therefore in most systems bit 20 is part of the frame sync and must be set. Following the above definition for bits 19 & 20, allows a system to distinguish all three versions but still be ISO compliant.
00 – MPEG Version 2.5
01 – Reserved
01 – Reserved
10 – MPEG Version 2
11 – MPEG Version 1</td></tr>
<tr><td>C</td><td>18-17</td><td>MPEG Layer
00 – Reserved
01 – Layer III
10 – Layer II
11 – Layer I</td></tr>
<tr><td>D</td><td>16</td><td>Protection bit
0 – Protected by a CRC (16-bit CRC follows the frame header)
1 – Not protected</td></tr>
<tr><td>E</td><td>15-12</td><td>Bitrate Index (in Kbps)
<table>
<tr><th>Bits</th><th>MPEG-1
Layer I</th><th>MPEG-1
Layer II</th><th>MPEG-1
Layer III</th><th>MPEG-2
Layer I</th><th>MPEG-2
Layer II & III</th></tr>
<tr><td>0000</td><td>free</td><td>free</td><td>free</td><td>free</td><td>free</td></tr>
<tr><td>0001</td><td>32</td><td>32</td><td>32</td><td>32</td><td>8</td></tr>
<tr><td>0010</td><td>64</td><td>48</td><td>40</td><td>48</td><td>16</td></tr>
<tr><td>0011</td><td>96</td><td>56</td><td>48</td><td>56</td><td>24</td></tr>
<tr><td>0100</td><td>128</td><td>64</td><td>56</td><td>64</td><td>32</td></tr>
<tr><td>0101</td><td>160</td><td>80</td><td>64</td><td>80</td><td>40</td></tr>
<tr><td>0110</td><td>192</td><td>96</td><td>80</td><td>96</td><td>48</td></tr>
<tr><td>0111</td><td>224</td><td>112</td><td>96</td><td>112</td><td>56</td></tr>
<tr><td>1000</td><td>256</td><td>128</td><td>112</td><td>128</td><td>64</td></tr>
<tr><td>1001</td><td>288</td><td>160</td><td>128</td><td>144</td><td>80</td></tr>
<tr><td>1010</td><td>320</td><td>192</td><td>160</td><td>160</td><td>96</td></tr>
<tr><td>1011</td><td>352</td><td>224</td><td>192</td><td>176</td><td>112</td></tr>
<tr><td>1100</td><td>384</td><td>256</td><td>224</td><td>192</td><td>128</td></tr>
<tr><td>1101</td><td>416</td><td>320</td><td>256</td><td>224</td><td>144</td></tr>
<tr><td>1110</td><td>448</td><td>384</td><td>320</td><td>256</td><td>160</td></tr>
<tr><td>1111</td><td>bad</td><td>bad</td><td>bad</td><td>bad</td><td>bad</td></tr>
</table></td></tr>
<tr><td>F</td><td>11-10</td><td>Sample rate (in KHz)
<table>
<tr><th>Bits</th><th>MPEG-1</th><th>MPEG-2</th><th>MPEG-2.5</th></tr>
<tr><td>00</td><td>44.1</td><td>22.05</td><td>11.025</td></tr>
<tr><td>01</td><td>48</td><td>24</td><td>12</td></tr>
<tr><td>10</td><td>32</td><td>16</td><td>8</td></tr>
<tr><td>11</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>G</td><td>9</td><td>Pad bit Pad bit is used to match frame length to bitrate
0 – Frame is not padded
1 – Frame is padded</td></tr>
<tr><td>H</td><td>8</td><td>Private Bit Application dependent</td></tr>
<tr><td>I</td><td>7-6</td><td>Channel Mode Bit
00 – Stereo
01 – Joint Stereo
10 – Dual Channel Stereo
11 – Single Channel Mono</td></tr>
<tr><td>J</td><td>5-4</td><td>Mode Extension Only used for Joint Stereo channel mode</td></tr>
<tr><td>K</td><td>3</td><td>Copyright
0 – Audio is not copyrighted
1 – Audio is copyrighted</td></tr>
<tr><td>L</td><td>2</td><td>Original
0 – Copy of the original media
1 – Original media</td></tr>
<tr><td>M</td><td>1-0</td><td>Emphasis
00 – None
01 – 50/15ms
10 – reserved
11 – CCIT J.17</td></tr>
</table>

O Audio *Tag* MPEG (ID3) contém informações sobre o artista, nome da música, álbum, etc para o arquivo MP3. Ele está localizado nos últimos 128 bytes do arquivo de áudio. O *Tag* ID3 é mostrado em um descritor similar ao cabeçalho do frame. Ele tem o formato definido nos campos designados de A até G. A descrição dos campos de bytes *Tag* ID3v1 estão indicados na tabela 13. Estas definições seguem a sintaxe :

AAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDEEEEFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFG

Tabela 13: Descrição dos campos ID3v1

Field	Bytes	Description
A	0-2	Tag Identification Must contain the ASCII characters TAG if the tag exists
B	3-32	Title of song
C	33-62	Artist of song
D	63-92	Album title
E	93-96	Year song was made
F	97-126	Comment
G	127	Genre

0 – Blues	20 – Alternative	40 – AlternRock	60 – Top 40
1 – Classic Rock	21 – Ska	41 – Bass	61 – Christian Rap
2 – Country	22 – Death Metal	42 – Soul	62 – Pop/Funk
3 – Dance	23 – Pranks	43 – Punk	63 – Jungle
4 – Disco	24 – Soundtrack	44 – Space	64 – Native American
5 – Funk	25 – Euro-Techno	45 – Meditative	65 – Cabaret
6 – Grunge	26 – Ambient	46 – Instrumental Pop	66 – New Wave
7 – Hip-Hop	27 – Trip-Hop	47 – Instrumental Rock	67 – Psychadelic
8 – Jazz	28 – Vocal	48 – Ethnic	68 – Rave
9 – Metal	29 – Jazz+Funk	49 – Gothic	69 – Showtunes
10 – New Age	30 – Fusion	50 – Darkwave	70 – Trailer
11 – Oldies	31 – Trance	51 – Techno-Industrial	71 – Lo-Fi
12 – Other	32 – Classical	52 – Electronic	72 – Tribal
13 – Pop	33 – Instrumental	53 – Pop-Folk	73 – Acid Punk
14 – R&B	34 – Acid	54 – Eurodance	74 – Acid Jazz
15 – Rap	35 – House	55 – Dream	75 – Polka
16 – Reggae	36 – Game	56 – Southern Rock	76 – Retro
17 – Rock	37 – Sound Clip	57 – Comedy	77 – Musical
18 – Techno	38 – Gospel	58 – Cult	78 – Rock & Roll
19 - Industrial	39 - Noise	59 - Gangsta	79 – Hard Rock

Todos os bytes são preenchidos com caracteres nulos, contudo, nem todas as aplicações respeitam este formato e usam caracteres espaço. ID3v1.1 propõe que o último byte no campo de comentários pode ser usado para especificar o número da faixa da música no álbum.

7 Desenvolvimento

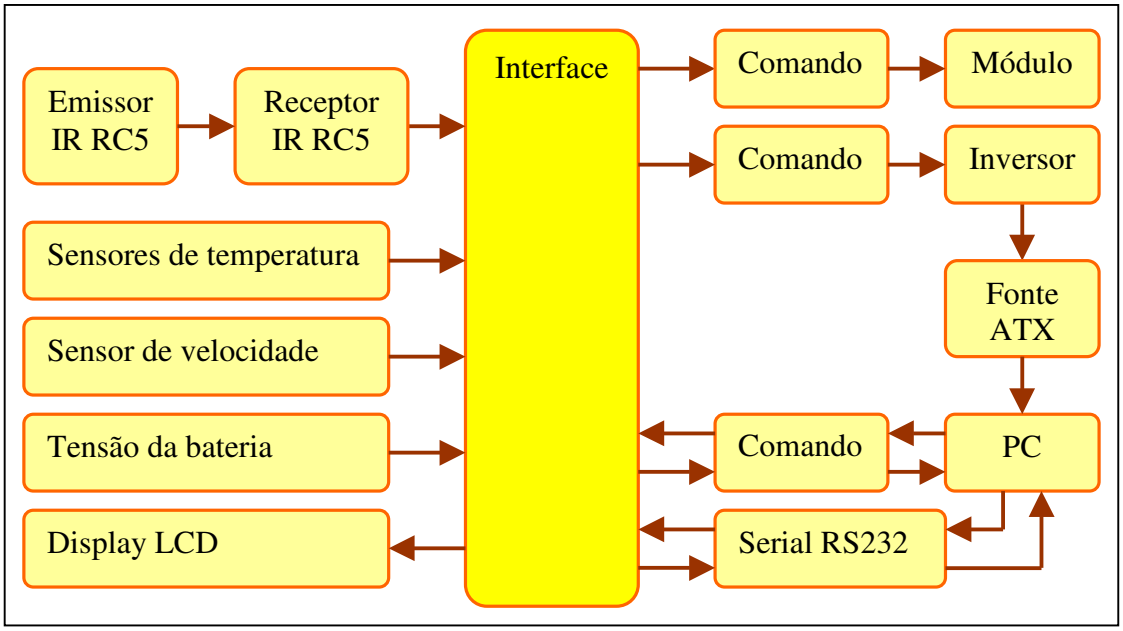
7.1 Funcionalidades

As principais funcionalidades do computador de bordo desenvolvido são :

- Indicação de temperaturas interna e externa ao veículo em °C;
- Controle de nível de carga da bateria automotiva;
- Indicação e controle de velocidade do veículo em Km/h, com indicação de alertas e registros de velocidades superiores à permitida;
- Indicação de data e hora atuais via comunicação serial com o PC;
- Reprodução de arquivos MP3 com controle de volume;
- Interação homem/máquina através de *display* LCD 4x20;
- Comandos emitidos via emissor infravermelho Norma RC5.

Estas funcionalidades são divididas entre a interface microcontrolada e pelo PC, sendo a interface a principal responsável pela execução das mesmas, pois controla diretamente o PC. A interface recebe dados via sensor infravermelho Norma RC5 para controlar o PC. Já a comunicação entre a interface e o PC é serial, protocolo RS232. A atualização de arquivos MP3 e do arquivo de velocidades do PC ocorre via FTP. A figura 20, ilustra a relação entre os dispositivos do computador de bordo desenvolvido.

Figura 20: Representação do computador de bordo desenvolvido



7.2 Especificação do software da interface

Dentre as principais funções do *software* da interface destacam-se :

- Leitura de valores de temperatura através dos conversores A/Ds internos;
- Leitura da tensão da bateria também através dos A/Ds internos;
- Leitura de velocidade através de um pino de I/O definido, captando os sinais gerados pelo sensor de velocidade efeito *Hall*;
- Interpretação dos sinais infravermelhos Norma RC5 decodificando cada pacote recebido, para a leitura do sistema e do comando que deve ser executado, via interrupção;
- Comando direto sobre o PC, substituindo os botões Pwr_On e Rst_Sw;
- Comando direto sobre o sinal remoto do módulo de potência;
- Comando direto sobre a alimentação do PC, ligando e desligando o inversor de tensão;
- Comunicação serial com o PC, com recepção através de interrupção para leitura do status do PC, informações ID3 *Tag* das músicas de MP3, tempo decorrido de execução das músicas, status de execução das músicas (*stop*,*pause*,*play*), velocidade máxima permitida sem emissão de alertas e informações de data e hora;
- Comunicação serial com o PC, com transmissão de dados, sem interrupção, para instruções de controle sobre as músicas MP3 (*play*,*pause*,*stop*,*next*,*previous*), controle de volume, velocidade máxima permitida pelo veículo sem emissão de alertas, status de carga da bateria, desligando o PC caso necessário, bem como desligar o PC quando houver este comando via emissor infravermelho.

Para a especificação da interface microcontrolada, foi utilizado o método dos fluxogramas.

7.2.1 *Interface*

Em primeiro momento, a interface (figuras 21, 22 e 23) inicia a execução e apresentação de uma mensagem de "saudação" e com a leitura temporizadora de "Temperatura 1" e "Temperatura 2" (figura 34) e nível de carga da bateria (figura 33), através de sua tensão nominal. Caso a bateria esteja em condições normais, e o veículo esteja parado, será somente apresentado os valores de temperatura e a mensagem de saudação. Quando o veículo estiver em movimento, também será apresentada a velocidade média do mesmo em "Km/h" (figura 35), porém sem emissão de alertas, pois o valor de referência está atribuído ao PC e será passado à interface via porta serial, somente quando o PC estiver ligado.

Para acionar a função de MP3, o usuário irá pressionar o botão "On-Off" do emissor infravermelho. Neste momento, o inversor de tensão do PC será ligado, e o mesmo receberá um pulso "Pwr_On" (figura 25) para ligar o PC (a mesma ação de acionamento do botão Liga-Desliga em um gabinete ATX). Durante o funcionamento do PC, as informações antes apresentadas em tela cheia, serão apresentadas na 4ª linha do *display* LCD, seguindo as mesmas temporizações.

Estando o PC ligado, a interface irá aguardar por um dado de controle recebido via serial (figuras 26 e 27), para saber se o PC está pronto para executar as músicas MP3 e outras tarefas. A partir deste momento a interface irá também apresentar informações de data e hora no *display* LCD, começará a emitir alertas caso a velocidade máxima configurada seja ultrapassada e irá acionar o sinal remoto do módulo de potência para que todo sistema sonoro do automóvel esteja habilitado.

As músicas MP3, também poderão ser executadas através de comandos via emissor infravermelho (figura 28). Durante a execução das músicas a interface receberá serialmente o ID3 *Tag* das músicas (Nome da música e Artista e *Stereo-Mono*), apresentando-os, respectivamente, na 1ª e 2ª linhas e tempo decorrido da música, juntamente com *status* de execução (*play*,*pause*,*stop*) na 3ª linha do *display* LCD.

Caso o veículo esteja parado e desligado e a execução das músicas permaneça durante um período prolongado, a bateria do automóvel irá descarregar até o momento que impossibilite a ignição do veículo. Como forma de evitar este descarregamento, que além de tudo, é extremamente prejudicial a vida útil da bateria, quando a tensão nominal

se aproximar de 11V, o PC receberá serialmente uma solicitação para que seja totalmente desligado, bem como o sinal remoto do módulo, liberando toda carga consumida também pelo sistema sonoro do veículo. Para evitar que o usuário religue o sistema, o *software* entrará em um loop infinito apresentando uma mensagem de alerta sobre a bateria no *display* LCD, até que a bateria seja novamente carregada, através da ignição do veículo.

Caso a tensão da bateria encontre-se numa faixa de tensão de 11V a 11,4V será somente emitido um alerta, sem acarretar no total desligamento do PC e do sistema sonoro.

Partindo-se do ponto de que a bateria está com carga e a reprodução de arquivos MP3 ocorre normalmente, o usuário pode desligar manualmente o PC a qualquer momento, pressionando novamente o botão "On-Off" do emissor infravermelho (figura 24). O PC irá receber serialmente a solicitação de desligamento e após ter sido totalmente desligado, será feito um teste sobre um sinal proveniente da alimentação 12V da fonte ATX do PC, que identifica se o PC está ou não ligado. Caso este seja "0", então o PC está realmente desligado e as informações gerais passarão novamente a ocupar tela cheia no *display* LCD, mantendo-se desta forma até que o PC seja ligado novamente. Os detalhes da especificação de cada funcionalidade da interface, são ilustrados nas figuras 21, 22 e 23, sendo auxiliado pelos demais processos auxiliares especificados.

Existem duas rotinas tratadas via interrupção, que possuem um identificador de qual interrupção deve ser tratada (figura 29) e as rotinas de interrupção propriamente ditas que são, a rotina de recepção serial (figura 30) e a rotina de recepção infravermelho (figuras 31 e 32) para os comandos Norma RC5.

Na especificação da figura 22, a mensagem "Alerta 1", corresponde á "Bateria descarregada... Recarregue...". Já a mensagem "Alerta 2", corresponde á "Recarregue a Bateria". O "Nível 1", significa que a bateria tem menos de 11V e o "Nível 2", significa que bateria está com a tensão entre 11V e 11,4V.

Na especificação da figura 23, a mensagem "Alerta 3", corresponde à "Reduza a velocidade". Na especificação não foi considerada a utilização do recurso *watchdog* do microcontrolador PIC16F877.

Figura 21: Especificação interface – I

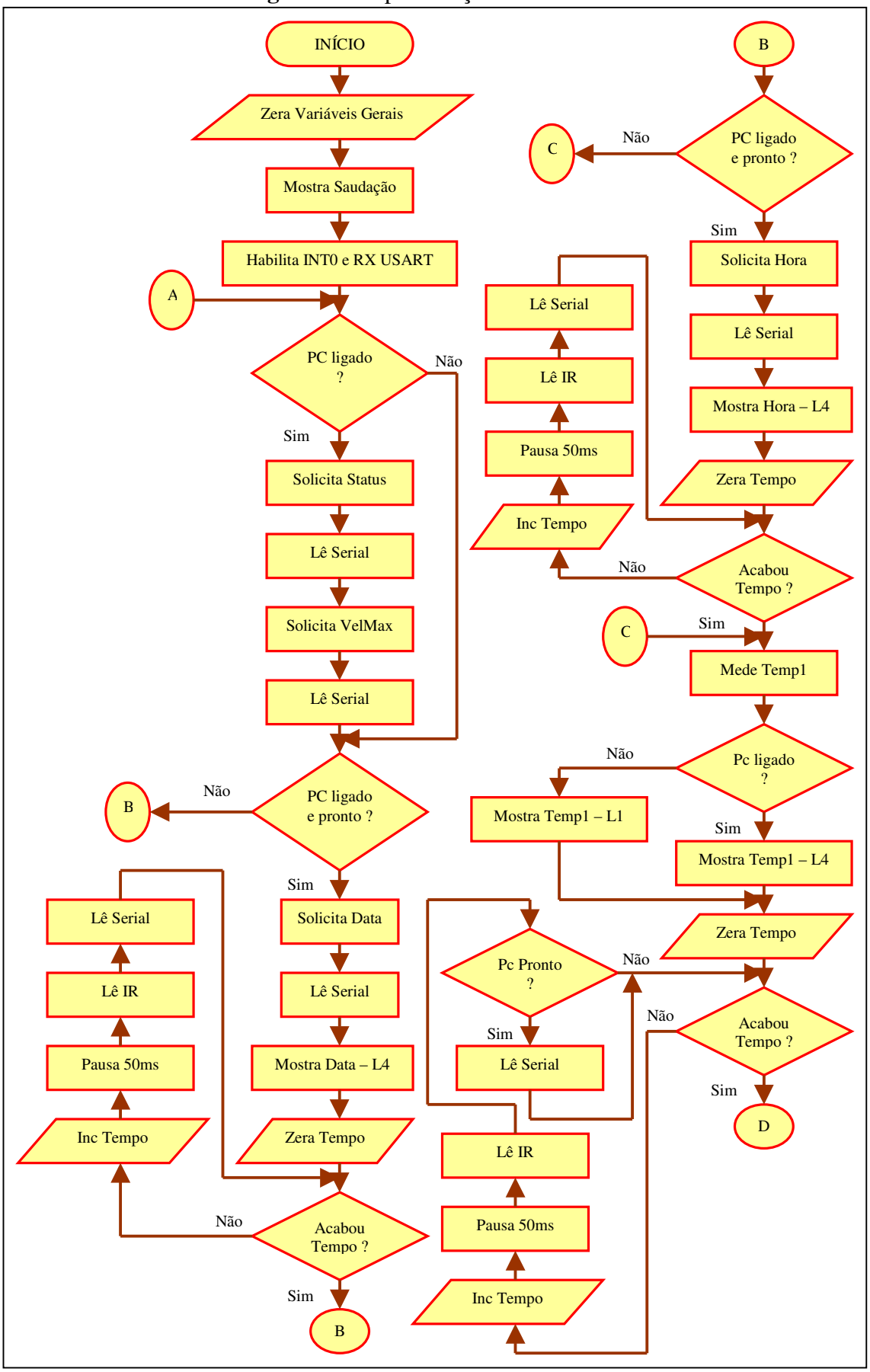


Figura 22: Especificação interface – II

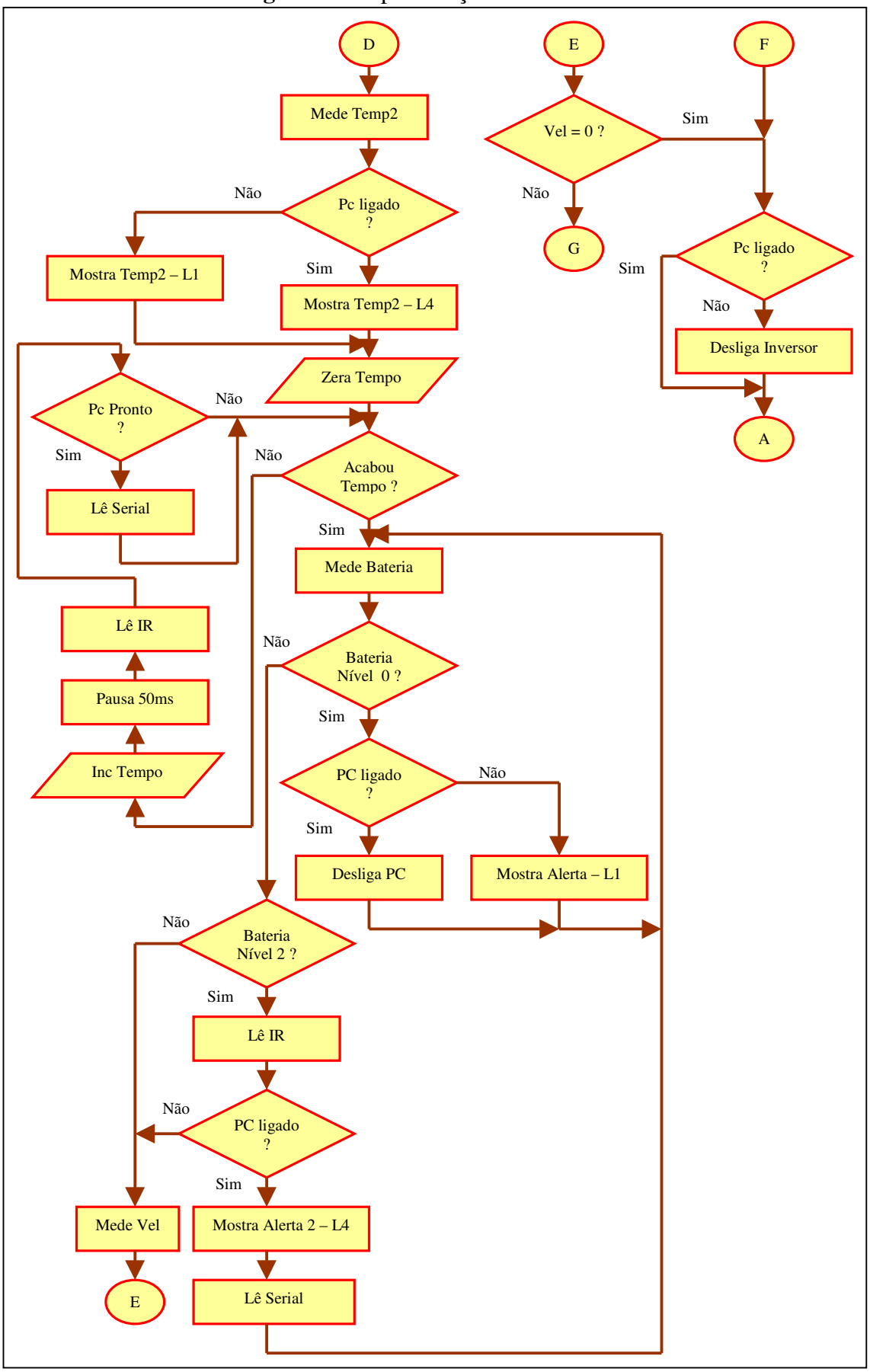
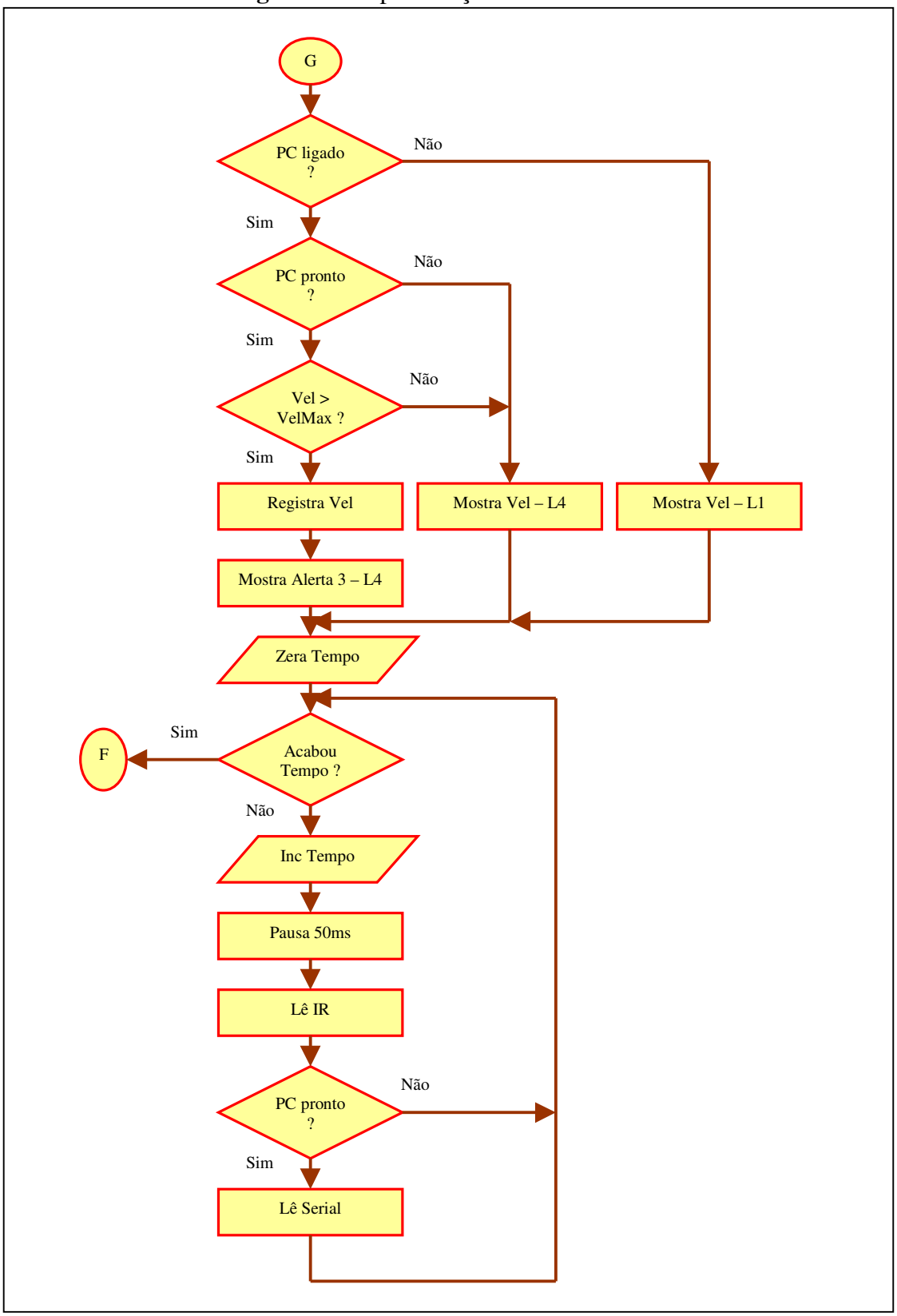


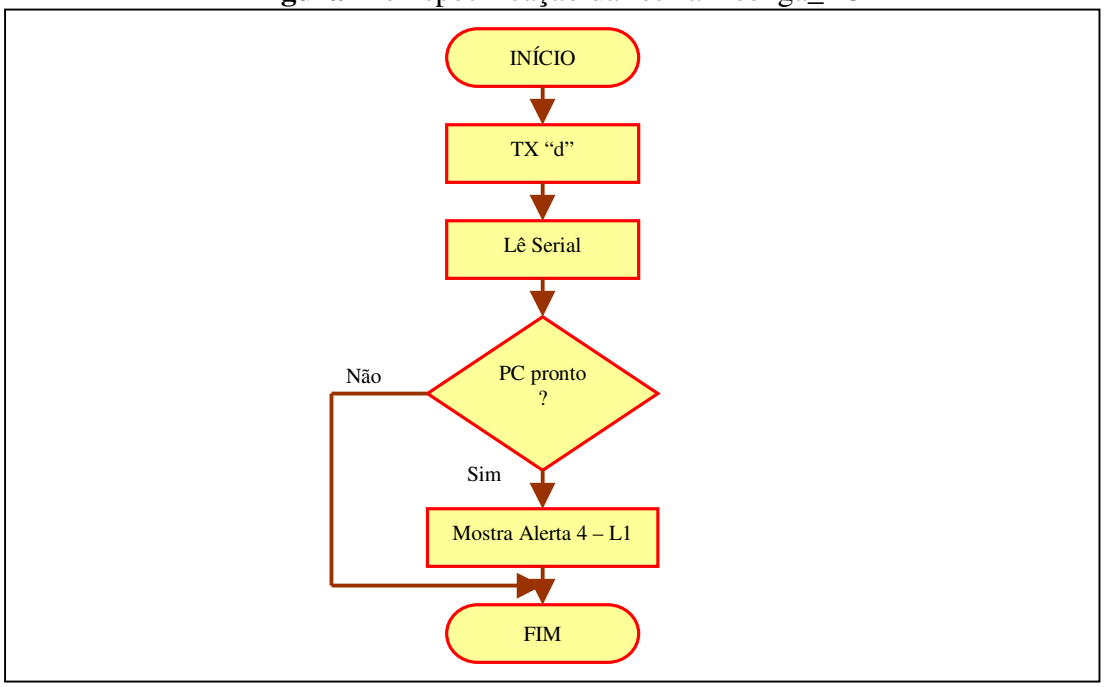
Figura 23: Especificação interface – III



7.2.2 Rotina Desliga_PC

A rotina Desliga_PC é responsável pelo processo de desligar o PC automaticamente, através de um procedimento padrão simulando uma ação manual para tal operação. Consiste basicamente em enviar um caracter de controle “d” para o PC, que ao receber tal caracter, retorna o mesmo caracter informando que será desligado. Neste momento o flag denominado pc_pronto fica falso e o PC desliga completamente, através de uma chamada de sistema do próprio PC. A figura 24 ilustra a especificação desta rotina. A mensagem “Alerta 4” representa “Aguarde... PC sendo desligado”.

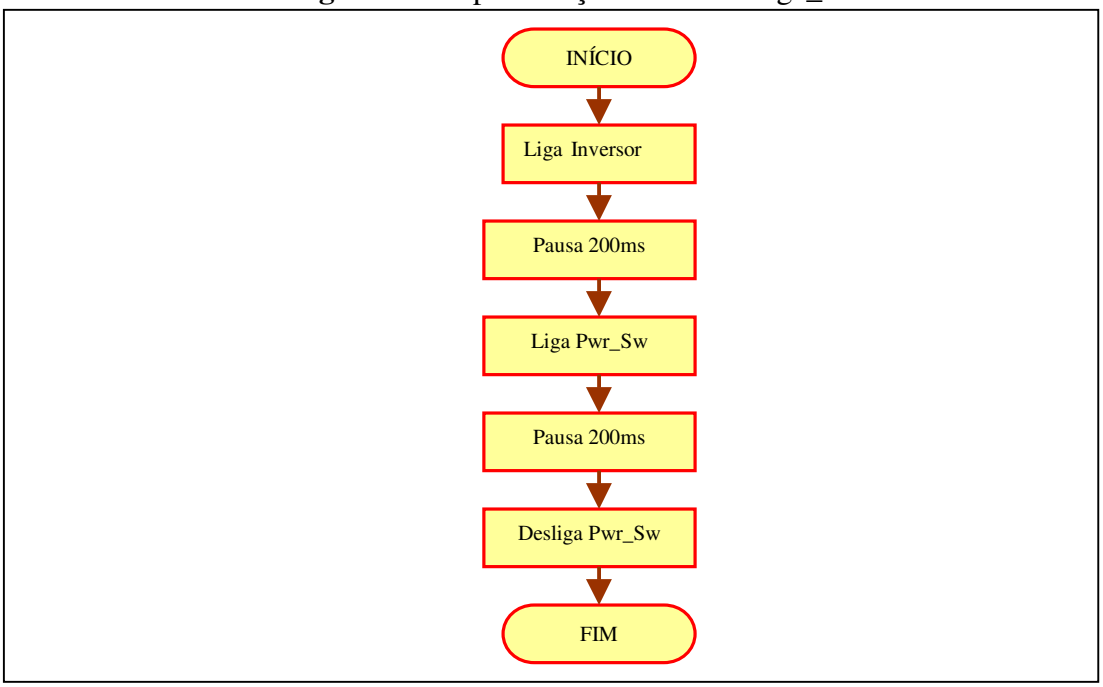
Figura 24: Especificação da rotina Desliga_PC



7.2.3 Rotina Liga_PC

A rotina Liga_PC tem como objetivo principal automatizar o processo de ligação do PC, substituindo a ação manual correspondente. Consiste basicamente em ligar o inversor de tensão, aguardar um tempo de 200ms para estabelecer a tensão na saída do mesmo e em seguida pulsar “Pwr_On” por 200ms para que a fonte ATX do PC forneça as tensões para o PC, que desta forma inicializa sua operação. A figura 25 ilustra a especificação desta rotina. O pulso em “Pwr_On” realiza a mesma ação manual de um usuário pressionando o botão “Liga/Desliga” de um gabinete ATX para ligar um PC qualquer.

Figura 25: Especificação da rotina Liga_PC



7.2.4 Rotina Atualiza_Serial

A rotina Atualiza_Serial é responsável por atualizar constantemente o fluxo de dados da interface, provenientes do PC. Este fluxo de dados compreende dados e *status* sobre as músicas MP3, data e hora atuais, e o valor máximo de velocidade veicular permitido sem a emissão de alertas. Estes dados são recepcionados através da interrupção serial, cuja especificação é ilustrada na figura 30, sendo armazenados em um *buffer*. Quando é chamada a rotina Atualiza_Serial, a mesma trata estes dados conforme o caracter inicial, que identifica o tipo de informação e executa diretamente a ação necessária para esta informação, apresentando os dados em uma posição específica do *display* LCD, ou então armazenando-as em variáveis, como é o caso de data e hora e também do valor de referência para a velocidade. O caracter "@" identifica o fim do pacote de caracteres para uma determinada informação.

Esta rotina torna o tratamento da interrupção serial, o mais ágil possível, evitando que o programa principal, fique durante muitos ciclos de *clock* dentro do tratador de interrupções. Conforme ilustra a especificação da interface, nas figuras 21,22 e 23, a rotina Atualiza_Serial é chamada constantemente apresentado os dados recebidos serialmente pelo PC, dados estes relacionados principalmente á função de MP3. As figuras 26 e 27 ilustram a especificação da rotina Atualiza_Serial.

Figura 26: Especificação da rotina Atualiza_Serial - I

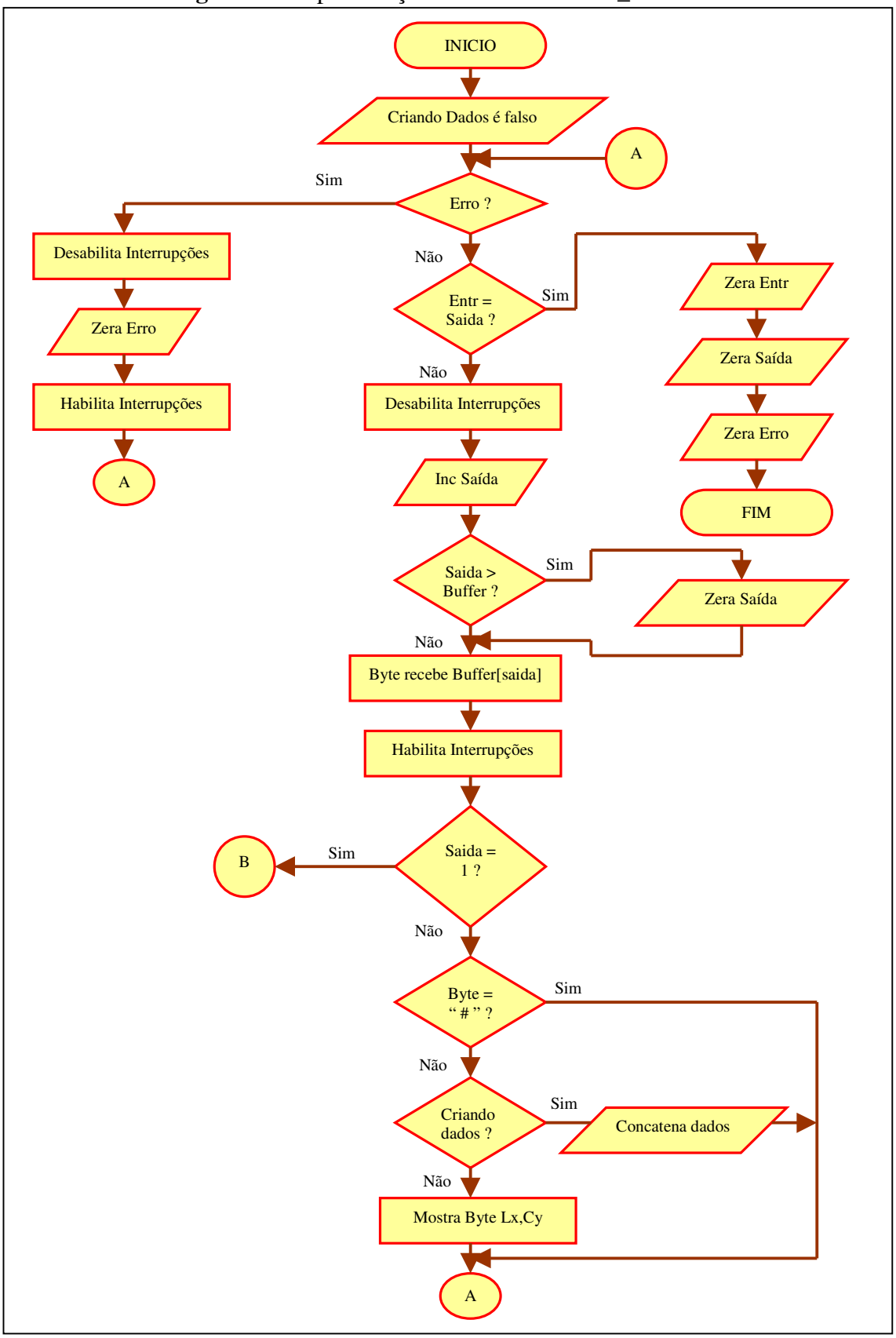
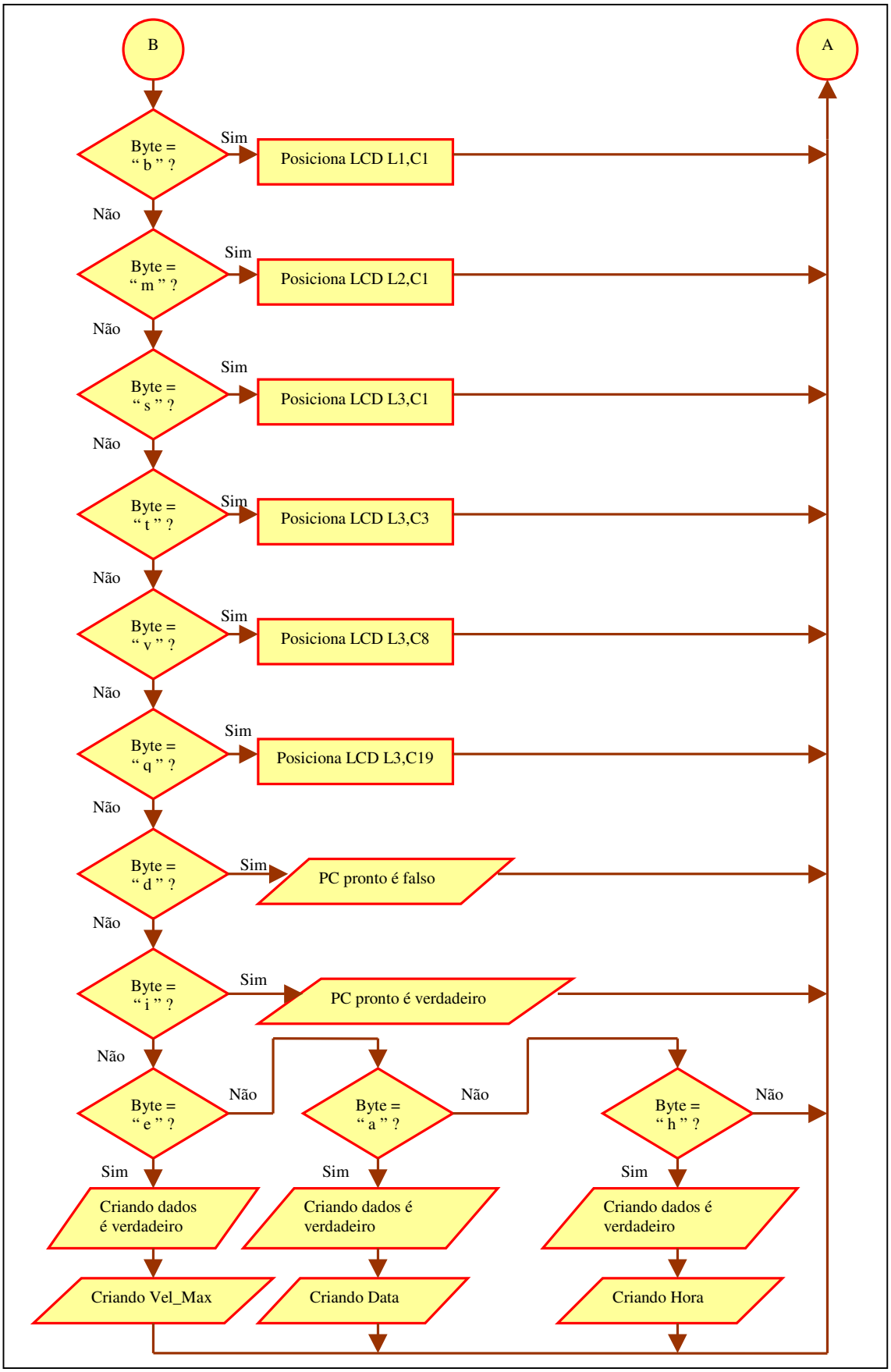


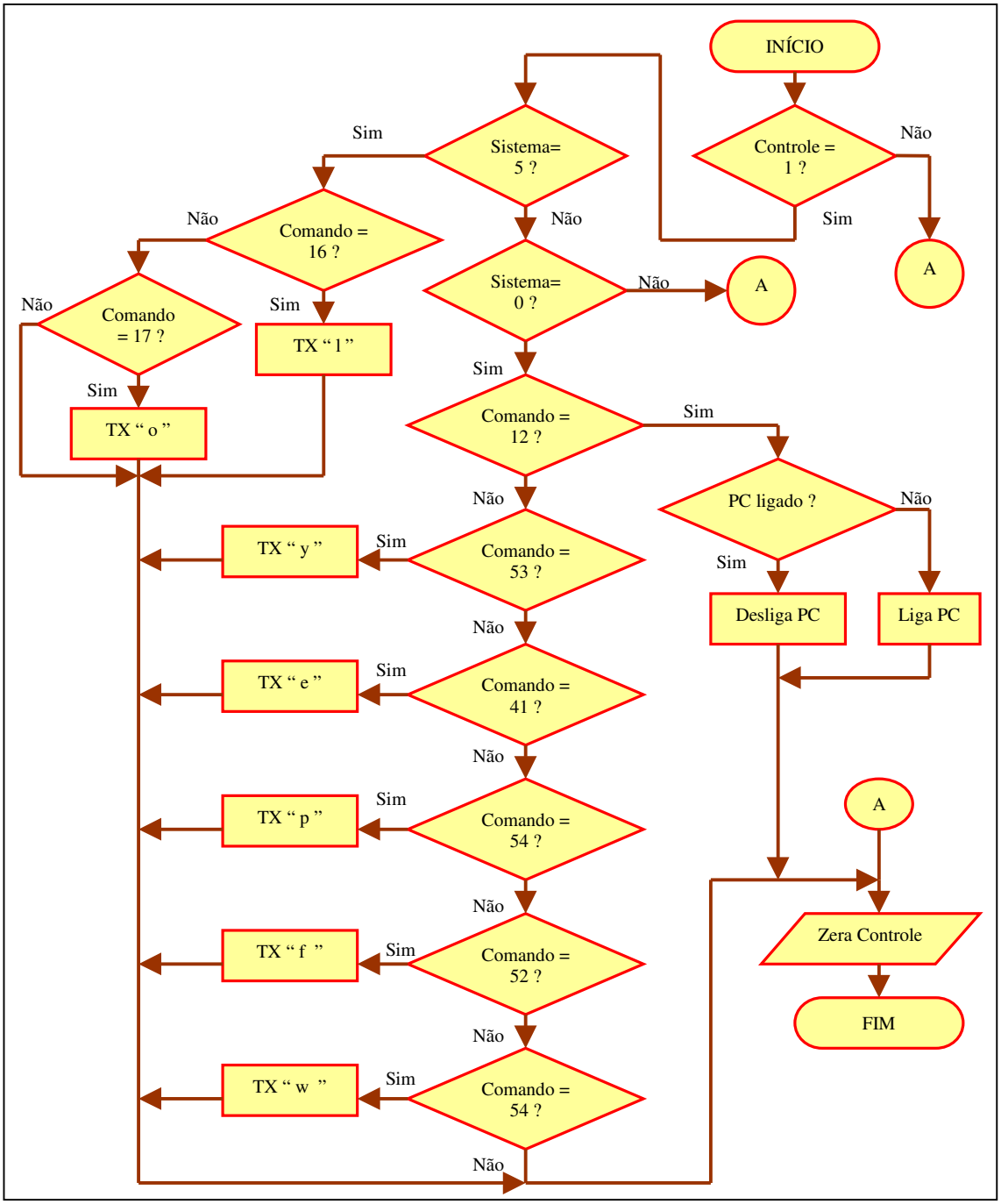
Figura 27: Especificação da rotina Atualiza_Serial – II



7.2.5 Rotina Atualiza_IR

A rotina Atualiza_IR segue a mesma lógica da rotina Atualiza_Serial, porém serve para executar ações provenientes do controle remoto. Estes dados são recepcionados pela interrupção INT0, que decodifica o pacote Norma RC5, conforme ilustra as figuras 31 e 32. Esta rotina é acessada constantemente e executa um dos comandos habilitados, agindo diretamente sobre o PC através da serial e comandos para ligar e desligar o mesmo. A figura 28 ilustra a especificação da rotina Atualiza_IR.

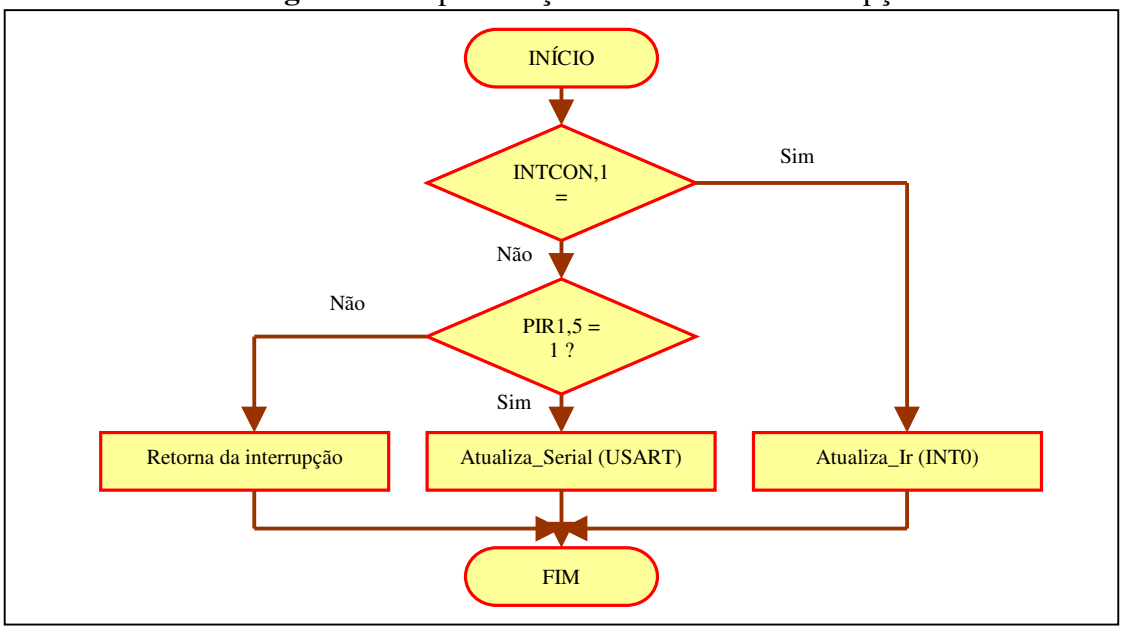
Figura 28: Especificação da rotina Atualiza_IR



7.2.6 Tratador de interrupções

O tratador de interrupções verifica qual foi a fonte da interrupção, dentre as duas opções possíveis, sendo elas a interrupção via INT0 para a recepção de comandos por infravermelho Norma RC5 (figuras 31 e 32), ou então via RX da USART para recepção serial de dados provenientes do PC (figura 30). É possível identificar a interrupção via INT0 através do bit 1 do registrador INTCON do PIC16F877. Se estiver setado, então a interrupção é via INT0. Se a interrupção ocorreu via RX, o bit 5 do registrador PIR1 do PIC16F877 deve estar setado. A figura 29 ilustra a especificação do tratador de interrupções.

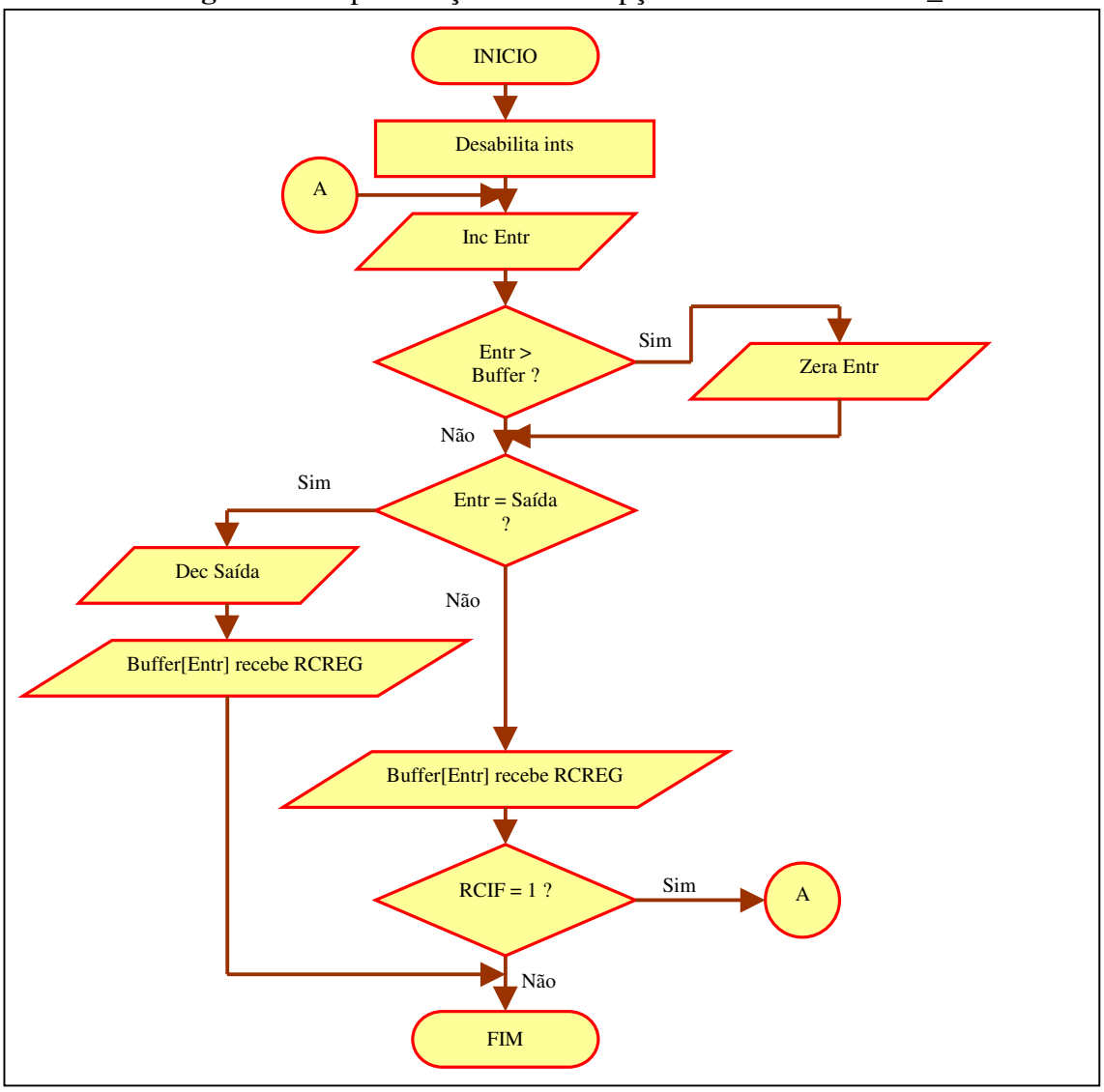
Figura 29: Especificação do tratador de interrupções



7.2.7 Interrupção via RX – Rotina Le_Serial

A interrupção via RX é responsável pela recepção de dados seriais provenientes do PC. Existe um *buffer* de 22 bytes que é preenchido conforme o tipo de informação recebida, visto que ao final de uma recepção serial, este *buffer* poderá ter 3 bytes no mínimo e 22 *bytes* no máximo (por exemplo, o nome de uma música ou de uma banda, mais o identificador inicial e o identificador final).

Figura 30: Especificação da interrupção via RX – Rotina Le_Serial



7.2.8 Interrupção via INT0 – Rotina Le_IR

A interrupção via INT0, tem como objetivo detectar e decodificar um pacote de bits provenientes do controle remoto, separando-o em informações de controle, comando e sistema pela Norma RC5. As figuras 31 e 32 representam a especificação desta interrupção. Inicialmente o pacote é recepcionado e armazenado em um vetor de bits chamado VetorBits.

Figura 31: Especificação da interrupção via INT0 – Rotina Le_IR – I

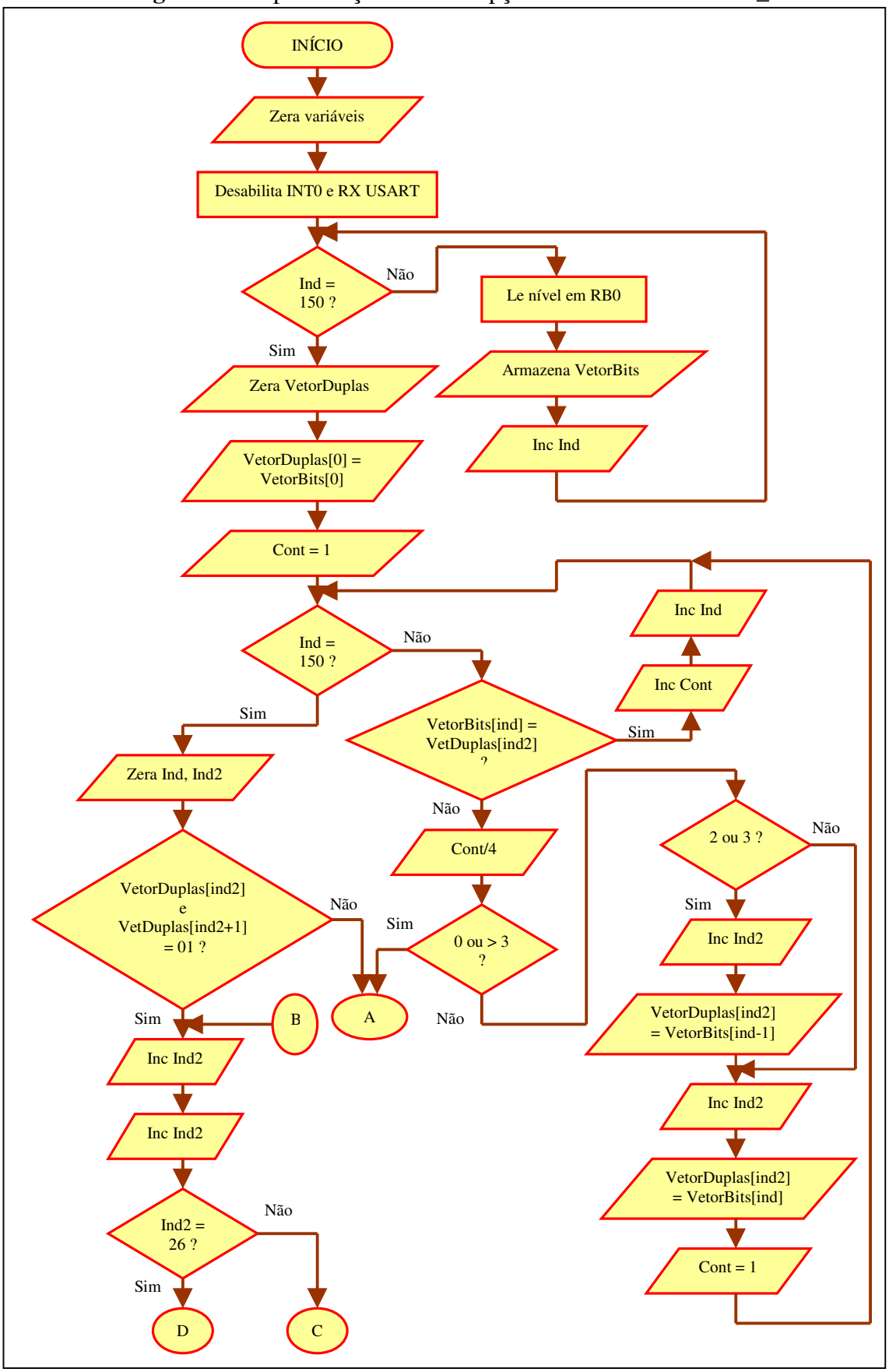
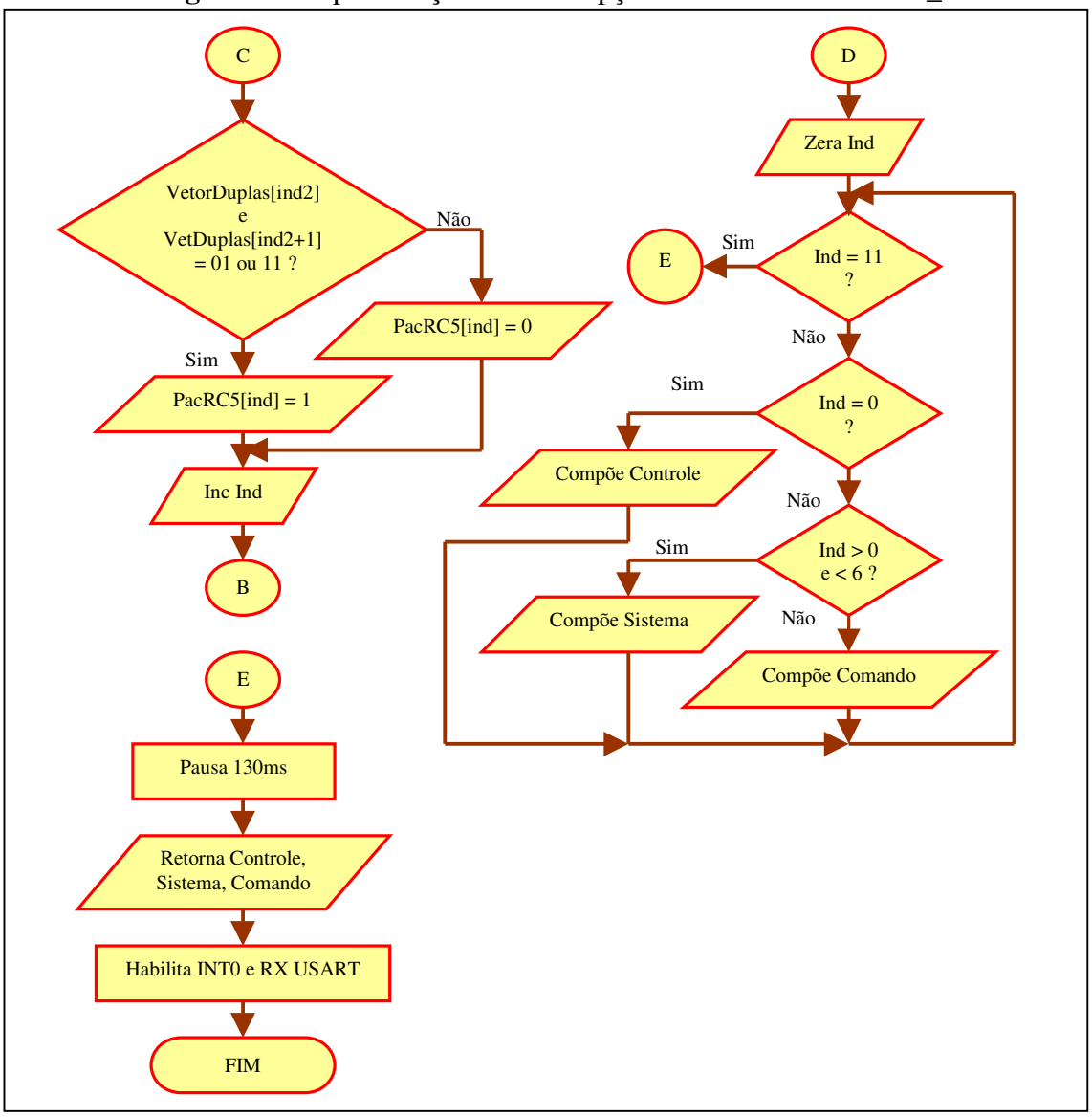


Figura 32: Especificação da interrupção via INT0 – Rotina Le_IR - II



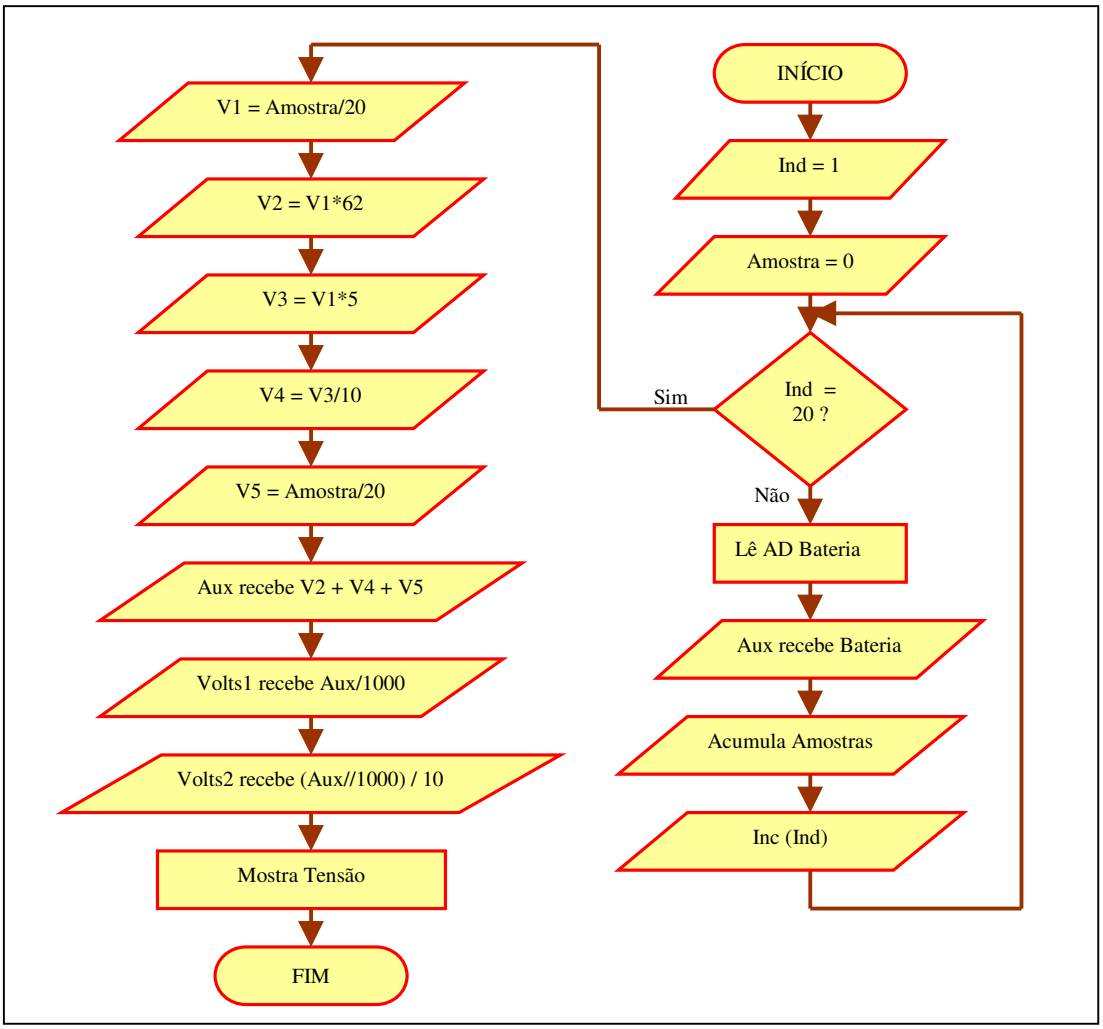
O VetorBits, registra um intervalo de níveis lógicos 0 e 1 que identifica exatamente o pacote recepcionado. Através destes intervalos é possível identificar uma seqüência dupla de bits, que irão formar cada bit dos 14 bits constituintes do pacote RC5. Cada dupla de bits é armazenada em outro vetor chamado VetorDuplas através de uma varredura completa sobre VetorBits.

Com o VetorDuplas preenchido, o pacote RC5 de 14 bits é criado, ou seja, VetorDuplas possui 28 bits, sendo que duplas de bits 01 e 11 correspondem a um nível lógico 1, enquanto que duplas de bits 00 e 10 correspondem a um nível lógico 0. Desta forma VetorDuplas é percorrido até o último bit formando o pacote de bits Norma RC5. Para finalizar, este pacote é separado em dados de controle, sistema e comando.

7.2.9 Leitura de tensão da bateria

A leitura de tensão da bateria é possível através da leitura de um dos 8 canais A/D do PIC16F877. Como apresentado mais adiante na seção de integração de *hardware*, a tensão da bateria, não é aplicada diretamente ao A/D, mas passa por um divisor de tensão para torná-lo equivalente aos sensores de temperatura. A tensão da bateria obtida é um valor referente a média de 20 leituras do canal A/D em questão. Este valor passa por um processo de conversão através de uma constante, conhecida de valor 62. As operações matemáticas para a descoberta do valor de tensão ocorrem numa escala de multiplicação de 1000, devido ao fato do compilador PicBasic não trabalhar com ponto flutuante. Após o divisão por 1000 do somatório de V2, V4 e V5 é obtida a parte inteira da tensão e com o resto da divisão por 1000 deste somatório é obtida a parte fracionária. Ao final ocorre a leitura de um valor de tensão no formato nn,nnV. A figura 33 mostra a especificação para este processo.

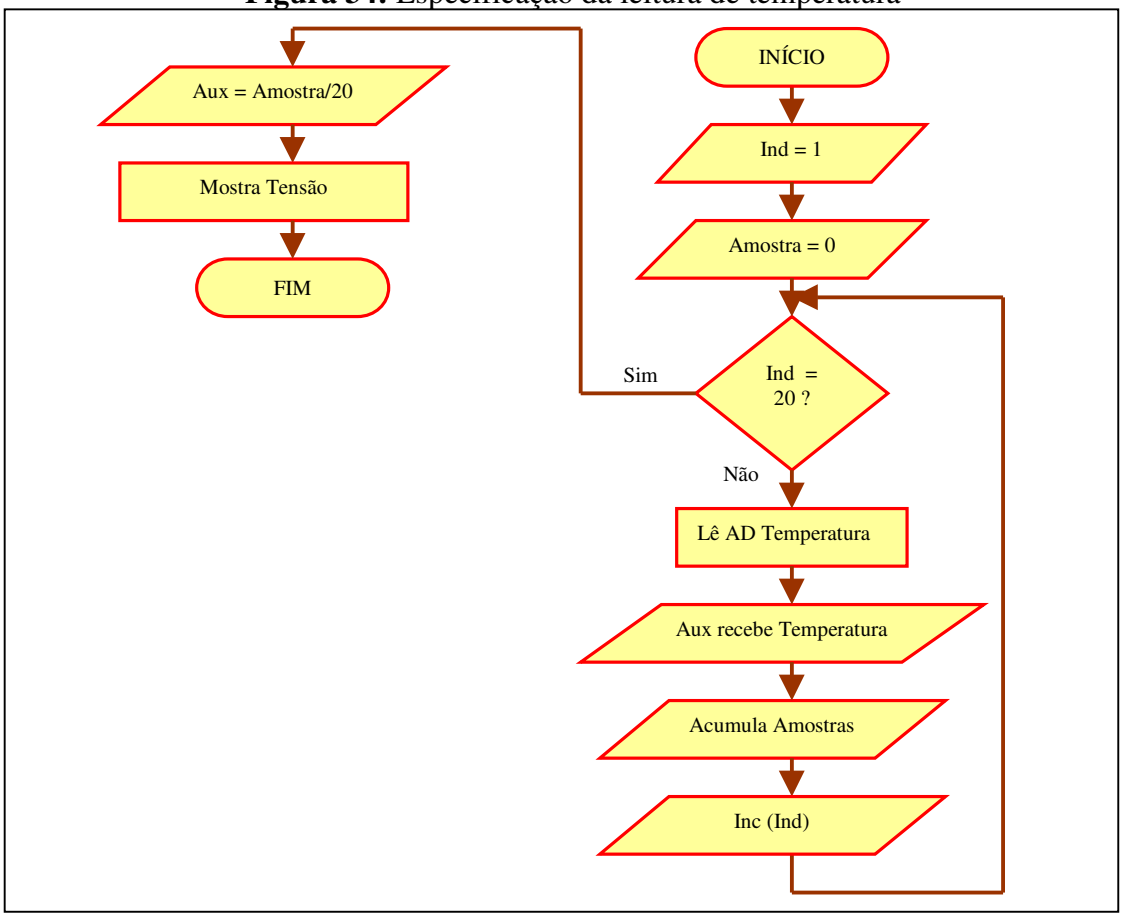
Figura 33: Especificação da leitura de tensão da bateria



7.2.10 Leitura da temperatura

O processo de leitura da temperatura, segue a mesma lógica da leitura de tensão da bateria, porém não existe a necessidade da conversão do valor final obtido pela média de 20 leituras nos canais de A/D, pois este valor final representa o próprio valor de temperatura, pois a tensão de referência dos A/Ds é justamente a máxima tensão fornecida na saída dos sensores de temperatura LM35 utilizados. A figura 34 demonstra a especificação da leitura de temperatura. Os valores de temperatura obtidos são expressos em °C.

Figura 34: Especificação da leitura de temperatura

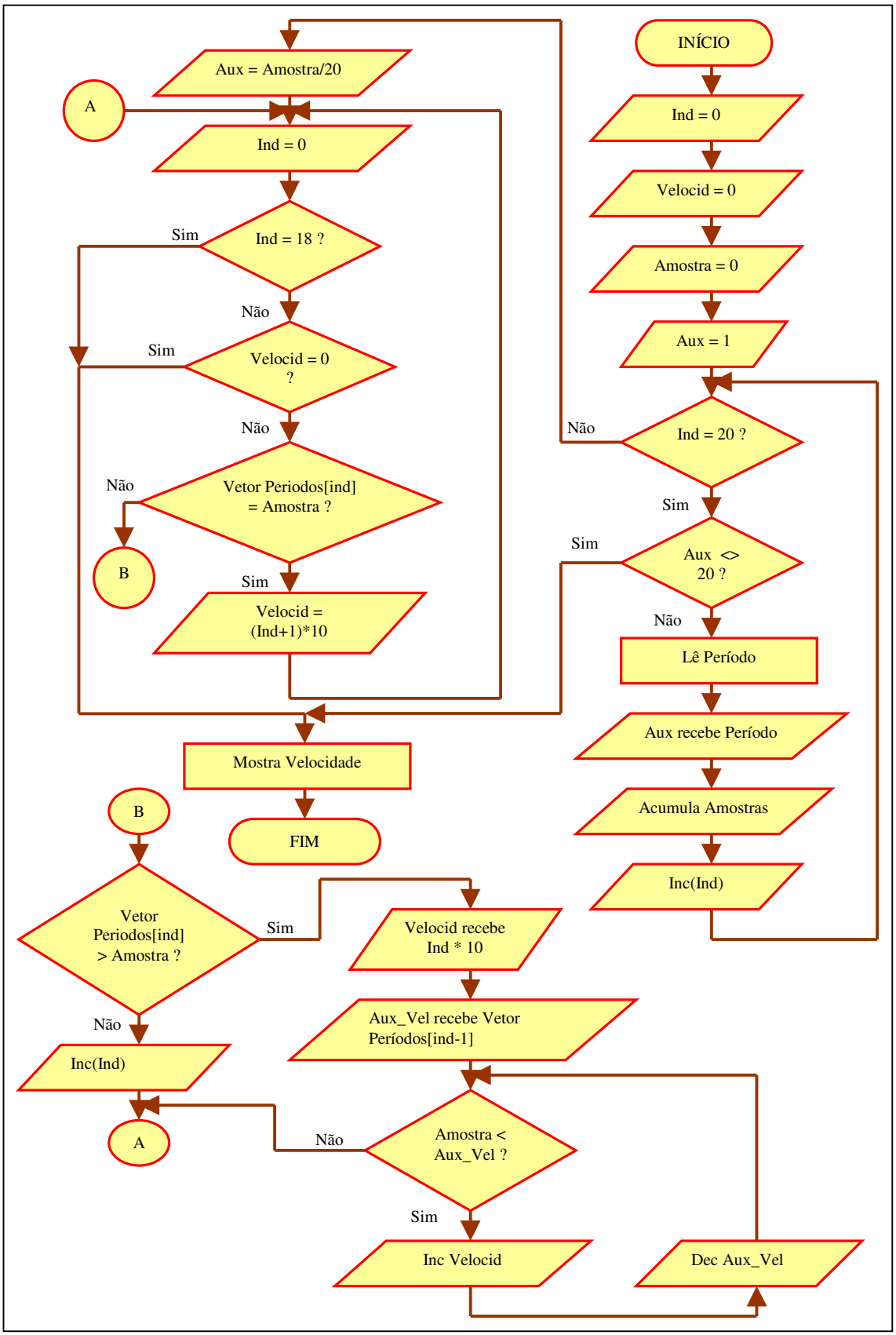


7.2.11 Leitura de velocidade

A leitura de velocidade é possível através do período dos pulsos gerados pelo sensor de efeito Hall V305. O valor médio do somatório de 20 períodos é comparado com um vetor de períodos denominado, VetorPeriodos, com 18 valores de referência até que se encontre o respectivo valor de velocidade entre 1 e 180 Km/h.

As figuras 35 ilustra a especificação da leitura de velocidade.

Figura 35: Especificação da leitura de velocidade



7.3 Especificação do software do PC

As principais funções do *software* do PC são :

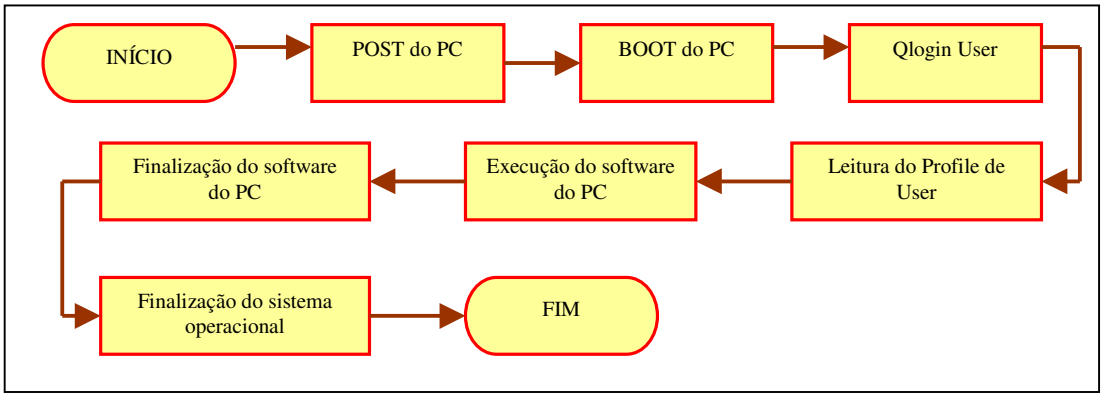
- Execução de arquivos MP3 através de comunicação inter-processos, redirecionando as suas respectivas entrada e saída padrões para o programa principal. O programa utilizado foi o "mpg321" que foi desenvolvido com a biblioteca MAD (*Mpeg Audio Decoder*), que produz uma saída decodificada de 24bits e é totalmente *freeware*;
- Controle de volume através da criação de outro processo executando o programa "aumix", redirecionando a saída padrão para o programa principal;
- Obtenção de dados ID3 dos arquivos MP3 através de um processo executando o programa mp3info, redirecionando a saída padrão para o programa principal;
- Registro de velocidades superiores a pré-configurada no arquivo "*velocmax.cfg*", criando um *log* de velocidades no arquivo "*logvel.log*", recebendo serialmente esta informações, através da interface;
- Desligamento do PC através de instrução recebida serialmente, podendo ser por solicitação via emissor infravermelho ou por problemas de carga na bateria;
- Fornecimento de data e hora para a interface via serial;
- Comunicação serial com a interface, com recepção e transmissão de dados, sem interrupção.

A especificação do *software* do PC é ilustrado nas figuras 37, 38 e 39.

Na figura 36 é ilustrado todo o processo necessário até o início da execução do *software* do PC. O POST (*Power-On Self-Test*) do PC, é executado toda vez que o PC é ligado ou quando ocorre um *reset*. O BOOT do PC consiste em iniciar o carregamento do sistema operacional Linux Red Hat 7.3.

A primeira tarefa importante desempenhada encontra-se no login. O programa *qlogin*, já visto anteriormente, irá automatizar o *login*, abstraindo a tela de *login*, onde seria inserido o nome de usuário e senha. Estas informações estarão previamente armazenadas em *qlogin*, que internamente validará a senha e usuário, e apontará diretamente para o *software* do PC, que se comunicará com a interface.

Figura 36: Processo de execuções até o *software* do PC



7.3.1 Programa principal

Quando o *software* de PC iniciar, irá fazer uma varredura no diretório "*/Tcc/MP3*" criando um arquivo de texto chamado "*Mp3.lst*" no diretório "*/Tcc*". Este arquivo servirá de base para a criação de um vetor contendo as músicas disponíveis e também como fonte de informações para o *software* de transferência de arquivos MP3 via FTP. O vetor criado será usado para executar as músicas MP3 mais adiante. Em seguida o arquivo "*logvel.log*" presente no diretório "*/Tcc*" é aberto para escrita e passará a registrar todos os excessos de velocidade detectados pela interface microcontrolada.

O *software* do PC irá criar um novo processo para executar o programa "mpg321" para execução dos arquivos MP3, redirecionando as suas respectivas entrada e saída padrões para o programa principal. O processo executa "mpg321 tcc –R" que abre um terminal remoto para o *player* de MP3, e que já possui internamente diversas funcionalidades, tais como *play*, *pause*, *stop*, carregar música, abandonar programa.

Desta forma o programa principal (figuras 37, 38 e 39) apenas passa mensagens para este processo, como se as mesmas estivessem sendo inseridas via teclado. A saída fornecerá status das músicas MP3, bem como, o tempo decorrido da mesma. Caso ocorra um erro ao criar este processo, um erro de criação do processo mpg321 é atribuído a variável "erro" e o programa finaliza, desligando o PC.

Se o processo com "mpg321" (figura 40) for criado, então o próximo passo será configurar e abrir a porta serial, responsável principal pela comunicação do *software* do PC com o *software* da interface microcontrolada. Caso ocorra um erro ao abrir a porta serial, um erro de abertura da porta serial será atribuído a variável "erro" e o programa finaliza, desligando o PC.

Após a abertura da porta serial, a mesma estará pronta para receber e enviar dados para a interface e ocorrerá a leitura de um valor numérico contido no arquivo "*velocmax.cfg*" no diretório "*/Tcc*" que servirá de referência para a detecção de excessos de velocidade pela interface microcontrolada, valor este transmitido serialmente para a mesma, logo após a sua devida solicitação pela interface. O mesmo procedimento ocorre com um dado de controle enviado para a interface, que indica que o PC está pronto para registrar velocidades e executar as músicas MP3 e fornecer informações de data e hora. Caso a solicitação de envio do dado de controle e do valor de referência de velocidade, demorem a ser transmitidos pela interface, então um erro de *timeout* será atribuído a variável "erro", e o programa irá finalizar, desligando o PC.

A partir deste ponto o programa entra em *loop* infinito, efetuando leituras na porta serial, analisando os dados recebidos e executando ações relacionadas a estes dados. Estas ações podem ser comandos de *playback* sobre os arquivos MP3, tais como *play*, *pause*, *stop*, *next*, *previous*, controle de volume, fornecimento de data e hora ou registro de excessos de velocidade de forma cíclica, assim que solicitado pela interface. Em princípio, ficará a espera de um comando *play* originado pelo emissor de infravermelho, para começar a reprodução de músicas MP3. Neste momento o sinal de remoto do módulo de potência será ativado na interface microcontrolada.

Ao iniciar a execução da música MP3, o *software* do PC envia serialmente para a interface as informações ID3 *Tag* e status durante a execução da mesma. Quando terminar a música o índice do vetor será incrementado, apontando e carregando a próxima música, seguindo as mesmas ações e procedimentos, até que o usuário resolva desligar o sistema ou o mesmo seja desligado automaticamente devido a problemas com a bateria. Durante a execução das músicas, o PC continuará fornecendo novas informações de data e hora, assim que solicitado.

Caso necessário aumentar ou diminuir o volume, o programa principal criará um novo processo executando "aumix" (figura 41) para alterar o volume para mais ou menos e finalizando-o logo após a alteração, tendo sua saída padrão redirecionada ao programa principal. Para especificação do *software* do PC foram usados fluxogramas.

Figura 37: Especificação do programa principal do PC – I

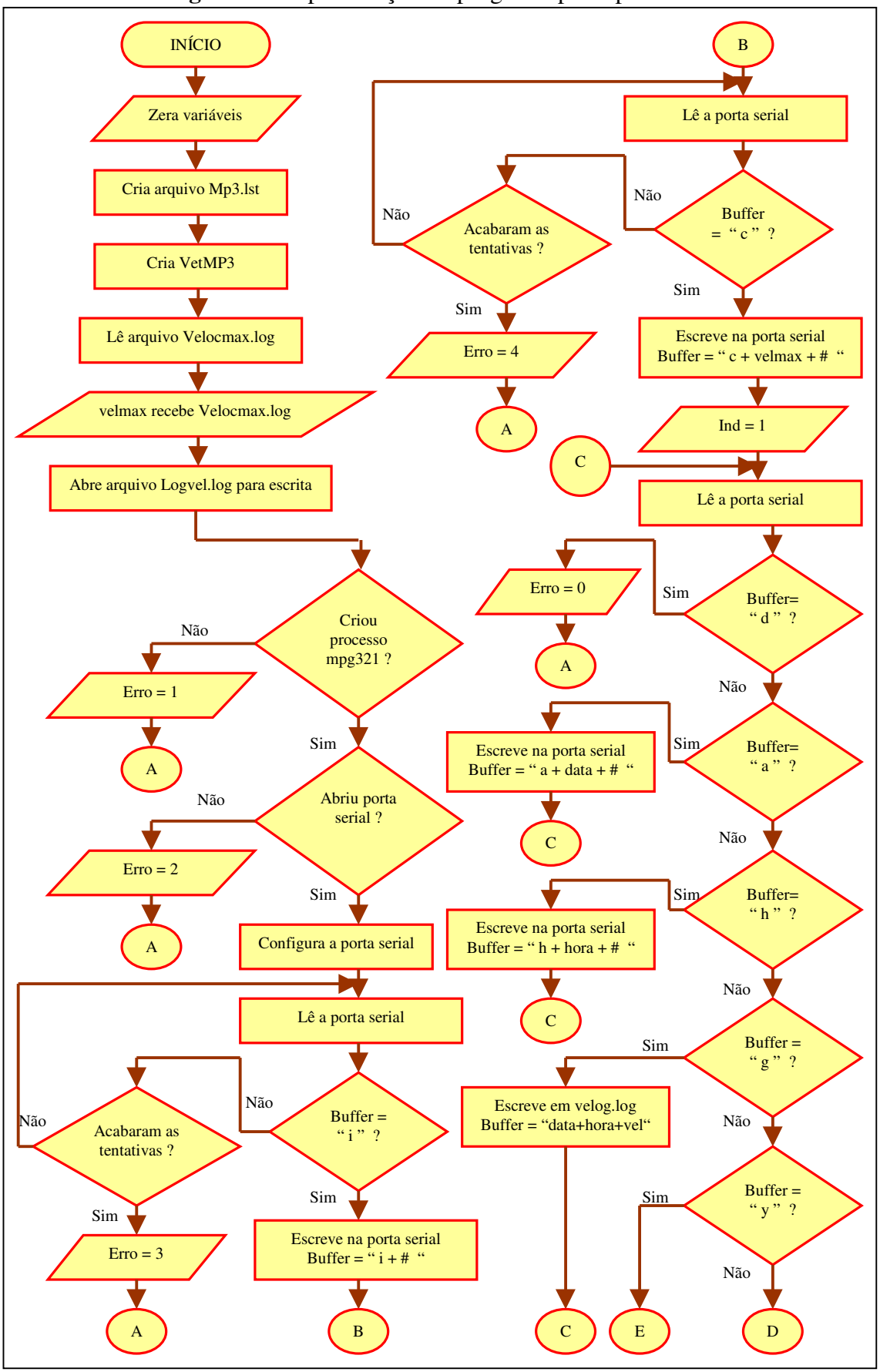


Figura 38: Especificação do programa principal do PC – II

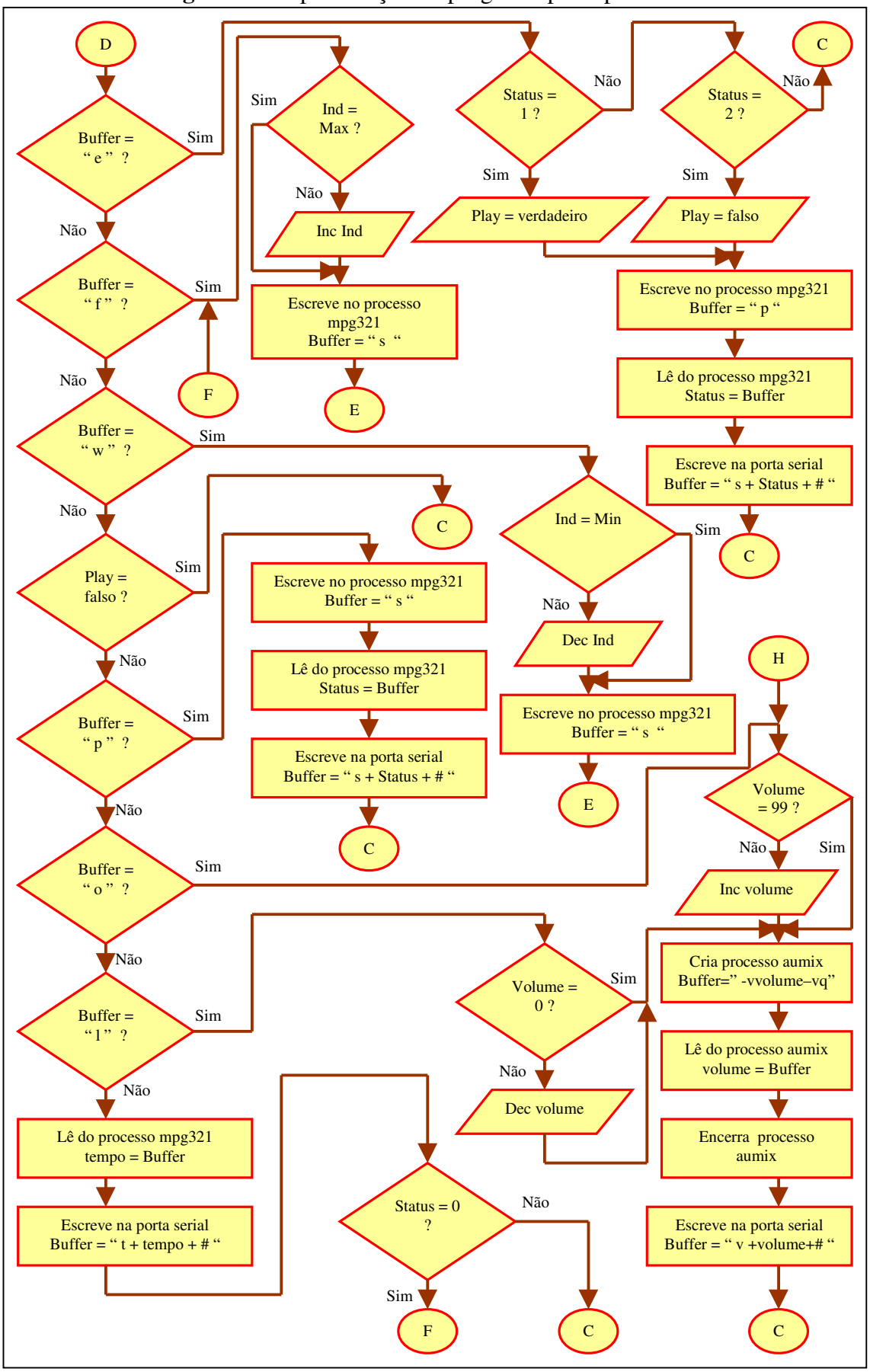
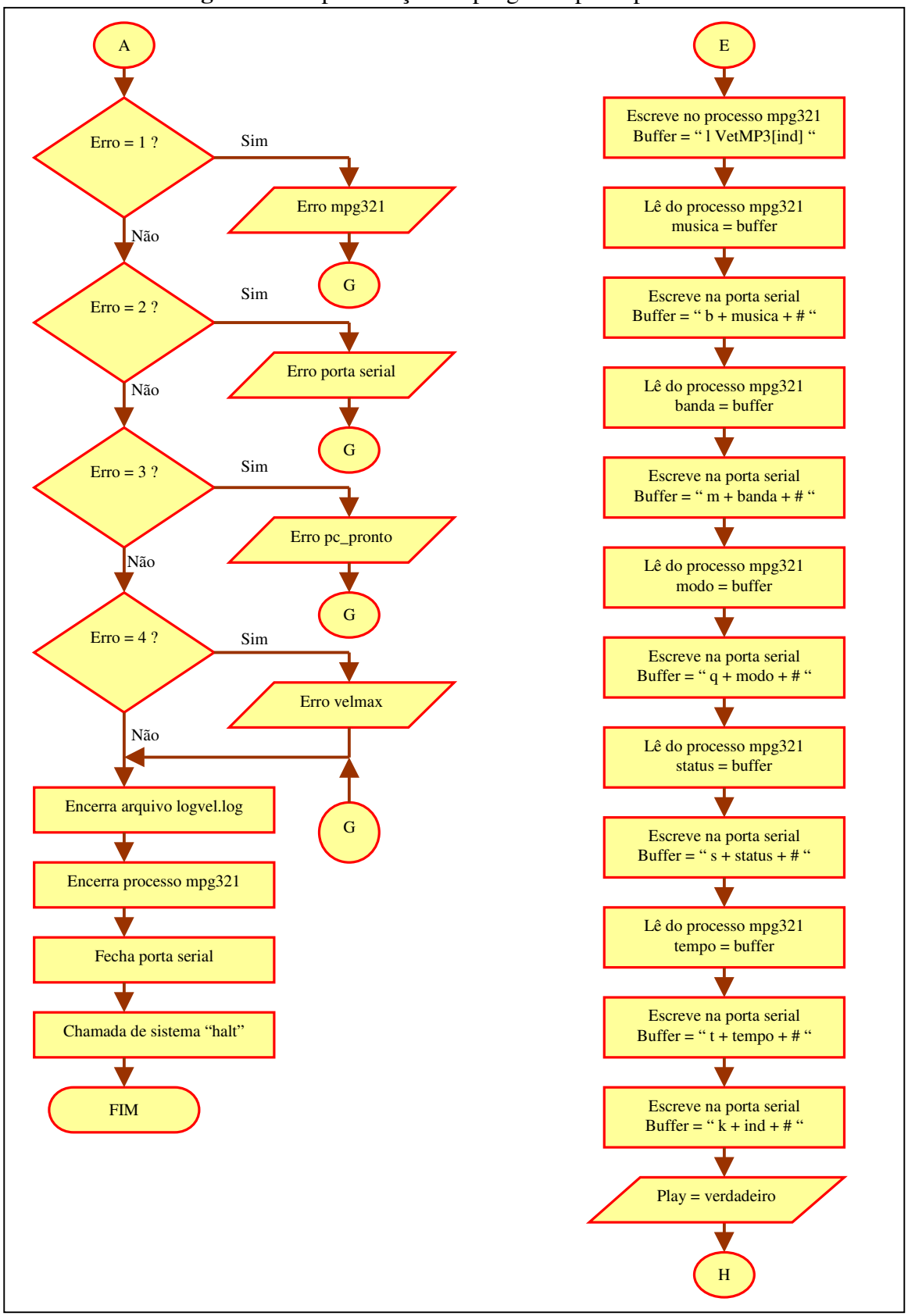


Figura 39: Especificação do programa principal do PC - III

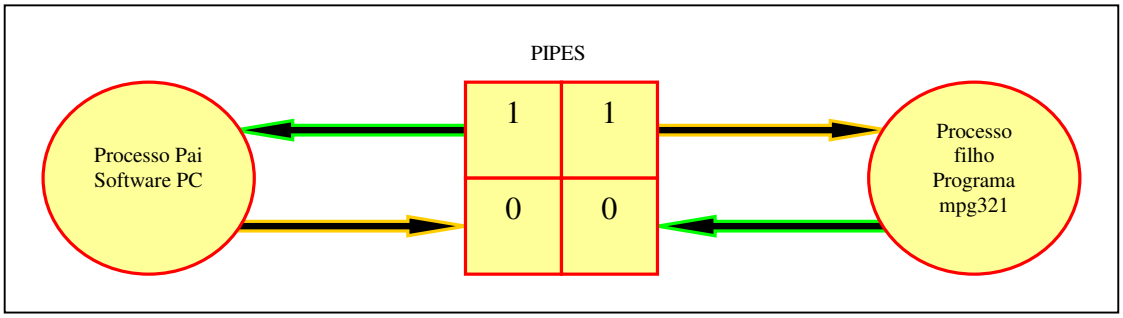


7.3.2 Processo mpg321

O processo mpg321, como o nome sugere, caracteriza a criação de um processo filho dentro do programa principal, responsável por executar o programa mpg321, para a execução de arquivos MP3. Este processo filho é comandado diretamente pelo programa principal, através de um descritor de arquivos duplo de *pipes*, redirecionando a sua entrada e saída padrão para o mesmo, para que o processo pai (programa principal), possa comandá-lo.

Para que isto seja possível, é necessário fechar a entrada e saída padrão do processo filho mpg321, duplicar o *link* de ligação do descritor de arquivos dos *pipes*, e após isso, fechar o *link* original do descritor de arquivos dos *pipes*, tornando-os a entrada e saída padrão do processo mpg321. Após estas operações, todo fluxo de dados do processo filho, poderá ser lido no processo pai, através de um acesso direto a este descritor de arquivos dos *pipes*. Na figura 40 é apresentada a representação final do processo filho mpg321 interligado com o descritor de arquivos de pipes.

Figura 40: Comunicação – Processo pai/Processo filho mpg321



O fluxo de comunicação entre os dois processos ocorre da seguinte maneira :

1) O processo pai escreve no descritor 0 uma mensagem para o processo filho, sendo esta mensagem um comando válido para o programa mpg321.

2) O processo filho lê esta mensagem no descritor 0, interpreta e executa o comando.

3) Ao invés de apresentar a saída na tela, que originalmente era a saída padrão, o processo filho escreve a saída no descritor 0 do *pipe*.

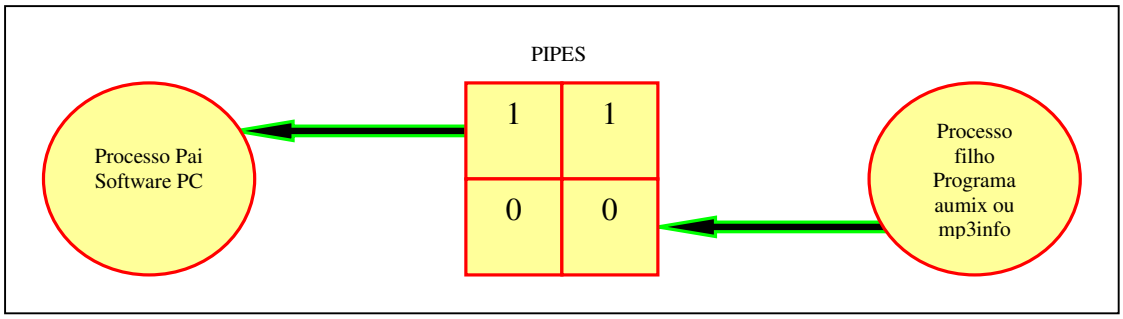
4) O processo pai, recupera a saída gerada pelo processo filho, através de uma leitura no descritor 1 do *pipe*. Neste momento o processo pai pode selecionar as informações relevantes, tais como nome da música e da banda, status, etc...

7.3.3 Processo aumix e mp3info

O processo aumix, é responsável pela criação de um processo filho para o programa aumix, responsável pelo comando de volume das músicas MP3. O principio de funcionamento deste processo é idêntico ao do processo gerado para o programa mpg321, porém neste caso, apenas a saída padrão do programa aumix é redirecionada para o descritor de arquivos *pipe*. O mesmo ocorre com o programa mp3info, que é responsável pela leitura de informações ID3 das músicas MP3.

O programa aumix e o programa mp3info, podem ser executados interativamente, ou então através de comandos diretos com ou sem retorno dos parâmetros atualizados. Em aplicação, ocorre um comando direto sobre aumix, retornando originalmente na tela o parâmetro "volume" atualizado. Já no caso do programa mp3info, ocorre um comando direto também que irá gerar uma determinada saída com informações ID3. Foram alteradas as saídas padrão de aumix e mp3info, para que o processo pai (programa principal), pudesse ler o valor de volume atualizado ou informação ID3 para apresentá-lo no *display* LCD da interface microcontrolada. A figura 41 ilustra a representação da comunicação entre o processo pai e o processo filho aumix ou mp3info.

Figura 41: Comunicação – Processo pai/Processo filho aumix e mp3info



Um diferença significativa entre o processo filho aumix/mp3info e o processo filho mpg321, é o tempo de duração dos mesmos. Enquanto o processo mpg321 fica aberto durante praticamente toda execução do programa principal, o processo aumix/mp3info, é criado dinamicamente, por n vezes durante a execução do programa principal, executando um alteração no parâmetro "volume" ou apresentação de informação ID3, escrevendo o novo valor do parâmetro no descritor 0 dos pipes e encerrando logo após. No caso de aumix este processo se repete em três situações

distintas: no início da reprodução de uma música de MP3, em um comando para aumentar o volume, ou em um comando para reduzir o volume. Estas três situações executam sempre a mesma rotina do processo aumix, que está centralizada no programa principal para ser compartilhada pelas mesmas. No caso do processo do programa mp3info, ele é executado três vezes no início da execução da música MP3, retornando o nome e artista da música e a qualidade da música (*Stereo*/*Mono*).

7.4 Implementação do software da interface

O *software* da interface foi desenvolvido com a linguagem Basic utilizando o compilador PicBasic Pro, já explanado anteriormente, com código também na área de tratamento de interrupção para as recepções seriais e de sinais infravermelhos (provenientes do emissor infravermelho padrão RC5).

A utilização do compilador PicBasic, representou um grande diferencial no desenvolvimento do sistema, visto que o mesmo, abstrai o excesso de código gerado pela linguagem Assembly, o que torna o programa mais legível e mais estruturado. A seguir são apresentados detalhes de implementação de cada funcionalidade desempenhada pela interface microcontrolada, sendo elas:

- Interface;
- Rotina Desliga_PC;
- Rotina Liga_PC;
- Rotina Atualiza_Serial;
- Rotina Atualiza_IR;
- Tratador de interrupções;
- Interrupção via RX – Rotina Le_Serial;
- Interrupção via INT0 – Rotina Le_IR;
- Rotina de leitura da tensão da bateria;
- Rotina de leitura de temperatura;
- Rotina de leitura de velocidade.

7.4.1 Interface

Consiste em realizar seqüencialmente todas as funcionalidades da interface, temporizando-as, e chamando rotinas auxiliares, tais como Le_IR, Le_Serial, Liga_PC, Desliga_PC, que completarão o funcionamento da mesma.

Inicialmente é apresentada a declaração de variáveis e configurações necessárias para os componentes auxiliares da interface, tais como *display* LCD, receptor IR, conversor RS232, ADs internos, entre outros. O quadro 11 apresenta a configuração para o *display* LCD.

Quadro 11: Definições do *display* LCD - Interface

```
' Configurações para o Display LCD
DEFINE  LCD_DREG        PORTB 'Porta para o LCD
DEFINE  LCD_DBIT        4     'Data Bit para modo 4 bits
DEFINE  LCD_RSREG       PORTC 'Porta usada por RS do LCD
DEFINE  LCD_RSBIT       4     'Pino usado por RS do LCD
DEFINE  LCD_EREG        PORTC 'Porta usada por Enable do LCD
DEFINE  LCD_EBIT        3     'Pino usado por Enable do LCD
DEFINE  LCD_BITS        4     'Define modo 4 bits
DEFINE  LCD_LINES       4     'Define 4 linhas para o LCD
DEFINE  LCD_COMMANDUS   2000  'Delay time dos comandos do LCD
DEFINE  LCD_DATAUS      50    'Delay time dos dados do LCD
```

A utilização da porta serial interna do PIC16F877 ocorreu de forma análoga para recepção e transmissão. Para recepção foi utilizado o comando *Hserin* e para transmissão de dados, foi utilizado comando o Hserout. Ambas configurações são indicadas no quadro 12.

Quadro 12: Definições da porta serial - Interface

```
' Configurações para a porta serial usados por TX via hserout
DEFINE HSER_TXSTA 24h  'Ativa TX(TXEN), Alta taxa de BaudRate(BRGH)
DEFINE HSER_BAUD 9600  'Configura Baud Rate para 9600
DEFINE HSER_RCSTA 90H  'Configura RX
```

Foram utilizadas três canais de AD internos do PIC16F877, sendo dois para leitura de temperaturas e um para a leitura da tensão da bateria. O quadro 13 apresenta as configurações necessárias para a utilização destes canais que também precisam ser especificados nas configurações.

Quadro 13: Definições dos canais de AD – Interface

```
' Configurações para as entradas analógicas
' AN0-TempInterna  AN1-TempExterna  AN3-VRef+  AN2-Bateria
ADCON1 = %00000011
DEFINE  ADC_BITS        8     'Define resolução de 8 bits
DEFINE  ADC_CLOCK       1     'Define clock do AD para Fosc/8
DEFINE  ADC_SAMPLEUS    50    'Define tempo de amostra para 50us
```

Foram utilizados pinos do PIC16F877 como *I/Os*, para o sensor de velocidade, acionamento de relés, sinal de alimentação do PC e sensor infravermelho. O quadro 14 apresenta as configurações para estes I/Os.

Quadro 14: Definição de I/Os individuais - Interface

```
' Configuração para o sensor de velocidade efeito Hall
Sen_vel     var PORTC.5   'Recebe os pulsos do sensor de velocidade

' Configuração para os pinos usados no comando do PC
Pwr_on      var PORTC.2   'Equilavente ao Power_On do PC
Rst_sw      var PORTC.1   'Equivalente ao Reset_Sw do PC
Sin_12      var PORTC.0   'Indica se o pc esta desligado
Inversor    var PORTB.1   'Alimentação do inversor de tensâo

' Configuração para o pino usado no comando do módulo de potência
Remoto      var PORTB.2   'Sinal remoto do módulo de potência

' Configuração para o receptor infravermelhor (IR)
Ir_sensor   var PortB.0   'Entrada física do sinal IR RC5

' Configuração individual destes I/Os
TRISC.0     = 1           'Entrada
TRISC.1     = 0           'Saída
TRISC.2     = 0           'Saída
TRISC.5     = 1           'Entrada
TRISB.0     = 1           'Entrada
TRISB.1     = 0           'Saída
TRISB.2     = 0           'Saída
```

No quadro 15, está descrita a declaração de variáveis utilizadas no *software* da interface. Os primeiros três níveis de declaração de variáveis referem-se exclusivamente á comunicação serial.

No último e mais extenso nível de declaração de variáveis, estão definidas todas as variáveis restantes do programa principal do *software* da interface, sendo que muitas delas são compartilhadas pelas funcionalidades da mesma.

Quadro 15: Declaração de variáveis - Interface

```
' Variáveis utilizadas na interrupção
wsave       var byte $70 system    'Backup de W
ssave       var byte bank0 system  'Backup de STATUS
psave       var byte bank0 system  'Backup de PCLATH
fsave       var byte bank0 system  'Backup de FSR

' Declaração de variáveis usadas pela interrupção serial
Rcif        var PIR1.5      'Alias RCIF (Flag de recepção USART)
Oerr        var RCSTA.1     'Alias OERR (Flag de erro USART)
Cren        var RCSTA.4     'Alias CREN (Recepção contínua USART)
Intf        var INTCON.1   'Alias INTF (Flag de interupção INT0)

' Declaração de variáveis relacionadas a comunicação serial
tam_buf     com 32        'Define o tamanho do buffer
buffer      var byte[tam_buf]'Vetor dos caracteres recebidos
index_in    var byte      'Indice auxiliar para bytes recebidos
index_out    var byte     'Índice auxiliar para bytes atualizados
bufchar      var byte     'Armazena o caracter recebido do buffer
errflag      var byte     'Contem flags de erro

' Declaração de variáveis
aux         var word      'Variável auxiliar para o prog principal
aux_vel     var word      'Conversão - Parte unitária da velocidade
volts1      var word      'Contem a tensão inteira da bateria
volts2      var word      'Contem a tensão decimal da bateria
amostra     var word      'Somatório - leitura nos ADs e velocidade
ind         var byte      'Usado com o comando for e Bit_vet
ind2        var byte      'Auxiliar na construção de Bits_vet
vet_vel     var word[18]  'Vetor de períodos para velocidade
velocid     var byte      'Contém a velocidade horária do veículo
vel_pc      var byte      'Contém a velocidade de referência
pc_on       var bit       'Indica se o PC está ligado
pc_pronto   var bit       'Indica se o PC está executando o Linux
bit_vet     var bit[250]  'Contém amostras do pacote RC5
bits_vet    var bit[28]   'Contém as duplas de bits do pacote RC5
pac_RC5     var word     'Contém o pacote RC5 final de 12 bits
tmp_RC5     var word      'Tempo de leitura sobre IR_Sensor
cont        var byte      'Auxiliar da decodificação de Bit_vet
ir_com       var byte     'Contém o código do botão pressionado
ir_sis       var byte     'Contém o código do sistema atual
ir_ctr      var byte      'Contém o bit de status - Start Bit
ind_pause   var byte      'Usado para looping com a função pause
sdata       var byte[1    'Contém a data atual dd/mm/aaaa
shora       var byte[8    'Contém a hora atual hh:mm:ss
```

A estrutura básica de implementação da interface é apresentada nos quadros 16, 17 e 18.

Quadro 16: Estrutura básica do *software* da interface – I

```
origem:
        goto tcc
.
.
{Tratador de interrupção, interrupção via INT0/RX, Le Serial/IR}
```

```
.
.
tcc:
         clear
         INTCON = %11010000 ' Ativa interrupções periféricas e INTF
         PIE1 = %00100000   ' Ativa interrupção para RX USART

inicio:
' Tenta estabelecer comunicação com o PC e obter velocidade refer.
        if ((not Sin_12) and (not pc_pronto)) then
         hserout ["i",10,13]          ' Solicita resposta do PC
         for ind = 1 to 10
          gosub atualiza_serial
          pause 10
         next ind
         svel_pc = " "
         hserout ["c",10,13]          ' Solicita velocidade refer.
         for ind = 1 to 10
          gosub atualiza_serial
          pause 10
         next ind
        endif

' Fornece a data atual caso o PC esteja ligado e pronto
        if (not Sin_12) and pc_pronto then
         hserout ["a",10,13]
         for ind = 1 to 10
          gosub atualiza_serial
          pause 10
         next ind
         lcdout $fe, $D4, "Data : ", str sdata \10
         for ind_pause = 1 to 40
          pause 50
          gosub atualiza_serial
          gosub atualiza_ir
         next ind_pause
        endif

' Fornece a hora atual caso o PC esteja ligado e pronto
        if (not Sin_12) and pc_pronto then
         hserout ["h",10,13]
         for ind = 1 to 10
          gosub atualiza_serial
          pause 10
         next ind
         lcdout $fe, $D4, "Hora : ", str shora \10
         for ind_pause = 1 to 40
          pause 50
          gosub atualiza_serial
          gosub atualiza_ir
         next ind_pause
        endif
```

Quadro 17: Estrutura básica do *software* da interface – II

```
' Leitura da Temperatura 1

         {le_temperatura}

          if (not Sin_12) then
          gosub atualiza_serial
          lcdout $fe, $D4, "Temperatura1 : ", dec amostra, $3, "C"
         else
          lcdout $fe, $80, "   TEMPERATURA 1    "
          lcdout $fe, $94 + 7, dec amostra, $3, "C"
         endif
        for ind_pause = 1 to 40
```

```
          if (not Sin_12) then gosub atualiza_serial
          gosub atualiza_ir
          pause 50
         next ind_pause

' Leitura da Temperatura 2

         {le_temperatura}

          if (not Sin_12) then
          gosub atualiza_serial
          lcdout $fe, $D4, "Temperatura2 : ", dec amostra, $3, "C"
         else
          lcdout $fe, $80, "   TEMPERATURA 2    "
          lcdout $fe, $94 + 7, dec amostra, $3, "C"
         endif
         for ind_pause = 1 to 40
          if (not Sin_12) then gosub atualiza_serial
          gosub atualiza_ir
          pause 50
         next ind_pause

' Leitura da tensão da bateria automotiva
bateria:
         gosub atualiza_ir
         if (not Sin_12) then atualiza_serial

         {le_tensao_bateria}

         if (volts1 < 11) and (not Sin_12) then
          lcdout $fe, $80, "BATERIA DESCARREGADA"
          gosub  desliga_pc          ' O PC será desligado
          goto bateria
         endif
         if (volts1<11 or volts1=11 and volts2<40) and (Sin_12) then
          lcdout $fe,$80, "BATERIA DESCARREGADA"
          goto bateria
         endif
         if (volts1  = 11 and volts2 < 40) then
          lcdout $fe, $D4, $2, " DESLIGUE O PC.       "
          goto bateria
         else
          gosub atualiza_ir
          if (not Sin_12) then atualiza_serial
         endif
```

Quadro 18: Estrutura básica do *software* da interface – III

```
' Leitura da velocidade do veículo em Km/h
          {mede_velocidade}

         if velocid <> 0 then
          'Apresenta o valor de velocidade obtido
          if (not Sin_12) then
           if (pc_pronto) then gosub atualiza_serial
           if vel_pc < velocid and pc_pronto then
            lcdout $fe, $D4, "REDUZA A VELOCIDADE!"
            hserout["g" + velocid,10,13]
           else
            lcdout $fe, $D4, "Velocidade : ", #velocid, "Km/h"
           endif
          else
           lcdout $fe, $80, "     VELOCIDADE     "
           lcdout $fe, $94, rep " " \7, #velocid, "Km/h      "
          endif
         endif
         for ind_pause = 1 to 40
```

```
            gosub atualiza_ir
            if (pc_pronto) then gosub atualiza_serial
            pause 50
           next ind_pause

   ' Verifica se o PC está desligado, para poder desligar o inversor
           if (Sin_12) then low inversor

           goto inicio                  'Volta ao início do programa.
   end                                  'Fim do programa
```

7.4.2 Rotina Desliga_PC

Esta rotina encerra a execução do *software* do PC, desligando-o no final do programa principal, através do byte "d" enviado via serial ao PC. No quadro 19 é apresentado o código desta rotina.

Quadro 19: Implementação da Rotina Desliga_PC - Interface

```
desliga_pc:
           lcdout $fe,1, "   Desligando o PC  "
           hserout ["d",10,13]
           for ind = 1 to 10
            gosub atualiza_serial
            pause 10
           next ind
           return
```

7.4.3 Rotina Liga_PC

O código da rotina Liga_PC é visto no quadro 20.

Quadro 20: Implementação da rotina Liga_PC – Interface

```
   liga_pc:
              lcdout $fe,1,"    Ligando o PC.   "
              high inversor
              pause 200
              pulsout Pwr_on,50000
              pause 300
              return
```

7.4.4 Rotina Atualiza_Serial

O quadro 21 apresenta o código desta rotina.

Quadro 21: Implementação da rotina Atualiza_Serial – Interface

```
atualiza_serial:
loopdisplay:
if erroflag then erro
if ind_ent = ind_sai then return
gosub obtembuf    ' Busca um caracter do buffer
if ind_sai = 1 then
select case bufchar
case "b" : lcdout $fe,$80 ' musicaxxxxxxxxxxxxxxxxxxx
case "m" : lcdout $fe,$C0 ' banda xxxxxxxxxxxxxxxxxxx
case "s" : lcdout $fe,$94 ' status play,pause,stop
case "k" : lcdout $fe,$94 + 2 ' faixa 000
case "t" : lcdout $fe,$94 + 6 ' tempo 00:00
case "v" : lcdout $fe,$94 + 12' volume Vol00
case "q" : lcdout $fe,$94 + 19' modo stereo ou mono
case "d" : pc_pronto = 0 ' desligando o PC
low remoto
case "i" : pc_pronto = 1 ' pc pronto para executar
high remoto
case "c" : criandovel = 1 ' criando vel_pc
case "a" : criandodata = 1 ' criando data
case "h" : criandohora = 1 ' criando hora
end select
else
if (bufchar <> "#") then lcdout bufchar
if (criandovel) then svel_pc[ind_var] = bufchar
if (criandodata) then sdata[ind_var] = bufchar
if (criandohora) then shora[ind_var] = bufchar
goto loopdisplay
endif
obtembuf:   ' Move o próximo caracter do buffer para bufchar
        INTCON = 0'Desativa interrupções na leitura do buffer
ind_sai = (ind_sai + 1)
if ind_sai > (tam_buf-1) then ind_sai = 0
bufchar = buffer[ind_sai]' Le posição atual do buffer
INTCON = %11000000      'Ativa novamente as interrupções
return
```

7.4.5 Rotina Atualiza_IR

O quadro 22 apresenta o código da rotina Atualiza_IR.

Quadro 22: Implementação da rotina Atualiza_IR – Interface

```
atualiza_ir:
        if (Ir_ctr) and (not Sin_12) and (pc_pronto) then
         if Ir_sis = 0 then
          select case Ir_com
           case 16: hserout["l",13] 'Volume +
           case 17: hserout["o",13] 'Volume -
          end select
         endif
         if Ir_sis = 5 then
          select case Ir_com
           case 12: 'Liga/Desliga PC
            if Sin_12 then gosub liga_pc
            else gosub desliga_pc endif
           case 50: hserout["w",13] 'Previous
           case 52: hserout["f",13] 'Next
           case 53: hserout["y",13] 'Play
           case 41: hserout["e",13] 'Pause
           case 54: hserout["p",13] 'Stop
          end select
         endif
        endif
        INTCON = %11010000
        PIE1 = %00100000
        INTF = 0
```

```
        Ir_ctr = 0
        return
```

7.4.6 Tratador de interrupções

O código do tratador de interrupções é apresentado no quadro 23. Foi utilizada a linguagem Assembly para a sua codificação.

Quadro 23: Implementação do tratador de interrupções – Interface

```
trata_int
        btfss INTCON,1 ;Testa se a interrupção foi por RB0
        btfss PIR1,5   ;Testa se a interrupção foi por RX USART
        goto  le_ir    ;se por RB0 desvia para ler o sensor IR
        goto  le_serial;se por RX desvia para recepção do buffer
```

7.4.7 Interrupção via RX – Rotina Le_Serial

O tratamento da interrupção via RX, é representado pela rotina Le_Serial, cujo código é apresentado no quadro 24.

Quadro 24: Implementação da rotina Le_Serial – Interface

```
le_serial   movwf wsave             ; Salva W
            swapf STATUS, W         ;
            clrf  STATUS            ; Limpa STATUS
            movwf ssave             ; Salva STATUS trocado
            movf  PCLATH, W         ; Move PCLATH para W
            movwf psave             ; Salva PCLATH
            movf  FSR, W            ; Move FSR para W
            movwf fsave             ; Salva FSR

; Verifica por erros de hardware na recepção
            btfsc RCSTA,OERR        ; Verifica erro na USART
            goto  erro_usart        ; salta para rotina de erro

; Encontra o banco de memória do buffer, e seta IRP
            if (_buffer > 0FFh)     ; Encontro o banco do buffer
                  bsf   STATUS,IRP  ; Se banco 2 ou 3 seta IRP
            else  bcf   STATUS,IRP  ; Se banco 0 ou 1 limpa IRP
            endif

; Verifica erros de buffer
            incf  _ind_ent, W ; incrementa index_in e joga em W
            subwf _ind_sai, W ; Testa index_out com index_in
            btfsc STATUS,Z    ; Se zero então ind_ent = ind_sai
            goto  erro_buffer ; Salta para rotina de erro buffer

; Incrementa ind_ent e zera se ultrapassar o limite do buffer
            incf  _ind_ent, F ; Incrementa index_in
            movf  _ind_ent, W ; Move novo new index_in para W
            sublw _tam_buf-1  ; Testa se encheu o buffer
            btfss STATUS,C    ; Se encheu então
            clrf  _ind_ent    ; Zera index_in

; Seta FSR com a posição do próximo endereço livre do buffer
```

```
            movlw Low _buffer ; Obtém a posição de buffer[0]
            addwf _ind_ent, W ; Incrementa ind_ent
            movwf FSR         ; Armazena o apontador em FSR

; Le e armazena o caracter da USART
            movf    RCREG, W  ; Le o caracter recebido
            movwf INDF        ; Coloca o caracter recebido em FSR

; Reestabelece os registradores FSR, PCLATH, STATUS e W
fim_int     movf  fsave, W          ; Backup do valor de FSR
            movwf FSR               ; Reestabelece FSR
            movf  psave, W          ; Backup do valor de PCLATH
            movwf PCLATH            ; Reestabelece PCLATH
            swapf ssave, W          ; Backup do valor de STATUS
            movwf STATUS            ; Reestabelece STATUS
            swapf wsave, F          ; Troca o valor de W
            swapf wsave, W          ; Reestabelece em W
            retfie                  ; Retorna da interrupção

; Rotinas de erro
erro_buffer bsf _erroflag,1   ; Seta o flag de erro do buffer
erro_usart  bsf _erroflag,0   ; Seta o flag de erro da USART
            movf  RCREG, W          ; Elimina o caracter recebido
            goto  fim_int          ; Volta ao programa principal
```

7.4.8 Interrupção via INT0 – Rotina Le_IR

O tratamento da interrupção via RB0 é apresentado no quadro 25.

Quadro 25: Implementação da rotina Le_IR – Interface

```
le_ir:  INTCON = 0
        for ind = 1 to 149    'Neste looping recolhe amostras
        Bit_vet[ind] = IR_Sensor 'do pacote RC5 e grava no array
        PauseUs 100              ' Bit_vet para ser decodificado
        next ind                 ' Amostras em tempos de 100us
        IR_Ctr=0                 ' Inicialização de variáveis
        IR_Sis=0
        IR_Com=0
        Pac_RC5=0
        for ind2 = 0 to 28
         Bits_vet[ind2]=0
        next ind2
        ind2 = 0
        cont = 1                 ' Define seqüências de 00 e 11
        bits_vet[ind2] = bit_vet[0]  ' Decodificação de Bit_vet
        for ind = 1 to 149       ' Varre Bit_vet
         if bit_vet[ind]=bits_vet[ind2] then ' de 0 e 1
          cont = cont + 1
         else
          select  case (cont/4) ' identifica sequencia 00 ou 11
           case 1 :             ' se 1 , período 888us de 1 ou 0
            ind2 = ind2 + 1
            bits_vet[ind2] = bit_vet[ind]
            cont = 1
           case 2, 3 :          ' se 2 ou 3, período 1,67ms
            ind2 = ind2 + 1     ' de 11 ou 00
            bits_vet[ind2] = bit_vet[ind-1]
            ind2 = ind2 + 1
            bits_vet[ind2] = bit_vet[ind]
            cont = 1
          end select
         endif     ' Agora Bits_vet possui as duplas de bits que
        next ind   ' irão constituir o pacote final RC5
        ind2 = 0           ' Inicio da criação do pacote RC5
        if not(not bits_vet[ind2+2] and bits_vet[ind2+3]) then
         goto fim_int
```

```
            For ind = 11 to 0 step -1
             ind2 = ind2 + 2 ' Vare Bits_vet. 00=0 01=1 10=0 11=1
             if (not bits_vet[ind2] and not bits_vet[ind2+1]) then
              Pac_RC5.0[ind] = 0
             if (not bits_vet[ind2] and bits_vet[ind2+1]) then
              Pac_RC5.0[ind] = 1
             if (bits_vet[ind2] and not bits_vet[ind2+1]) then
              Pac_RC5.0[ind] = 0
             if (bits_vet[ind2] and bits_vet[ind2+1]) then
              Pac_RC5.0[ind] = 1
            Next ind ' Agora  o pacote RC5 possui 12 bits.
            'Desmembra o pacote. 6 bits comando,5 sistema,1 controle
            IR_Com=Pac_RC5 & %00111111
            IR_Sis=(Pac_RC5 >>6) & %00011111
            IR_Ctr=(Pac_RC5 >>11)& %00000001
            return
```

7.4.9 Rotina de leitura da tensão da bateria

O código desta rotina está descrito no quadro 26.

Quadro 26: Implementação da rotina de leitura da tensão - Interface

```
aux = 0
amostra = 0
for ind = 1 to 20
 adcin 2, aux
 amostra = amostra + aux
next ind
aux = (((amostra/20)*62)+(((amostra/20)*5)/10)+(amostra//20))
volts1 = aux / 1000
volts2 = (aux // 1000) / 10
```

7.4.10 Rotina de leitura da temperatura

O código desta rotina está descrito no quadro 27.

Quadro 27: Implementação da rotina de leitura de temperatura – Interface

```
aux = 0
amostra = 0
for ind = 1 to 20
 adcin 4, aux
 amostra = amostra + aux
 aux = 0
 pause 30
 next ind
amostra = (amostra / 20)
```

7.4.11 Rotina de leitura de velocidade

O código da rotina de leitura de velocidade está no quadro 28.

Quadro 28: Implementação da rotina de leitura de velocidade - Interface

```
amostra = 0                ' Inicialização das variáveis
velocid = 0
ind = 0
while ((ind < 21) and (aux <> 0)) 'Obtenção de 20 períodos
 pulsin Sen_vel,1,aux
 amostra = amostra + aux
 ind = ind + 1
wend
if Aux > 0 then
 amostra = amostra / 20 'Obtém a média das 20 amostras
 ind = 0
 while ((ind < 18) and (velocid = 0)) 'Varredura sobre Vet_Vel
  if vet_vel[ind] = amostra then      'Se Amostra é igual então
   velocid = (ind + 1) * 10           'achou velocidade
  else
   if vet_vel[ind] > amostra then     'Se Amostra é menor então
    ind = ind + 1                     'incrementa indice
   else                               'Senão achou o intervalo
    'Aux, Aux_Vel e Amostra, são multiplicado por 10. Aux contém a
    'diferença do intervalo. Aux_Vel o valor inicial do intervalo.
    aux = ((vet_vel[ind-1] - vet_vel[ind]) * 10)
    aux_vel = vet_vel[ind-1] * 10
    amostra = amostra * 10
    velocid = ind * 10
    while amostra < aux_vel          'Se Amostra é maior então fim
     velocid = velocid + 1             'Incrementa velocid em 1
     aux_vel = aux_vel - (aux / 10)   'Soma 10% de Aux. =1 em veloc
    wend
   endif
  endif
 wend
endif
```

7.5 Implementação do software do PC

O PC possui o sistema operacional Linux Red Hat 7.3 e o *software* do PC foi desenvolvido em linguagem C usando o compilador "*gcc*".

7.5.1 Programa principal

O programa principal foi inteiramente direcionado para comando via porta serial, ou seja, todas as funcionalidades do mesmo, são executadas através da recepção de dados via porta serial RS232, provenientes da interface microcontrolada. Como retorno a estes dados, são enviados informações de controle, status dos arquivos MP3 executados, informações de data e hora e também ocorrem registros de excessos de velocidade no arquivo logvel.log. Na tela do programa principal, são apresentadas mensagens de status, informando dados recepcionados, ou dados enviados via porta serial.

Basicamente, o programa principal, executa três processos auxiliares, relacionados diretamente com a funcionalidade de execução de arquivos MP3. Estes programas são :

- mp3info – consulta informações ID3 *Tag* dos arquivos MP3;
- aumix – consulta, aumenta ou diminui o nível de volume geral das músicas;
- mpg321 – reproduz os arquivos MP3, com funções internas para *pause*, *stop*, *play* obtenção do tempo decorrido das músicas.

Os dois primeiros programas são chamados através das funções, LeID3 e Volume, respectivamente, enviando serialmente a informação solicitada e encerrando logo após, enquanto que o programa mpg321 executa juntamente com o programa principal até o seu encerramento. Existem também funções para utilização da porta serial, sendo elas, AbrePortaSerial, EscrevePortaSerial, EnviaDadosSeriais, FechaPortaSerial.

Durante a execução do programa principal, são detectados 9 tipos diferentes de erros, mais relevantes, que podem influenciar diretamente no programa principal, erros estes que são identificados e apresentados pela função FinalizaTudo, que ainda fecha a porta serial e encerra o processo gerado pelo programa mpg321.

Como ilustrado na seção de especificação do *software* do PC, o programa principal segue uma lista de atividades específicas de inicialização identificando erros, caso ocorrerem, para então entrar em execução em loop infinito, lendo os dados recepcionados via serial e provenientes da interface microcontrolada. As atividades específicas, são a criação do arquivo *Mp3.lst*, geração de uma lista de músicas através deste arquivo, leitura da velocidade de referência contida em velocmax.cfg, criação do processo mpg321, abertura e configuração da porta serial, abertura do arquivo logvel.log para registro de excessos de velocidade, envio de status de execução para a interface e envio da velocidade de referência para a interface, sendo que a partir deste

ponto fica em looping executando tarefas, como tocar, pausar, parar, avançar, retornar músicas MP3, fornecer data e hora, registrar velocidades excessivas e encerrar o PC. Os quadros 29, 30, 31, 32, 33 e 34 apresentam a implementação do *software* do PC.

Quadro 29: Implementação do *software* do PC – I

```
/* Programa EPIA.C Autor : Cristiano Freese Data  : 10/06/03 */
/* Definição das bibliotecas utilizadas */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#define portaserial "/dev/ttyS0"

/* Definição de variáveis globais */
 int porta; /* Apontador para a porta serial */
 struct termios config; /* Configuração da porta serial */
 char bufferserial[30]; /* Buffer de envio de dados para a serial */

/* Declaração de funções usadas pelo programa principal */
int AbrePortaSerial(char *portserial)
int FechaPortaSerial()
int EnviaDadosSeriais(char *bufferescrita)
int LeDadosSeriais(int (*bufferleitura))
int FinalizaTudo(int iderr)
int topo()
int LeID3(char *op, char *caminho)
int Volume(char *opvol, int volum)
/* Inicio do programa principal
main()
{
 /* Declaração de variáveis locais */
 int erro;     /* identificador de erro de atividade não completada */
 FILE *arqmp3; /* Ponteiro para o arquivo Mp3.lst */
 FILE *arqvel; /* Ponteiro para o arquivo velocmax.cfg */
 char velmax[4];  /* String contendo o dado lido em velocmax.cfg */
 FILE *arqlogvel; /* Ponteiro para o arquiov logvel.log */
 char vetmp3[100][30]; /* Armazena os paths dos arquivos MP3 */
 int indvetmp3;   /* Indice auxiliar para vetmp3 */
 int charserial;  /* Buffer para bytes recepcionados serialmente */
 int pipesmpg321[2];   /* Descritor utilizado pelo processo mpg321 */
 int forkmpg321;  /* Pid do processo filho mpg321 */
 char buffermpg321[128]; /* Buffer para comandos no processo mpg321 */
 char velog[30];  /* Armazena velocidades e insere em logvel.log */
 int registravel; /* Flag  de recepção de velocidades */
 struct tm *apont_tm;  /* Estrutura de informações de data e hora */
 time_t tempo;         /* Contém dados de data e hora */
 int esperaserial;/* Contador auxiliar para temporizar recepção */
 int modompg321;  /* Status de execução de MP3. 0stop,1pause,2play */
 char pathid3[30];/* Caminho do arquivo MP3 para leitura de ID3 */
 int indstr;      /* Indice auxiliar para manipulação de strings */
```

```
int vol;          /* Volume atual ajustado para as músicas MP3 */
char tempompg321[6];  /* string com tempo decorrido/total dos MP3s */
int espaco;   /* Espaços para obter tempo decorrido/total dos MP3s */
int indstr2;  /* Indice auxiliar 2 para manipulação de strings. */
int tempompg321int;   /* Segundos decorridos do MP3 executado */
int tempototalmpg321int; /* Segudos totais do MP3 executado */
int atualizatempompg321; /* Escalona atualizações de tempo do MP3 */
```

Quadro 30: Implementação do *software* do PC – II

```
system("clear");
printf("TCC - COMPUTADOR DE BORDO PC-LINUX - INICIALIZANDO\n\n\n");

/* Cria Mp3.lst contendo os arquivos MP3 do diretório /Tcc/MP3 */
printf("Criando arquivo Mp3.lst ...\n");
system("rm -f /Tcc/Mp3.lst");
system("find /Tcc/MP3/*.mp3 >> /Tcc/Mp3.lst");

/* Cria o vetor vetmp3 baseado no conteúdo do arquivo Mp3.lst */
printf("Criando vetor de arquivos MP3 ...\n");
if ((arqmp3=fopen("/Tcc/Mp3.lst","r")) == NULL)
{ erro = 5;
 FinalizaTudo(erro);  } else
{ indvetmp3=0;
  while ((fgets(vetmp3[indvetmp3], 30, arqmp3))!=NULL) indvetmp3++;
  fclose(arqmp3); }

/* Abre Velocmax.cfg para ler o valor da velocidade de referência */
printf("Obtendo velocidade de referência para a interface ...\n");
if ((arqvel=fopen("/Tcc/Velocmax.cfg","r")) == NULL)
{ erro = 6;
  FinalizaTudo(erro); } else
{ fgets(velmax,4,arqvel); /* Atribui o valor velmax */
  fclose(arqvel); }

/* Abre logvel.log para registrar os excessos de velocidade */
printf("Abrindo o arquivo logvel.log ...\n");
if ((arqlogvel=fopen("/Tcc/logvel.log","a")) == NULL)
{ erro = 7;
  FinalizaTudo(erro); }

/* Cria o pipe pipesmpg321 para a processo filho mpg321 */
printf("Criando os pipes do processo mpg321 ...\n");
if (pipe(pipesmpg321) == 0) {
 forkmpg321 = fork();
 if (forkmpg321 == -1)
 { erro = 8;
   FinalizaTudo(erro); }
 if (forkmpg321 == 0) /* Execução do processo filho mpg321 */
 { close(0); /* Fecha a entrada padrão do processo filho mpg321 */
   dup(pipesmpg321[0]); /* pipesmpg321[0] é a nova entrada padrão */
   close(pipesmpg321[0]);
   close(1); /* Fecha a saída padrão do processo filho mpg321 */
   dup(pipesmpg321[1]);   /* pipesmpg321[1] é a nova saída padrão */
   close(pipesmpg321[1]);
   close(2);
   execlp("mpg123","mpg123","tcc","-R",(char *)0);/*executa mpg321 */
   while (1); } } else
{ erro = 1;
  FinalizaTudo(erro); }

/* Testa se o programa filho mpg321 emitiu o status '@R MPG123' */
printf("Recebendo status inicial de mpg321 ...\n");
while(read(pipesmpg321[0],buffermpg321,sizeof(buffermpg321))==0);
if (strcmp(buffermpg321,"@R MPG123")==0)
{ erro = 9;
  FinalizaTudo(erro); }
```

Quadro 31: Implementação do *software* do PC – III

```
/* Abre/configura a serial /dev/ttyS0. 9600bauds, 8bits, 1stop bit */
 printf("Abrindo a porta serial ...\n");
 if (AbrePortaSerial(portaserial) == -1)
 { erro = 2;
   FinalizaTudo(erro); }

 /* Looping por 20 segundos aguardando 'i' para enviar confirmação */
 esperaserial = 0;  charserial = 33;
 while ((esperaserial < 20) && (charserial == 33))
 { printf("Aguardando status ... Tentativa %d\n",esperaserial);
   LeDadosSeriais(&charserial); /* Le a serial em busca de dados */
   sleep(1); esperaserial++; }
 if (charserial != 105)
 { erro = 3;
   FinalizaTudo(erro); } else
 { bufferserial[0] = '\0'; /* Zera o bufferserial */
   strcat(bufferserial,"i#"); /* Prepara bufferserial */
   EnviaDadosSeriais(bufferserial); /* Envia confirmação */ }

 /* Looping por 20 segundos aguardando 'c' para envio da velocidade */
 esperaserial = 0; charserial = 33;
 while ((esperaserial < 20) && (charserial == 33))
 { printf("Aguardando vel monit... Tentativa %d\n",esperaserial);
   LeDadosSeriais(&charserial); /* Le a serial em busca de dados */
   sleep(1);
   esperaserial++; }
 if (charserial != 99)
 { erro = 4;
   FinalizaTudo(erro); } else
 { bufferserial[0] = '\0'; /* Zera o bufferserial */
   strcat(bufferserial,"c"); /* Prepara bufferserial */
   strcat(bufferserial,velmax);
   strcat(bufferserial,"#");
   EnviaDadosSeriais(bufferserial); /* Envia vel via serial */ }

 /* Execução em looping que executará ações via dados seriais */
 printf("\nSistema pronto . Aguardando comandos via serial ...\n");
 sleep(1); topo(); /* Exibe status do programa principal */
 charserial = 33; indvetmp3 = 0;
 while (1) /* Looping infinito, lendo a porta serial*/
 { LeDadosSeriais(&charserial);
   switch (charserial) /* Testa se algum caracter foi recebido */
   { case 100: erro = 0; /* Solicitação de encerramento do programa */
     bufferserial[0] = '\0'; /* Zera o bufferserial */
     strcat(bufferserial,"d#"); /* Prepara bufferserial */
     EnviaDadosSeriais(bufferserial); /* Envia encerramento */
     fclose(arqlogvel);
     if (modompg321==2||modompg321==1) write(pipesmpg321[1],"s\n",2);
     close(pipesmpg321[1]); close(pipesmpg321[0]);
     kill(forkmpg321, SIGTERM); waitpid(forkmpg321, NULL, 0);
     FinalizaTudo(erro); break;

     case 97: (void) time(&tempo); /* Solicitação de data atual */
     apont_tm = localtime(&tempo); /* armazena a data tm tempo */
     strftime(bufferserial, 11, "a%d/%m/%y#", apont_tm);
     EnviaDadosSeriais(bufferserial); /* Envia a data via serial */
     printf("Dt:%02d/%02d/03\n",apont_tm->tm_mday,apont_tm->tm_mon+1);
     break;
```

Quadro 32: Implementação do *software* do PC – IV

```
     case 104: (void) time(&tempo); /* Solicitação de hora atual */
     apont_tm = localtime(&tempo); /* insere hora atual em tm tempo */
     bufferserial[0] ='\0';
     strftime(bufferserial, 11, "h%H:%M:%S#", apont_tm);
     EnviaDadosSeriais(bufferserial); /* Envia a hora via serial */
```

```
printf("Hr:%02d:%02d:%02d\n",apont_tm->tm_hour,apont_tm->tm_min,
apont_tm->tm_sec); break;

case 103: registravel = 1; /* Inicia leitura da velocidade */
printf("Recebeu - Velocidade: ");
(void) time(&tempo);
apont_tm = localtime(&tempo);
velog[0]='\0';
strftime(velog, 22, "%d/%m/%y %H:%M:%S - ", apont_tm);
fputs(velog,arqlogvel);
velog[0] = '\0';
break;
case 48:  strcat(velog,"0"); break;
case 49:  strcat(velog,"1"); break;
case 50:  strcat(velog,"2"); break;
case 51:  strcat(velog,"3"); break;
case 52:  strcat(velog,"4"); break;
case 53:  strcat(velog,"5"); break;
case 54:  strcat(velog,"6"); break;
case 55:  strcat(velog,"7"); break;
case 56:  strcat(velog,"8"); break;
case 57:  strcat(velog,"9"); break;
case 64:  registravel = 0; /* Finaliza inserção do registro */
printf("%s\n",velog);
fputs(velog, arqlogvel); /* Insere a velocidade em logvel.log */
fputs("\n", arqlogvel); /* Posiciona próx linha de logvel.log */
break;

case 121: if (modompg321 == 2 || modompg321 == 1) break;
topo(); printf("Recebeu - Play \n");
pathid3[indstr] = vetmp3[indvetmp3][indstr];
sprintf(pathid3,"%s",pathid3);
LeID3("b",pathid3); /* Envia nome da banda via serial */
LeID3("m",pathid3); /* Envia nome da música via serial */
LeID3("q",pathid3); /* Envia qualidade S ou M via serial */
modompg321 = 2; /* Esta tocando a música MP3 - Play */
printf("Enviou  - Status Tocando\n");
EnviaDadosSeriais("s2#"); /* Envia status de Play */
printf("Enviou  - Tempo 00:00\n");
EnviaDadosSeriais("t00:00#"); /* Envia tempo da música */
if (indvetmp3 < 9) sprintf(bufferserial,"k00%d#",indvetmp3+1);
sprintf(bufferserial,"k%d#",indvetmp3+1);
printf("Enviou  - Faixa %d\n",indvetmp3+1);
EnviaDadosSeriais(bufferserial);
vol = Volume("c",vol); /* Envia o valor de volume via serial */
buffermpg321[0]='\0';
sprintf(buffermpg321,"l %s\n",vetmp3[indvetmp3]);
write(pipesmpg321[1], buffermpg321, strlen(buffermpg321));
sleep(1);
tempototalmpg321int = 999; /* Inicializa tempototalmpg321 */
charserial = 33; /* Inicializa charserial */
break;
```

Quadro 33: Implementação do *software* do PC – V

```
case 101: if (modompg321 == 0) break; /* Comando Pause */
buffermpg321[0] = '\0';
sprintf(buffermpg321,"p\n");
write(pipesmpg321[1], buffermpg321, strlen(buffermpg321));
printf("Recebeu - Pause \n");
if (modompg321 == 1)
{ modompg321 = 2;
  EnviaDadosSeriais("s2#");
  printf("Enviou  - Status Tocando\n");
  break; }
 if (modompg321 == 2)
 { modompg321 = 1;
```

```
        EnviaDadosSeriais("s1#");
        printf("Enviou  - Status Pausado\n");
        break; }

    case 102: if (indvetmp3 < 100) indvetmp3++; /* Comando Next */
    printf("Recebeu - Next\n");
    if (modompg321 == 2 || modompg321 == 1)
    { buffermpg321[0] = '\0';
      sprintf(buffermpg321,"s\n"); /* Para a execução da música */
      write(pipesmpg321[1], buffermpg321, strlen(buffermpg321));
      if (vetmp3[indvetmp3][0] != '/') indvetmp3--; }
    modompg321 = 0;
    sleep(1); break;

    case 119: if (indvetmp3 > 0) indvetmp3--; /* Comando Previous */
    printf("Recebeu - Previous\n");
    if (modompg321 == 2 || modompg321 == 1)
    { buffermpg321[0] = '\0';
      sprintf(buffermpg321,"s\n"); /* Para a execução da música */
      write(pipesmpg321[1], buffermpg321, strlen(buffermpg321)); }
    modompg321 = 0;
    sleep(1); break;

    case 112: if (modompg321 == 0) break; /* Comando Stop */ else
    { printf("Recebeu - Stop\n");
      buffermpg321[0] = '\0';
      sprintf(buffermpg321,"s\n");
      write(pipesmpg321[1], buffermpg321, strlen(buffermpg321));
      modompg321 = 0; /* Seta status da música para parado */
      EnviaDadosSeriais("s0#"); /* Envia status via serial */
      printf("Enviou  - Status Parado\n");
      break; }

    case 111: if (modompg321 == 0) break; else
    { printf("Recebeu - Volume +\n");
      vol = Volume("+",vol); /* Aumenta o volume em um nível */
      break; }

    case 108: if (modompg321 == 0) break; else
    { printf("Recebeu - Volume -\n"); /* Diminui o volume */
      vol = Volume("-",vol);
      break; } }

  if ((charserial == 102) || (charserial == 119)) charserial = 121;
  else charserial = 33;
```

Quadro 34: Implementação do *software* do PC – VI

```
  if (modompg321 == 2)
  { for (indstr=0;indstr<128;indstr++) buffermpg321[indstr]='\0';
    if (tempototalmpg321int == tempompg321int)
    { if (vetmp3[indvetmp3+1][0] != '/')
      { modompg321 == 0;
        EnviaDadosSeriais("s0#");
        printf("Enviou  - Status Parado - Fim da Lista de MP3\n");
        tempototalmpg321int = 999; }
      charserial = 102;
      sleep(1); } else
    {while(read(pipempg321[0],buffermpg321,sizeof(buffermpg321))==0);
    indstr = 0; indstr2 = 0; espaco = 0; tempompg321[0]= '\0';
    indstr2++;
    while(espaco <4)
    { if ((buffermpg321[indstr]) == ' ') espaco++;
      if (espaco == 3) indstr2++;
      indstr++; }
    indstr2 = indstr2 - 2;
    indstr = indstr - 1 - indstr2;
```

```
     strncat(tempompg321, buffermpg321+indstr, indstr2);
     tempompg321int = atoi(tempompg321);

     if (tempototalmpg321int == 999)
     { indstr = 0; indstr2 = 0; espaco = 0; tempompg321[0]= '\0';
       indstr2++;
       while(buffermpg321[indstr] != '\0')
       { if ((buffermpg321[indstr]) == ' ') espaco++;
         if (espaco == 4) indstr2++;
         indstr++; }
       indstr2 = indstr2 - 2;
       indstr = indstr - 1 - indstr2;
       strncat(tempompg321, buffermpg321+indstr, indstr2);
       tempototalmpg321int = atoi(tempompg321); }
     sprintf(bufferserial,"t%02d:%02d#",(tempompg321int / 60),
     (tempompg321int % 60));
     if (atualizatempompg321 == 6) /* Escalona o envio do tempo */
     { EnviaDadosSeriais(bufferserial);
       printf("Enviou  - Tempo decorrido %02d:%02d\n",
        (tempompg321int / 60), (tempompg321int % 60));
       atualizatempompg321 = 0; }
     atualizatempompg321++; /* Incrementa escalonador */ } } } }
```

7.5.2 Processo mpg321

O quadro 30 apresenta a criação do processo filho mpg321 através dos descritores de arquivos denominados pipesmpg321.

7.5.3 Processo aumix e mp3info

No quadro 35 é apresentado o código gerado para o processo aumix através da função Volume(). No quadro 36 é apresentado o código gerado para o processo mp3info através da função LeID3().

Quadro 35: Implementação da função Volume

```
int Volume(char *opvol, int volum)
{ int pipesaumix[2]; /* Descritor utilizado pelo processo aumix */
  int forkaumix; /* Pid do processo aumix */
  char bufferaumix[128]; /* Buffer para leitura do processo aumix */
  char volume[3]; /* volume a ser manipulado para a função execlp */
  if (pipe(pipesaumix) == 0) /* Criação de pipesaumix para aumix */
  { forkaumix = fork();
    if (forkaumix == -1) return(volum);
    if (forkaumix == 0) /* Execução do processo filho aumix */
    { close(1); /* Fechamento da saída padrão do processo aumix */
      dup(pipesaumix[1]); /* Nova saída padrão do processo */
      close(pipesaumix[1]);
      close(pipesaumix[0]);
      if (opvol == "+") /* Aumentar o volume em um nível */
      { if (volum <99) volum++; /* Testa se já está no máximo */
```

```
        volume[0]='\0'; volume[1]='\0';
        sprintf(volume,"-v%d",volum);
        execlp("aumix", "aumix", volume,"-vq", (char *)0); }
      if (opvol == "-") /* Diminuir o volume em um nível */
      { if (volum >0) volum--; /* Testa se já está no mínimo */
        volume[0]='\0'; volume[1]='\0';
        sprintf(volume,"-v%d",volum);
        execlp("aumix", "aumix", volume,"-vq", (char *)0); }
      if (opvol == "c") /* Comando para consultar o volume atual */
      { volume[0]='\0'; volume[1]='\0';
        sprintf(volume,"-v%d",volum);
        execlp("aumix", "aumix", volume,"-vq", (char *)0); } }
    else /* Execução do processo pai */
    { close(pipesaumix[1]);
      while (read(pipesaumix[0],bufferaumix,sizeof(bufferaumix))== 0);
      if (opvol == "+") /* Volume incrementado e enviado serialmente*/
      { sprintf(bufferserial,"v%c%c#",bufferaumix[4],bufferaumix[5]);
        EnviaDadosSeriais(bufferserial);
        printf("Enviou –Volume %c%c\n",bufferaumix[4],bufferaumix[5]);
        if (volum < 99) return(volum+1); }
      if (opvol == "-") /* Volume derementado e enviado serialmente */
      { sprintf(bufferserial,"v%c%c#",bufferaumix[4],bufferaumix[5]);
        EnviaDadosSeriais(bufferserial);
        printf("Enviou- Volume %c%c\n",bufferaumix[4],bufferaumix[5]);
        if (volum > 0) return(volum-1); }
      if (opvol == "c") /* Volume consultado e enviado serialmente */
      { sprintf(bufferserial,"v%c%c#",bufferaumix[4],bufferaumix[5]);
        EnviaDadosSeriais(bufferserial);
        printf("Enviou- Volume %c%c\n",bufferaumix[4],bufferaumix[5]);
        return(volum); }
      close(pipesaumix[0]); /* Encerramento de pipesaumix */
      kill(forkaumix, SIGTERM);
      waitpid(forkaumix, NULL, 0); /* Encerramento de aumix */ } }
  return(volum); /* Retorno do volume atual */ }
```

Quadro 36: Implementação da função LeID3

```
int LeID3(char *op, char *caminho)
{ int pipesid3[2]; /* Pipe usado pelo processo MP3INFO */
  int forkid3; /* Pid do processo filho MP3INFO */
  char bufid3[128]; /* Buffer para leitura de dados de MP3INFO */
  int indbufid3; /* Indice auxiliar para manipulação de bufid3 */

  if (pipe(pipesid3) == 0) /* Criação do pipe pipesid3 */
  { forkid3 = fork();
    if (forkid3 == -1) return(-1);
    if (forkid3 == 0) /* Execução do processo filho */
    { close(1); /* Fecha sua saída padrão */
      dup(pipesid3[1]); /* Nova saída padrão do processo */
      close(pipesid3[1]);
      close(pipesid3[0]); /* Executa a ação selecinado em op */
      if (op == "m") execlp("mp3info", "mp3info", "-p","%t", caminho,
       (char *)0); /* Consulta música */
      if (op == "b") execlp("mp3info", "mp3info", "-p","%a", caminho,
       (char *)0); /* Consulta banda */
      if (op == "q") execlp("mp3info", "mp3info", "-p","%o", caminho,
       (char *)0); /* Consulta qualidade */ }
    else /* Execução do processo pai */
    { close(pipesid3[1]);
      while (read(pipesid3[0], bufid3, sizeof(bufid3)) == 0);
      if (op == "m") /* Envia o nome da musica serialmente */
      { bufferserial[0] = '\0';
        strcat(bufferserial,"b");
        strcat(bufferserial,bufid3);
```

```
        strcat(bufferserial,"#");
        EnviaDadosSeriais(bufferserial);
        printf("Enviou  - Música %s\n",bufid3); }
      if (op == "b") /* Envia o nome da banda serialmente */
      { bufferserial[0] = '\0';
        strcat(bufferserial,"m");
        strcat(bufferserial,bufid3);
        strcat(bufferserial,"#");
        EnviaDadosSeriais(bufferserial);
        printf("Enviou  - Banda %s\n",bufid3); }
      if (op == "q") /* Envia qualidade. 1=Stereo e 0=Mono*/
      { bufferserial[0] = '\0';
        strcat(bufferserial,"q");
        if((strcmp(bufid3,"joint stereo"))==0)strcat(buffrserial,"1");
        else strcat(bufferserial,"0");
        strcat(bufferserial,"#");
        EnviaDadosSeriais(bufferserial);
        printf("Enviou  - Qualidade %s\n",bufid3);}
      close(pipesid3[0]); /* Finaliza pipesid3[0] */
      kill(forkid3, SIGTERM);
      waitpid(forkid3, NULL, 0); /* Encerra o processo MP3INFO */ } }
  return(0); }
```

7.5.4 Funções para a comunicação serial

No quadro 37 estão agrupadas todas as funções usadas na comunicação serial.

Quadro 37: Implementação da comunicação serial - PC

```
/* Esta função abre a porta serial designada em portserial e retorna
   -1 se houve erro ou 0 se sucesso */
int AbrePortaSerial(char *portserial)
{ if((porta=open(portserial,O_RDWR|O_NOCTTY|O_NDELAY))==-1)return(-1);
  config.c_iflag = IGNPAR;
  config.c_oflag = 0;
  config.c_lflag = 0;
  config.c_cflag = (B9600 | CS8 | CLOCAL | CREAD);
  config.c_cc [VMIN] = 0;
  config.c_cc [VTIME] = 0;
  tcflush (porta, TCIFLUSH);
  tcsetattr(porta, TCSANOW, &config);
  return(0); }

/* Esta função fecha a porta serial designada em portserial e retorna
   -1 se houve erro ou 0 se sucesso */
int FechaPortaSerial()
{ if ((close(porta)) == -1) return(-1);
  return(0); }

/* Esta função envia uma string bufferescrita via serial */
```

```
int EnviaDadosSeriais(char *bufferescrita)
{ write(porta, bufferescrita, strlen(bufferescrita)); }

/* Esta função recebe um byte bufferleitura via serial */
int LeDadosSeriais(int (*bufferleitura))
{ read(porta, bufferleitura, 1); }
```

7.6 Software de transferência de arquivos via FTP

Trata-se de um *software* básico FTP, para facilitar a transferência de arquivos MP3 e leitura do arquivo de velocidades no PC, sendo responsável pela remoção das músicas, desenvolvido no ambiente de desenvolvimento Delphi 5.0 com o componente NMFtp.

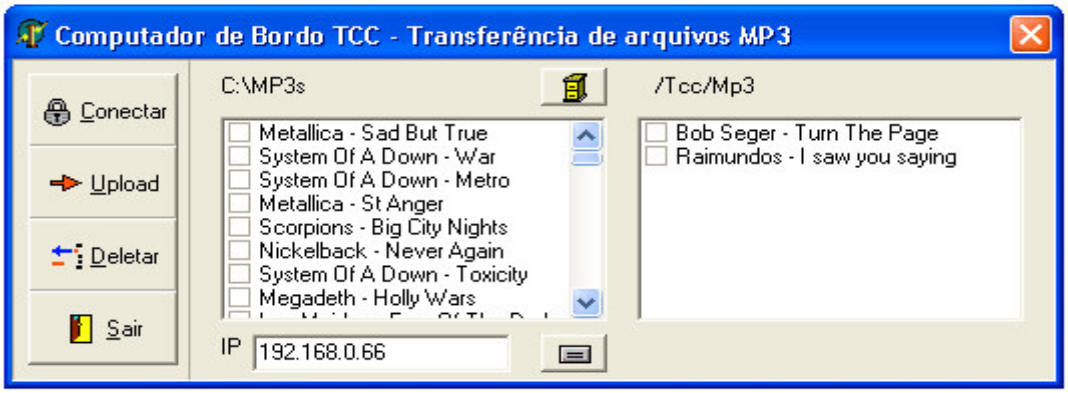


Figura 42: *Software* de transferência de arquivos MP3

7.7 Integração hardware Interface/PC

Esta seção descreve a integração entre o *hardware* do PC e da interface. Inicialmente é apresentada a composição de cada um deles.

O *hardware* da interface microcontrolada é apresentada na tabela 14. Os demais componentes são de uso geral e não necessitam de maiores esclarecimentos.

Na tabela 15 é apresentada a disposição dos pinos utilizados pelo microcontrolador PIC 16F877 na interface microcontrolada.

Tabela 14: Componentes principais da interface microcontrolada

Componente	Descrição
Microcontrolador PIC16C877	Responsável pelo gerenciamento de toda interface. Recebe e transmite dados serialmente, decodifica sinais infravermelhos obtidos pelo receptor infravermelho seguindo a norma RC5, aciona relés, lê A/Ds para indicação de temperaturas e carga de bateria, lê pulsos de um sensor efeito Hall para cálculo de velocidade.
Controle Remoto	Responsável pela emissão de pacotes infravermelho seguindo a

SystemLINK4	norma RC5 da Phillips.
Oscilador 20MHz	Responsável pelo *clock* externo do microcontrolador PIC16C877.
Conversor MAX232	Responsável pela conversão de sinais lógicos TTL para o sinais padrão RS232, possibilitando a comunicação entre interface e o PC.
Receptor PHC38C	Receptor de sinais infravermelho com freqüência de operação interna de 38KHz, provenientes do emissor infravermelho.
Sensor LM35	Responsável pela leitura de temperatura em °C
Regulador LM7805	Responsável pela tensão de alimentação da interface, regulando a mesma para 5Vcc.
Display LCD WH2004A	Responsável pela interação homem/máquina de todo sistema. Este é único meio de consulta visual sobre a operacionalidade de todo o sistema. Trabalha com fluxo de dados de 4 bits.
Sensor V305	Sensor de efeito Hall, que emite pulsos de período proporcional á velocidade do veículo e que são convertidos em valores de velocidade em Km/h.
Trimpot 2K2Ω	Cria um divisor de tensão para a tensão da bateria de forma a torná-lo equivalente ás tensões gerados pelos sensores de temperatura, podendo utilizar desta forma a mesma tensão de referência.
Relés	Responsáveis por pulsos nos botões "Pwr_On" e "Rst_Sw" do PC, alimentação do inversor de tensão e habilitação do módulo de potência através do sinal remoto.
Acoplador 4N25	Responsável por isolar o sinal de 12V do PC, da interface microcontrolada.

Tabela 15: Disposição dos pinos do PIC16F877 utilizado na interface

Pino	**Descrição**
1	Reset. Ativo em nível "0"
2	Leitura A/D Temperatura 1 (Sensor LM35 – 0 a 2,55V – 0 a 255°C)
3	Leitura A/D Temperatura 2 (Sensor LM35 – 0 a 2,55V – 0 a 255°C)
4	Leitura A/D Tensão Bateria (Trimpot 2K2Ω - 0 a 2,55V – 0 a 16V)
5	Tensão de referência para os A/Ds – 2,55V
11	Tensão 5Vcc
12	Terra (GND)
13	Oscilador Cristal 1
14	Oscilador Cristal 2
15	Sinal 12V da alimentação via acoplador 4N25 (0 – PC ligado, 1 – PC desligado)
16	Relé "Pwr_On" do PC
17	Relé "Rst_Sw" do PC
18	Bit "E" do *display* LCD
23	Bit "R/S" do *display* LCD
24	Sinal pulsante do sensor Hall de velocidade
25	TX – ligado ao driver MAX232 (transmite dados ao PC serialmente)

26	RX – ligado ao driver MAX232 (recebe dados do PC serialmente) via interrupção
31	Terra (GND)
32	Tensão 5Vcc
33	Pacote de bits proveniente do receptor infravermelho PHC38C via interrupção
34	Relé Sinal "Remoto" do módulo de potência
35	Relé de alimentação do inversor de tensão
37	Bit "D4" do *display* LCD
38	Bit "D5" do d*isplay* LCD
39	Bit "D6" do *display* LCD
40	Bit "D7" do *display* LCD

Todos os comandos solicitados pelo usuário, são possíveis somente via controle remoto. A tabela 16 apresenta os botões disponíveis com as respectivas funções.

Tabela 16: Botões habilitados no controle remoto – Interface

Botão	**Função**
On-Off	Liga/Desliga o PC
Play	Executa músicas MP3
Stop	Para a execução da música MP3
Pause	Pausa a execução da música MP3
Next	Avança uma música MP3 da lista
Previous	Retrocede uma música MP3 da lista
Vol -	Diminui o volume
Vol +	Aumenta o volume

O *hardware* do PC está descrito na tabela 17.

Tabela 17: Componentes principais do PC

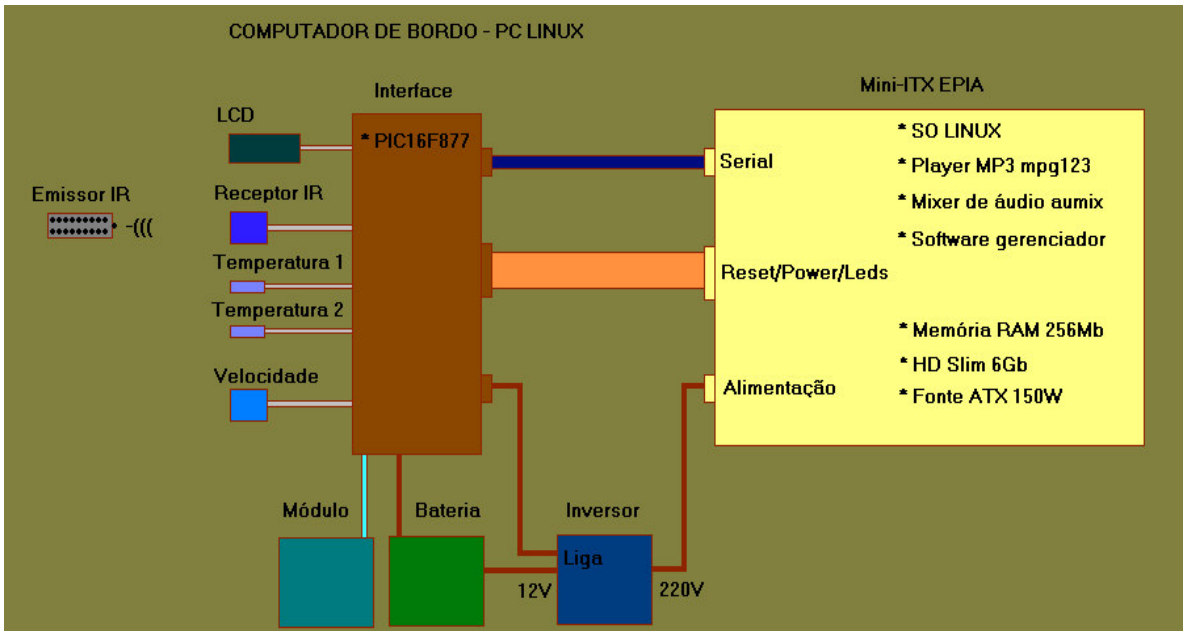
Componente	**Descrição**
Placa-Mãe Mini-ITX VIA EPIA800	Placa-mãe extremamente compacta (170x170) com alta integração de periféricos. Destes periféricos, são utilizados o som, a rede, e a porta serial RS232. Possui o processador Cyrix C3 800MHz EBGA com cooler.
HardDisk Quantun Slim 6Gb	Responsável pelo armazenamento do sistema operacional Linux Red Hat 7.3, do *software* do PC, das músicas MP3 e arquivos de configuração e registro de velocidade. Possui apenas alimentação 5Vcc, e tamanho extremamente reduzido, sendo conectado na interface IDE da placa-mãe através de um adaptador especial IDE para HDs padrão Slim.
Memória RAM PC133 DIMM 256Mb	
Fonte ATX 150W 220V	Responsável pelo fornecimento das tensões necessárias ao funcionamento de placas-mãe com alimentação padrão ATX, ou seja, fornece, ± 12V, ±5V, +3,3V, -12V, -5V.
Cooler externo	responsável pela circulação de ar do *hardware* do PC.

Encapsulamento metálico	Responsável por proteger e alojar os dispositivos do *hardware* do PC.

Os dois tipos diferentes de *hardware* que compõem o sistema de computador de bordo, estão interligados principalmente via serial. A interface detém controle total sobre as ações do PC, enviando via serial comandos e recebendo dados como resposta à estes comandos. A interface também é responsável por ligar e desligar o PC, atuando diretamente sobre os pinos Pwr_On do PC, bem como, fornecendo tensão a fonte de alimentação, através de relés. A interface também interfere no funcionamento do sistema de som automotivo pois irá ser responsável pelo acionamento do sinal remoto do módulo, liberando a potência do mesmo, quando a função de MP3 estiver sendo utilizada.

A figura 43 ilustra a representação de integração entre os dois tipos de *hardware* empregados.

Figura 43: Representação da integração de *hardware* PC/Interface



7.8 Resultados e discussão

Esta seção apresenta diversas imagens do protótipo, com o intuito de demonstrar o funcionamento integral de todo o sistema de computador de bordo.

A figura 44 ilustra o protótipo da interface montado no *Proto-Board* com a identificação dos principais componentes.

A figura 45 ilustra a montagem do PC fechada e a figura 46, mostra detalhes internos do mesmo.

Já a figura 47 ilustra todo sistema interligado, incluindo *hardware* do PC, *hardware* da interface, cabos e controle remoto.

A figura 48 ilustra a instalação do sensor de velocidade V305, abaixo do sensor de velocidade original de um automóvel VW Gol GIII.

Figura 44: Protótipo da interface microcontrolada

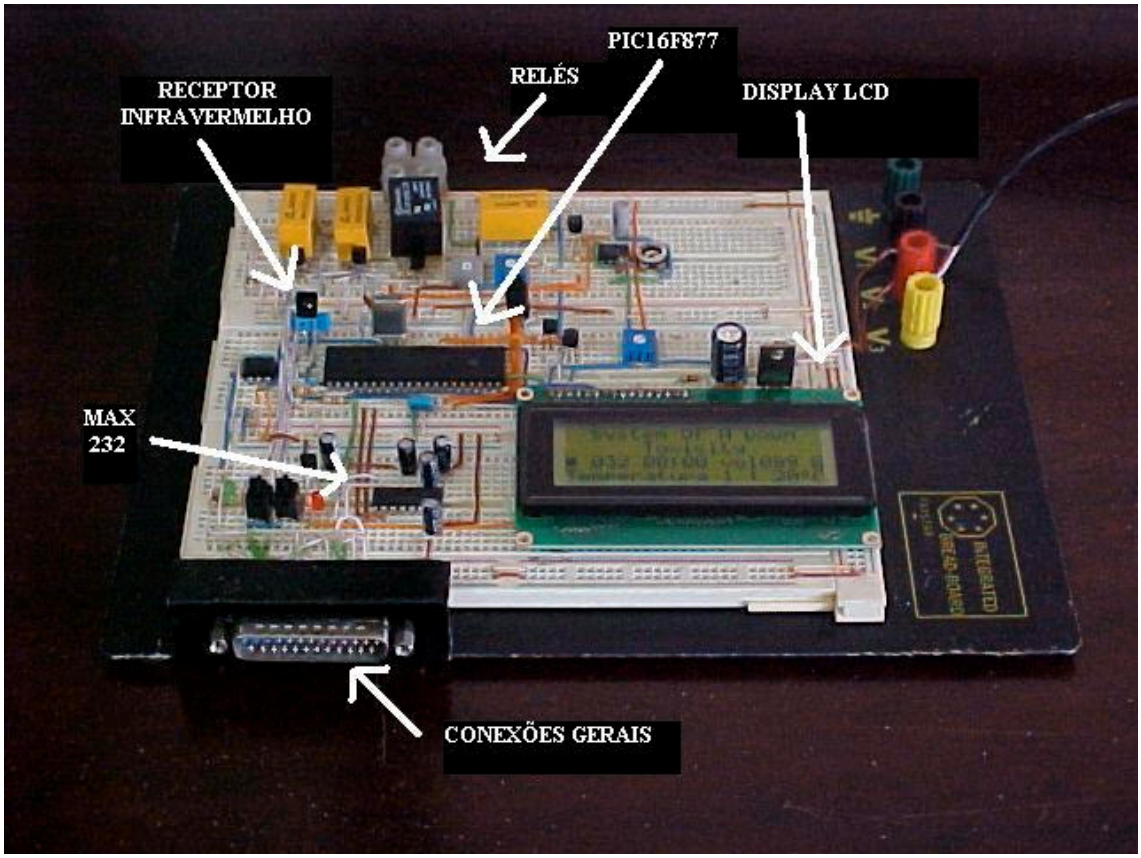


Figura 45: Vista externa do PC

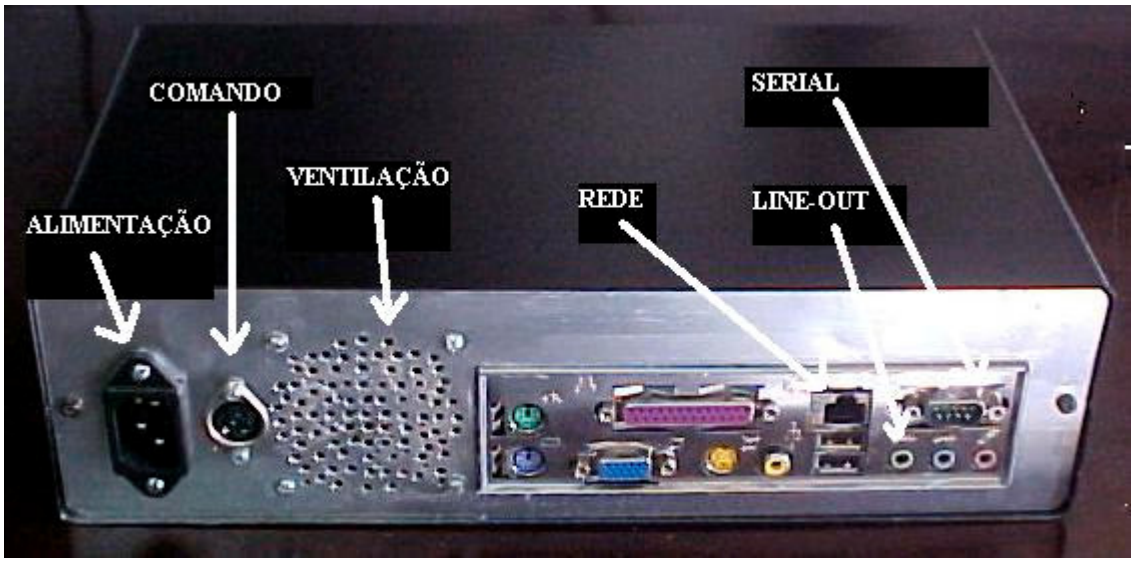


Figura 46: Vista interna do PC

Figura 47: Computador de bordo completo

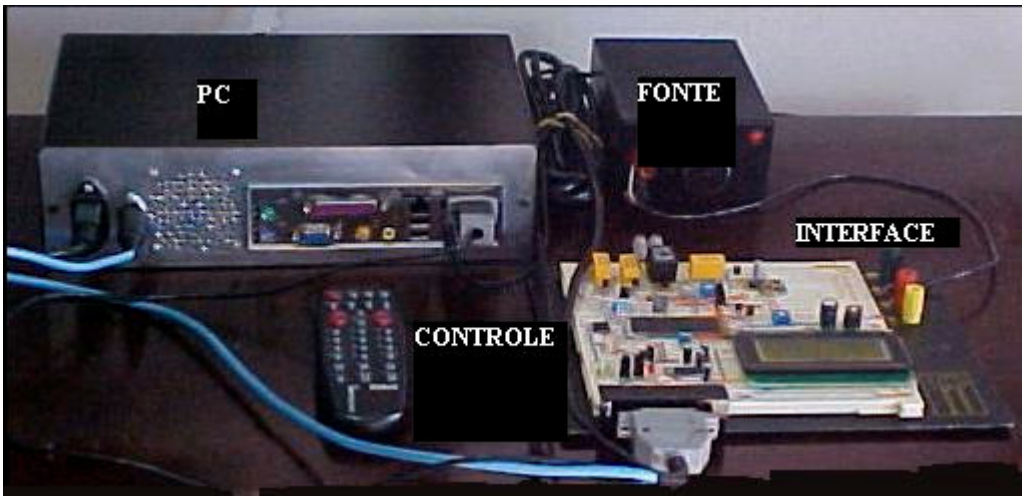
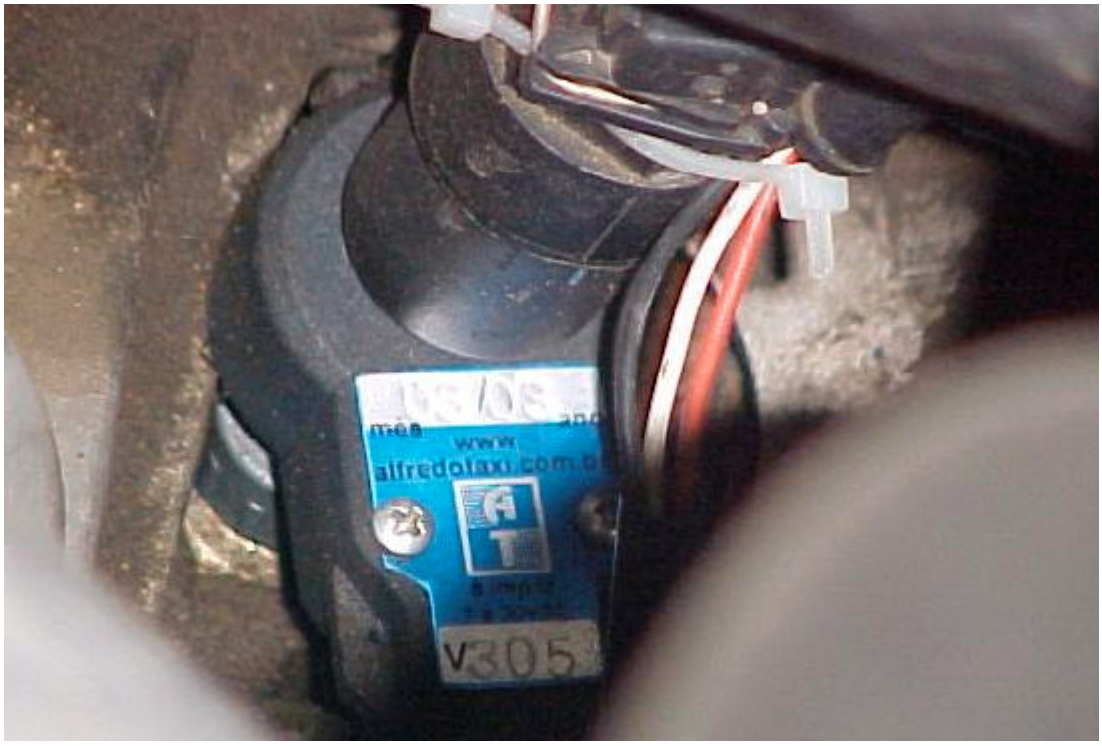


Figura 48: Instalação do sensor de velocidade V305



A seguir são apresentadas todas as telas de operação da interface microcontrolada. Estando o PC desligado, a bateria carregada, e o veículo parado, então serão apenas apresentadas as telas de saudação, vista na figura 49, e as telas de leitura da temperatura 1 e da temperatura 2 em tela cheia, vistas nas figuras 50 e 51 respectivamente. Com o veículo em movimento, é apresentada a tela com a velocidade do mesmo em tela cheia, vista na figura 52.

Figura 49: Tela de saudação – Interface

Figura 50: Tela da temperatura 1 (tela cheia) – Interface

Figura 51: Tela da temperatura 2 (tela cheia) – Interface

Figura 52: Tela da velocidade (tela cheia) – Interface

Quando o usuário do sistema apertar o botão "*On-Off*" o PC será ligado e será apresentada a tela, ilustrada na figura 53. A partir deste momento as 3 linhas do *display* LCD serão ocupadas pela função de MP3, restando a 4ª linha para apresentação temporizada de informações como data, hora, temperaturas, status da bateria, velocidade. Se o botão On-Off for pressionado novamente, então o PC será desligado e será apresentada a tela, ilustrada na figura 54.

Figura 53: Tela de início do PC – Interface

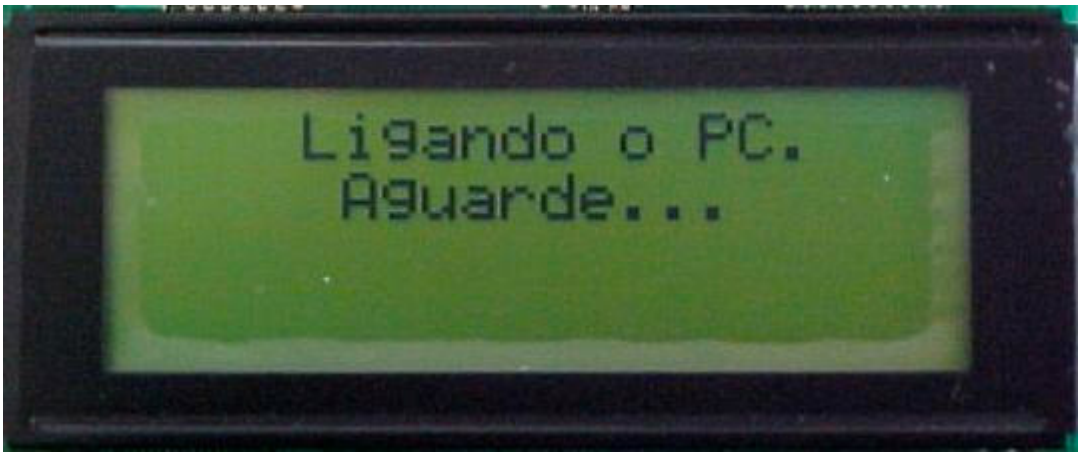


Figura 54: Tela de desligamento do PC – Interface

Quando o PC estiver com o sistema operacional devidamente carregado e o *software* do PC estiver executando, então ele estará apto a receber os comandos para executar as músicas MP3, registrar velocidades, e informar data e hora.

Caso o usuário pressione o botão *Play*, o sistema apresentará a tela ilustrada na figura 55, neste caso apresentando também a data. As figuras 56 e 57 ilustram exemplos com *Play*, apresentando a temperatura 2 e a velocidade.

Figura 55: Tela execução MP3 + Data – Interface

Figura 56: Tela execução MP3 + Velocidade – Interface

Figura 57: Tela execução MP3 + Temperatura 2 – Interface

A figura 58 ilustra uma situação em que uma música está sendo tocada e o veículo excede a velocidade de referência. Neste momento o usuário será alertado e a velocidade será registrada no arquivo “logvel.log” do PC.

Figura 58: Tela de alerta por excesso de velocidade – Interface

Caso o usuário pressione o botão *Stop*, será apresentada uma tela, semelhante á ilustrada na figura 59, apresentando também a temperatura 1. A figura 60 ilustra um exemplo com o botão *Pause* pressionado, apresentando também a hora.

Quando o PC estiver ligado e a bateria atingir a faixa de tensão de 11,4V a 11V será apresentada uma mensagem de alerta na 4ª linha, que ficará fixa até que a bateria seja carregada. Ela é ilustrada na figura 61. Caso a bateria fique abaixo de 11V o PC será automaticamente desligado e será apresentada a tela ilustrada na figura 62.

Figura 59: Tela Stop MP3 + Temperatura 1 – Interface

Figura 60: Tela Pause MP3 + Hora - Interface

Figura 61: Tela de alerta 1 da bateria – Interface

Figura 62: Tela de alerta 2 da bateria – Interface

Após o PC ter sido desligado o sistema ficará bloqueado até que a bateria seja recarregada. A figura 63 apresenta esta tela.

Figura 63: Tela de bloqueio – Bateria descarregada – Interface

O *software* do PC possui três telas distintas, denominadas inicialização, ilustrada na figura 64, execução, ilustrada na figura 65 e encerramento, ilustrada na figura 66. Ele apenas exibe status das operações realizadas e dos dados recebidos e enviados serialmente. A apresentação destas informações de *status* é bastante simples, sendo que, para uma melhor qualidade na apresentação destas telas, tratando-se de linguagem C para Linux, existe a biblioteca *curses*. Como o monitor não é utilizado no sistema, tornou-se dispensável a utilização da biblioteca *curses*.

Figura 64: Tela de inicialização – PC

```
TCC - COMPUTADOR DE BORDO PC-LINUX - INICIALIZANDO


Criando arquivo Mp3.lst ...
Criando vetor de arquivos MP3 ...
Obtendo velocidade de referência para a interface ...
Abrindo o arquivo logvel.log ...
Criando os pipes do processo mpg321 ...
Recebendo status inicial de mpg321 ...
Abrindo a porta serial ...
Aguardando status da interface ... Tentativa 0
Aguardando status da interface ... Tentativa 1
Aguardando status da interface ... Tentativa 2
Aguardando status da interface ... Tentativa 3
Aguardando solicitação de envio de vel. referên ... Tentativa 0
Aguardando solicitação de envio da vel. referên ... Tentativa 1
Aguardando solicitação de envio da vel. referên ... Tentativa 2
Aguardando solicitação de envio da vel. referên ... Tentativa 3

Sistema pronto para operar. Aguardando comandos via serial ...
```

Figura 65: Tela de execução – PC

```
TCC - COMPUTADOR DE BORDO PC-LINUX - EXECUTANDO
```

```
Recebeu - Play
Enviou  - Banda Black Rebel Motorcycle Club
Enviou  - Música Whatever Happened to My Rock a
Enviou  - Qualidade joint stereo
Enviou  - Status Tocando
Enviou  - Tempo 00:00
Enviou  - Faixa 1
Enviou  - Volume 75
Enviou  - Tempo decorrido 00:02
Enviou  - Tempo decorrido 00:02
Enviou  - Tempo decorrido 00:02
Enviou  - Tempo decorrido 00:03
Enviou  - Tempo decorrido 00:03
Enviou  - Tempo decorrido 00:04
Enviou  - Tempo decorrido 00:04
Enviou  - Tempo decorrido 00:05
Enviou  - Tempo decorrido 00:06
Enviou  - Tempo decorrido 00:06
Enviou  - Tempo decorrido 00:07
Recebeu - Next
Recebeu - Play
Enviou  - Banda CPM22
Enviou  - Música Tarde de outubro
Enviou  - Qualidade joint stereo
Enviou  - Status Tocando
Enviou  - Tempo 00:00
Enviou  - Faixa 2
Enviou  - Volume 75
Enviou  - Tempo decorrido 00:01
Enviou  - Tempo decorrido 00:02
Enviou  - Tempo decorrido 00:03
Enviou  - Tempo decorrido 00:03
Recebeu - Pause
Enviou  - Status Parado
Enviou  - Tempo decorrido 00:04
Enviou  - Tempo decorrido 00:04
Enviou  - Tempo decorrido 00:04
Enviou  - Tempo decorrido 00:05
Enviou  - Tempo decorrido 00:05
Enviou  - Tempo decorrido 00:06
Enviou  - Tempo decorrido 00:06
Enviou  - Tempo decorrido 00:06
Recebeu - Stop
Enviou  - Status Parado
Recebeu - Volume -
Enviou  - Volume 74
Recebeu - Volume -
Enviou  - Volume 73
Recebeu - Volume +
Enviou  - Volume 74
Recebeu - Volume +
Enviou  - Volume 75
Enviou  - Data: 02/06/03
Enviou  - Hora: 18:06:35
Recebeu - Velocidade: 120
```

Figura 66: Tela de encerramento - PC

```
TCC - COMPUTADOR DE BORDO PC-LINUX - FINALIZANDO
Finalização normal do programa
Fechando a porta serial ...
O programa será encerrado ...
```

Existem 10 erros diferentes, que podem acarretar no encerramento forçado do programa principal. A tabela 18 apresenta os mesmos.

Tabela 18: Erros de execução no programa principal - PC

Erro	Descrição
0	Finalização normal do programa.
1	Erro na criação dos pipes do processo mpg321.
2	Erro na abertura da porta serial.
3	Erro na recepção do status da interface.
4	Erro na recepção da solicitação de envio da velocidade máxima.
5	Erro na abertura do arquivo Mp3.lst.
6	Erro na abertura do arquivo Velocmax.cfg.
7	Erro na abertura do arquivo logvel.log.
8	Erro na criação do processo mpg321.
9	Erro no status de execução do processo mpg321.

Por exemplo : caso ocorra o erro 2 será apresentada a tela ilustrada na figura 67.

Figura 67: Tela de encerramento com erro – PC

```
TCC – COMPUTADOR DE BORDO PC-LINUX – FINALIZANDO
Erro na recepção do status da interface
Fechando a porta serial ...
O programa será encerrado ...
```

A tabela 19 apresenta os custos para aquisição dos principais componentes do sistema.

Tabela 19: Custos de desenvolvimento do protótipo

Descrição	Custo (R$)
Placa-mãe Mini-ITX EPIA 800	700,00
HD *Slim* 6Gb IBM	240,00
Memória RAM DIMM PC133 256Mb	180,00
Fonte ATX 300W	60,00
Display LCD WH2004-A	70,00
Gravador ProPic 2	250,00
Microcontrolador PIC16F877	30,00
Controle remoto universal Systemlink 4	30,00
Caixa metálica 200x175x60	50,00
Inversor de tensão Hayonick 300W	190,00
Sensor de velocidade	80,00
Demais componentes interface	70,00
Total Geral	**1950,00**

8 Conclusão

O protótipo desenvolvido, apontou para uma nova tendência em equipamentos automotivos, caracterizando um sistema de computador de bordo. Aliando a versatilidade de aplicações embarcadas dos microcontroladores com o alto poder de processamento de um placa-mãe extremamente reduzida, houve a oportunidade de

demonstrar a interação de diversas áreas de conhecimento da computação, citando a comunicação de dados serial, a comunicação de dados infravermelho Norma RC5, o protocolo FTP, a arquitetura de *hardware*, sistema operacional Linux, linguagem C, linguagem PICBASIC entre outros.

O sistema uniu funcionalidades básicas de um computador de bordo, tais como medições de temperatura, velocidade, data e hora, nível da bateria, com um recurso multimídia, já bastante difundido e que vem tornando-se um produto comercial bastante presente na área de equipamentos de som automotivo, o *player* MP3. O sistema propôs também uma funcionalidade bastante interessante para empresas, simulando a ação de um tacógrafo, através do registro de velocidades excessivas.

Atualmente os *players* de MP3 já são comercializados em larga escala e possuem diversos fabricantes, sendo a maioria dos modelos disponíveis, na versão CD, que apesar do preço mais competitivo, torna-se um limitador, no que diz respeito a capacidade de armazenamento, pois CDs com MP3 armazenam em média no máximo 170 músicas. A função MP3 do sistema desenvolvido conta com um HD como fonte de armazenamento das músicas, o que permite o armazenamento de até 1000 músicas, com facilidades muito superiores, no momento de atualização das mesmas, não necessitando de gravadores de CD. Estes *players* MP3 possuem um menor número de modelos e o seu custo é superior aos *players* MP3 versão CD, devido á uma série de vantagens que o CD não proporciona. A utilização de uma placa-mãe para a reprodução de MP3 ao invés de um *hardware* microcontrolado com decodificador MP3, deve-se a dois fatores principalmente: o tamanho extremamente reduzido da placa-mãe utilizada, constituindo um novo padrão de placas-mãe, denominado Mini-ITX, e principalmente pela possibilidade de fácil *upgrade* para novas tecnologias na área de compactação de áudio que tendem a se consolidar no futuro tais, como *MP4*, *Ogg Vorbis* e outras que surgirão em meio ao avanço tecnológico cada vez mais constante e ágil. Existe também a portabilidade e compatibilidade de recursos mínimos de *hardware* necessários para outras diversas funcionalidades.

Em contraste ao PC utilizado, a interface microcontrolada serviu de intermediadora ao funcionamento do PC, fornecendo também funcionalidades descritas acima e automatizando o processo de comando do PC.

A utilização do mecanismo inter-processos do sistema operacional Linux, mostrou-se extremamente versátil e apropriado, reduzindo drasticamente o trabalho de programação de funcionalidades já prontas e bem executadas por programas prontos, como é o caso do aumix, do mpg321 e do MP3INFO.

O custo de aquisição dos componentes do protótipo tornou-se alto, porém o desafio e o nível didático a que o protótipo foi exposto, tornou-o extremamente viável. Sistemas como estes tenderão a se tornar cada vez mais constantes no cotidiano dos motoristas, e com funcionalidades cada vez mais avançadas.

8.1 Sugestões para trabalhos futuros

Entre as sugestões para trabalhos futuros, propõe-se a criação de uma nova funcionalidade, para medição de consumo de combustível a cada abastecimento em tempo real com registro de operações.

Implementação de rotinas gráficas para o limitado *display* LCD padrão HD44738, aumentando o nível de estética da apresentação de dados, juntamente com um equalizador gráfico para o *player* MP3, sem comprometer o custo do produto final.

Implementação de um sistema de direção inteligente, que funcionaria como um detector de saída de pista do veículo, caso por exemplo o motorista pegasse no sono, utilizando o processamento da placa-mãe EPIA Mini-ITX.

Integração de um GPS com o sistema de computador de bordo.

9 Referências bibliográficas

ABRAMCET. **Computador de bordo**, São Paulo, 2003. Apresenta informações sobre novas tendências para os computadores de bordo. Disponível em: <http://www.abramcet.com.br/home/old/hist_conceitos.shtml>. Acesso em: 6 fev. 2003.

ANUNCIACAO, Heverton Silva. **Linux** – guia prático em português**.** São Paulo: Érica, 1999.

BERTOLI, Roberto Angelo. **Software Tango**. São Paulo: Editora do Colégio Técnico de Campinas, 2000.

BRAGA, Newton. Módulos inteligentes LCD multi-matrix. **Saber Eletrônica**, São Paulo, ano 17, n. 201, p. 11-23, set. 1989.

CANTU, Marcos. **Delphi 5.0** – a bíblia. São Paulo: Makron Books, 2000.

CANZIAN, Edmur. Comunicação Serial – **RS232**. São Paulo: Editora da Escola Técnica CNZ de Cotia, 2002.

DANESH, Arman. **Dominando Linux** – a bíblia**.** São Paulo: Makron Books, 2000.

DE MARCO, Tom. **Analise estruturada e especificação de sistema**. Rio de Janeiro: Campus, 1989.

FIP. **Sensores de velocidade**, São Paulo, 2002. Apresenta detalhes de funcionamento e instalação para os sensores de velocidades comercializados pelo mesmo. Disponível em: <http://www.fip.com.br/Sensores/Sensor%20de%20Velocidade.pdf>. Acesso em: 9 mar. 2003.

HENDERSON, Brian. **Make your virtual console log in automatically**, Seattle, 2001. Informações para configuração de login automático no Linux. Disponível em: < http://www.linuxgazette.com/issue69/henderson.html>. Acesso em: 10 abr. 2003.

HITECHMODS. **VIA EPIA-800** Mini-ITX Motherboard Review, Flórida, 2002. Apresenta uma revisão completa sobre a placa-mãe EPIA 800 Mini-ITX. Disponível em: < http://www.hitechmods.com/reviews/motherboards/VIA_EPIA/via_epia.shtml>. Acesso em: 20 out. 2002.

IBIBLIO. **MP3Info**, 2001. Apresenta informações sobre o programa mp3info. Disponível em: <http://www.ibiblio.org/mp3info/>. Acesso em: 10 mai. 2003.

JCONLINE. **Computadores de bordo vs check control**, Recife, 1999. Apresenta particularidades destes dois tipos de sistema disponíveis em veículos. Disponível em: < http://www2.uol.com.br/JC/_1999/1903/vc1403b.htm>. Acesso em: 12 fev. 2003.

KAINKA, Brian. Emissor/receptor IV para PC. **ELEKTOR**, São Paulo, ano 1, n. 7, p. 6-11, out. 2002.

KERNIGHAN, Brian W.; RITCHIE, Dennis M. **C: A linguagem de programação**. Rio de Janeiro: Campus, 1990.

LESLIE, Rob. **MAD : mpeg áudio decoder**, California, 2001. Site oficial da Biblioteca MAD. Disponível em: < http://www.underbit.com/products/mad/>. Acesso em: 20 fev. 2003.

The GCC Team. **GCC Home Page – GNU Project**, Boston, 2003. Site oficial do compilador GCC. Disponível em: < http://gcc.gnu.org/>. Acesso em: 30 mai. 2003.

NATIONAL SEMICONDUTOR. **Analog and interface products databook**. California: National, 2002.

MATTHEW, Neil; STONES, Richard. **Beginning Linux programming**. Birmingham: Wrox, 1996.

MAXIM. **MAX232** product, California, 2003. Dados técnicos sobre este componente. Disponível em: < http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1798/ln/en>. Acesso em 20 nov. 2002.

MELAB MicroEngineering Labs Corporation. **PicBasic** compiler, Califórnia, 2003. Apresenta manuais e bibliotecas existentes para utilização do compilador PicBasic. Disponível em: <http://www.melabs.com/resources/index.htm>. Acesso em: 01 mar. 2003.

MICROCHIP. **PIC16F877** device, Arizona, 2001. Apresenta informações técnicas sobre o microcontrolador PIC16F877 e exemplos de aplicação prática. Disponível em: <http://www.microchip.com/1010/pline/picmicro/category/embctrl/14kbytes/devices/16f877/index.htm>. Acesso em: 10 jan. 2003.

MINI-ITX. **Hardware,** 2002. Apresenta diversas informações sobre o padrão Mini-ITX. Disponível em: <http://www.mini-itx.com/hardware.asp>. Acesso em: 20 nov. 2002.

MITCHELL, Mark; OLDHAM, Jeffrey; SAMUEL, Alex. **Advanced Linux Programming**. Indiana: New Riders, 2001.

MOTORCITY. **A bateria**, Apresenta informações técnicas sobre baterias automotivas. Disponível em: < http://www.geocities.com/MotorCity/Track/7110/Bateria.htm>. Acesso em: 05 mar. 2003.

MPEG ORG. **The best Mp3 resources**, California, 1998. Apresenta informações técnicas sobre MP3. Disponível em: < http://www.mpeg.org/MPEG/mp3.html>. Acesso em: 10 jan. 2003.

OAKES, Gordon. Era Mp3 chega ao automóvel. **Som & Carro**, São Paulo, ano 4, n. 48, p. 20-21, set. 2000.

OLIVEIRA, Luis Antonio Alves de. **Comunicação de dados e teleprocessamento** – uma abordagem básica. 3. ed. São Paulo: Atlas, 1989.

RCA. **RCU404** product, Ronks, 1999. Dados sobre o controle remoto Systemlink 4. Disponível em: < http://www.rca.com/product/viewdetail/0,2588,PI45089,00.html?>. Acesso em: 10 nov. 2002.

REDHAT. **Red Hat Linux 7.3:** The Official Red Hat Linux Customization Guide. Raleigh: Red Hat Inc, 2003.

SCHILDT, Herbert. **C completo e total**. 2. ed. São Paulo: Makron Books, 1990.

SOUZA, David Jose de. **Desbravando o PIC**. 5. ed. São Paulo: Érica, 2002.

SIEMENS. **SFH506**. Siemens Semiconductor Group, 2002.

STEVENS, W. Richard. **Advanced programming in the UNIX environment**. Boston: Addison-Wesley, 1993.

TCHE. **FTP** – File Transfer Protocol, Rio Grande do Sul, 1995. Apresenta informações e operações básicas de FTP. Disponível em: < http://www.tche.br/ftp.html>. Acesso em: 23 mai. 2003.

TORRES, Gabriel. **Hardware** – curso completo. 3. ed. Rio de Janeiro: Axcel Books, 1999.

TORRES, Gabriel **Hardware** – curso completo. 4. ed. Rio de Janeiro: Axcel Books, 2001.

VIA. **EPIA Mini-ITX user's manual**. Newark: Via VPSD, 2002.

VOLKERDING, Patrick. **Linux** – programando para Linux. São Paulo: Makron Books, 1998.

WINSTAR Displays. **WH2004** product, Taipei, 2003. Apresenta dados técnicos do display LCD WH2004. Disponível em: < http://www.winstar.com.tw>. Acesso em: 15 mar. 2003.

WELSH, Matt. **Linux installation and getting started.** Seattle: Specialized Systems Consultants, 1995.