UNICEUB – CENTRO UNIVERSITÁRIO DE BRASÍLIA
FAET – FACULDADE DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Computador de Bordo para Automóveis

Toni Gledson Dantas da Silva

Brasília
2005

Toni Gledson Dantas da Silva

Computador de Bordo para Automóveis

Orientador: Professor D.C. Emilio Carlos Acocella**.**

Monografia apresentada ao Centro Universitário de Brasília, para obtenção do título de Bacharel em Engenharia da Computação.

Brasília-DF
Dezembro de 2005.

AGRADECIMENTOS

A lista de pessoas que contribuíram, de forma direta ou indireta, para a realização deste projeto é muito extensa e é quase impossível referenciar a todos.

Primeiramente não poderia deixar de fora todo o corpo de professores do Cento Universitário de Brasília, que foram os responsáveis por boa parte dos conhecimentos necessários na realização deste projeto, em especial o professor Emilio Carlos Acocella, que na posição de orientador tem compartilhado a glória na transposição de cada dificuldade encontrada.

Outras duas pessoas fundamentais foram os meus pais. Eles ficaram ao meu lado em toda minha vida, desde os primeiros passos acadêmicos até hoje. Isso ajudou na construção de uma base firme e bem sedimentada, não só para o ensino superior, mas também para a vida.

Por fim, mas não menos importante, preciso agradecer a todos os meus colegas de turma, em especial a Ana Paula Silvestre e ao João Paulo Barbosa, amigos na inteireza da palavra.

RESUMO

Este trabalho apresenta um modelo de computador de bordo para veículos. Tal computador automotivo tem como objetivo apresentar no monitor de um computador pessoal os dados coletados por três sensores oriundos do veículo.

Foi desenvolvida uma interface que faz uso de um microcontrolador para receber, digitalizar e tratar os sinais dos sensores: de nível de combustível no reservatório, de velocidade do veículo e de rotação do motor. As informações coletadas são enviadas ao microcomputador através de uma interface serial RS-232. Estas informações são tratadas por um software desenvolvido em Java, para permitir uma interface amigável com o usuário.

Palavras chave: microcontrolador, sensor automotivo, PC, interface serial RS-232.

ABSTRACT

This project has the aim to present a model of computer for vehicles. This computer objective is to show at a personal computer screen the information collected by three sensors located at the vehicle.

A interface using a microcontroller was developed to receive, digitalize and process the signals from the three sensors: one that verifies the level of the fuel in car's tank, one that checks the speed of the vehicle and one that checks the rotation of the engine. The information collected will be sent to microcomputer by a serial interface RS-232. These informations will be processed with a software developed in Java that has a friendly interface for the user.

Key-word: microcontroller, automotive sensor, PC, serial interface RS-232.

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO 11

1.1. CONTEXTUALIZAÇÃO DO PROJETO 11
1.2. OBJETIVO DO PROJETO 13
1.3. MOTIVAÇÃO 13
1.4. ESTRUTURA DO TRABALHO 14

CAPÍTULO 2 - REVISÂO BIBLIOGRÁFICA 16

2.1. COMUNICAÇÃO DE DADOS SERIAL 16
2.2. INTERFACE SERIAL RS-232 (EIA 232) 18
2.3. CONVERSÃO A/D 20

CAPÍTULO 3 - OS SENSORES 23

3.1. SENSOR DE ROTAÇÃO DO MOTOR 23
3.2. SENSOR DE VELOCIDADE 26
3.3. SENSOR DE NÍVEL DE COMBUSTÍVEL 28

CAPÍTULO 4 - A INTERFACE 33

4.1. MÓDULO DE ACOPLAMENTO 33
4.2. MÓDULO DE PROCESSAMENTO 37
4.3. MÓDULO DE TRANSMISSÃO SERIAL 40

CAPÍTULO 5 - O FIRMWARE 43

5.1. FLUXOGRAMA DO FIRMWARE 43
5.2. O DESENVOLVIMENTO DO FIRMWARE 44
5.3. ESTRATÉGIA DO FIRMWARE 44
5.4. A GRAVAÇÃO DO FIRMWARE 45

CAPÍTULO 6 - O SOFTWARE RECEPTOR 47

6.1. LINGUAGEM JAVA E IDE NETBEANS 47
6.2. API JAVA COMMUNICATIONS 47
6.3. DESENVOLVIMENTO DO SOFTWARE RECEPTOR 48
6.4. O TRATAMENTO DOS DADOS 48

CAPÍTULO 7 - CONSIDERAÇÕES FINAIS 51

7.1. DIFICULDADES ENCONTRADAS 51
7.2. RESULTADOS OBTIDOS 52
7.3. CONCLUSÕES 52
7.4. SUGESTÕES PARA TRABALHOS FUTUROS 52
REFERÊNCIAS BIBLIOGRÁFICAS 54
ANEXO A – CÓDIGO FONTE DO SOFTWARE DO MICROCONTROLADOR 55
ANEXO B – CÓDIGO FONTE DO SOFTWARE RECEPTOR 58
ANEXO C – DIAGRAMA ELÉTRICO COMPLETO DA INTERFACE 64
ANEXO D – ESQUEMA ELÉTRICO DE LIGAÇÃO DA CIE DO VECTRA GLS 2.2 65

LISTA DE SIMBOLOS

A/D – Conversor Analógico Digital
AMPOP – Amplificador Operacional
API - Application Program Interface
CI – Circuito Integrado
CIE – Central de Injeção Eletrônica
CPE- Data Circit-terminating
ddp – Diferença de Potencial
DTE - Data Terminal Equipment
EEPROM - Electrically Erasable Programmable Read Only Memory
EIA - Eletronic Industries Association
GEIA - Grupo Executivo da Indústria Automobilística
GND – Ground
IDE – Integrated Development Environment
MCU - Micro Controler Unit
PAM – Pulse Amplitude Modulation
PC – Computador Pessoal
PCM – Pulse Code Modulation
PIC – Programmable Intelligent Computer
RPM – Rotação Por Minuto
RS - Recommended Standard
SCI - Interface de Comunicação Serial
TTL - Transistor Transistor Logic
UART - Universal Asynchronous Receiver and Transmitter
USART - Universal Synchronous Asynchronous Receiver Transmitter

ÍNDICE DE FIGURAS

Figura 1.1 - Modelo do Projeto 13
Figura 2.1 – Exemplo de Transmissão Serial Assíncrona 18
Figura 2.2 – Pinagem do DTE e DCE 19
Figura 2.3 – Níveis de Tensão para RS-232 19
Figura 2.4 – Processo de Amostragem 21
Figura 2.5 – Processo de Quantização e Codificação 22
Figura 3.1 - Sensor de Rotação 24
Figura 3.2 – Esquema do Sensor de Rotação 25
Figura 3.3 - Esquema do Sensor de Velocidade 27
Figura 3.4 - Sensor de Nível de Combustível 29
Figura 3.5 – Esquema do Sensor de Nível 30
Figura 3.6 – Gráfico de Litros por ddp 31
Figura 4.1 – Diagrama de Blocos da Interface 33
Figura 4.2 – Modelo de Utilização do AMPO 34
Figura 4.3 – Circuito elétrico do acoplador dos Sinais Discretos 36
Figura 4.4 - Circuito elétrico do acoplador do Sinal Analógico 36
Figura 4.5 – Diagrama do LM 324N 37
Figura 4.6 – PIC 16F877A 38
Figura 4.7 – Esquema de ligação do cristal 39
Figura 4.8 – Esquema de ligação do MCLR (Microchip) 40
Figura 4.9 – Diagrama Esquemático do CI MAX 232 41
Figura 5.1 - Fluxograma do Firmware 43
Figura 5.2 – MC PLUS 46
Figura 6.1 - Tela Para Seleção da Porta 48
Figura 6.2 – Tela de Exibição dos Dados 48

ÍNDICE DE TABELAS

Tabela 3.1 - Relação entre rpm e freqüência 26
Tabela 3.2 - Relação entre velocidade e freqüência 28
Tabela 3.3 - Diferença de Potencial por Litro 30
Tabela 3.4 - Diferença entre o Mostrador e o Calculado em Litros 32
Tabela 4.1 - Valores de Capacitores por Freqüência de Trabalho 39

CAPÍTULO 1 - INTRODUÇÃO

1.1. Contextualização do Projeto

A indústria brasileira de automóveis teve seu início apenas em 1957. O primeiro veículo automotor transitou no Brasil em 1893, na cidade de São Paulo. Tratava-se de um Daimler europeu, cujo proprietário era Henrique Santos Dumond. Apenas 21 anos depois, já havia no País algo em torno de 86 veículos. [Arquivo do carro nacional,99]

A primeira montadora a se instalar no Brasil foi a Ford Motor Co, em 1919. Com a citada montadora, os preços dos automóveis comercializados no Brasil caíram para apenas um terço dos preços cobrados anteriormente. Com o sucesso da Ford, as demais montadoras seguiram os seus passos, sendo a General Motors do Brasil a primeira delas, instalando-se no Brasil em 1925.

Em 1952, a indústria automotiva teve sua primeira grande guinada. Neste ano, a importação de veículos atingiu números elevados, o que impulsionou a criação de uma subcomissão de veículos automotores. Em 1953, ocorreu a proibição de importação de veículos completos e montados, além da criação do GEIA (Grupo Executivo da Indústria Automobilística). No fim dos anos 60, a indústria nacional estava consolidada, com 320.680 veículos produzidos para uma população de 66.755.000 brasileiros. Já na década de 70, o Brasil possuía uma população de 93.139.000 habitantes e uma produção acumulada de 2.284.054 veículos. Na década de 80, a produção nacional de automóveis variou entre 780.841 e 1.165.174 unidades ao ano. Apenas em 1997 o Brasil

superou a marca de 2.000.000 de unidades fabricadas por ano, registrando um total de 2.069.273 unidades. [ANFAVEA, 2005]

O Brasil detinha em 2002 a 9ª maior frota de veículos automotores do mundo, com 20,8 milhões de unidades. No ano de 2003 o Brasil ocupava a 10° posição no ranking mundial de produção de veículos automotores, com 1.828 milhões de unidades fabricadas por ano. Na atualidade a produção brasileira anual é de 2.210.741 unidades. Na década de 90 se deu o apogeu da indústria automobilística nacional.

No ano de 1989, inicio do apogeu da indústria automobilística brasileira, a Volkswagen lançou o Gol GTI, o primeiro veículo nacional equipado com injeção eletrônica de combustível.

O sistema de injeção eletrônica de combustível foi desenvolvido com a finalidade de substituir o sistema de admissão carburada, melhorando principalmente a potência e o torque do veículo. No sistema de injeção eletrônica, todo o controle de admissão do veículo é feito por uma central microcontrolada que, com base em dados disponibilizados por sensores elétricos / eletrônicos, atua diretamente na injeção de combustível.

Todos os veículos comercializados atualmente são dotados de injeção eletrônica de combustível e, por conseqüência, de uma série de sensores elétricos / eletrônicos, que são utilizados também para alimentar o painel de instrumentos do veículo.

1.2. Objetivo do Projeto

Este projeto foi desenvolvido para atender requisitos acadêmicos. Pode ser aprimorado e, assim, prestar-se para aplicações comerciais.

O primeiro objetivo do projeto é produzir uma interface que se propõe a coletar e tratar - entenda-se digitalizar e multiplexar - os dados fornecidos pelos sensores de velocidade do veículo (odômetro); rotações por minuto do motor (tacômetro) e o nível do reservatório de combustível (bóia). Depois de tratar os sinais recebidos, a interface disponibiliza os dados em uma porta serial padrão RS-232. Os dados são então exibidos na tela de um PC em tempo real, por intermédio de um software de comunicação desenvolvido neste projeto.

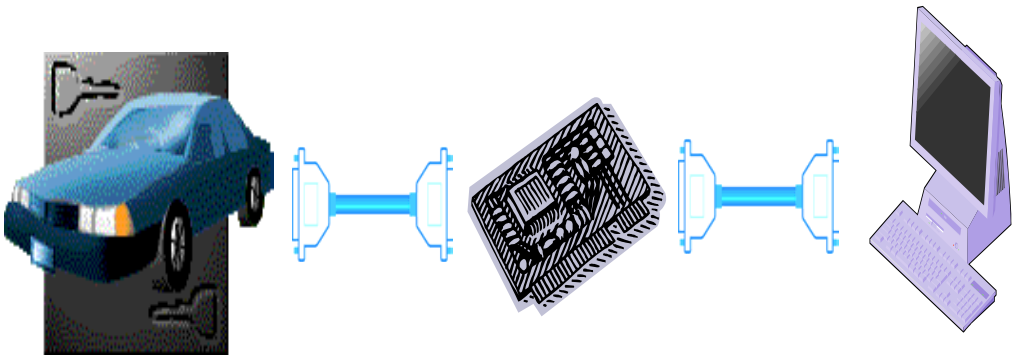


Figura 1.1 - Modelo do Projeto

1.3. Motivação

Devido às grandes conquistas no automobilismo acumuladas por vários pilotos brasileiros, dentre os quais se destaca Ayrton Senna, o Brasil foi conquistado por uma outra paixão: automóveis. Em conseqüência disto pode-se notar que uma grande parte da população brasileira não vislumbra um

automóvel apenas como meio de condução, mas também como um bem de consumo. Para este público, conforto não é algo opcional. Desta forma, acessórios modernos, que disponibilizem um maior conforto, têm grande aceitabilidade no mercado.

A primeira motivação deste projeto é a melhoria da forma de exibição dos dados apresentados no painel de instrumentos. Na grande maioria dos veículos modernos, todo o painel de instrumentos é analógico, com poucas divisões por unidade, o que implica valores imprecisos.

Outra motivação é a aplicação de conhecimentos de diversas disciplinas do curso de Engenharia de Computação. Dado o seu escopo muito abrangente, o curso Engenharia de Computação é um dos poucos que capacitam para elaboração de projetos que envolvem programação tanto de baixo nível como também de alto nível, além de uma grande aplicação de Eletrônica.

Finalmente, busca – se ainda desenvolver um projeto que possibilite várias opções de aprimoramento e, desta forma, fundamentar futuros projetos.

1.4. Estrutura do Trabalho

A Introdução deste trabalho tem como objetivo apresentar uma breve explanação do projeto.

O Capitulo 2 apresenta uma revisão bibliográfica abordando algumas das tecnologias utilizadas no projeto.

No Capítulo 3 foram apresentados todos os detalhes dos sensores interligados à interface.

O Capítulo 4 descreve todo o hardware desenvolvido no projeto.

No capitulo 5 foi descrito o software do microcontrolador, apresentando todas as funcionalidades e disposições utilizadas no projeto.

O Capítulo 6 foi destinado à apresentação e detalhamento do software receptor que terá a funcionalidade de ler os dados na porta serial do PC, e exibi-los de forma amigável.

Por fim, a Conclusão traz as considerações finais sobre o trabalho, as principais contribuições, os resultados obtidos, as dificuldades encontradas e as sugestões para trabalhos futuros.

CAPÍTULO 2 - REVISÂO BIBLIOGRÁFICA

Considerando o fato de que o projeto abordado nesta monografia não é uma evolução de projeto anterior, mas sim uma nova proposta que possibilitará diversas opções de projetos futuros, esta revisão bibliográfica abordará apenas as tecnologias utilizadas no projeto.

2.1. Comunicação de Dados Serial

Pode-se considerar como transmissão de dados o trânsito de informações digitais (bits) entre dois dispositivos. Estes dispositivos podem estar a poucos centímetros ou longas distâncias. Quanto maior a distância maior será a dificuldade na transmissão. [Silveira, 91]

A taxa máxima de transmissão de um sinal digital é diretamente proporcional à potência do sinal e inversamente proporcional ao ruído. O principal objetivo consiste em transmitir dados a maiores distâncias com menores potências e taxas de ruído reduzidas.

Um canal de comunicação pode ser definido como um meio físico por onde os dados transitam. Um simples fio metálico pode se constituir num canal de comunicação.

Os canais de comunicação são divididos em três tipos, de acordo com o sentido de transmissão. No caso de canais ‘simplex’, a transmissão só é possível em um único sentido. Para os canais ‘halfduplex’ a transmissão pode ser feita nos dois sentidos, porém com a limitação de um sentido por vez. Por

fim, tem-se o canal ‘fullduplex’ que permite a transmissão em ambos os sentidos simultaneamente.

Em uma transmissão de dados é possível a utilização apenas de um canal de comunicação ou vários canais de comunicação simultaneamente. A quantidade de canais utilizados caracteriza a transmissão como serial, quando é utilizado apenas um canal, ou paralela quando se utiliza dois ou mais canais de comunicação.

Na transmissão serial os dados são transmitidos um bit por vez e em seqüência. A transmissão serial pode ser dividida em duas subcategorias, de acordo como é feita a marcação de tempo entre o transmissor e o receptor. Para que o dado enviado pelo transmissor seja corretamente identificado pelo receptor, é necessário que o transmissor e o receptor estejam sincronizados. Para tal sincronização, na primeira técnica empregada, conhecida como Transmissão Serial Síncrona, transmite-se o sinal de relógio do transmissor para o receptor; a segunda técnica, denominada Transmissão Serial Assíncrona, consiste em inserir bits para indicar o começo e o término da seqüência transmitida.

Para a transmissão serial assíncrona é utilizado apenas um canal para a transmissão dos dados e dos bits de sincronização. Isto é feito acrescentando um bit de começo *start bit* e um bit de parada *stop bit*; o canal de comunicação fica em nível alto até um *start bit* ser transmitido, forçando o valor no canal para nível baixo; após este bit segue a informação, com um tamanho de palavra pré-configurado no receptor e no transmissor.

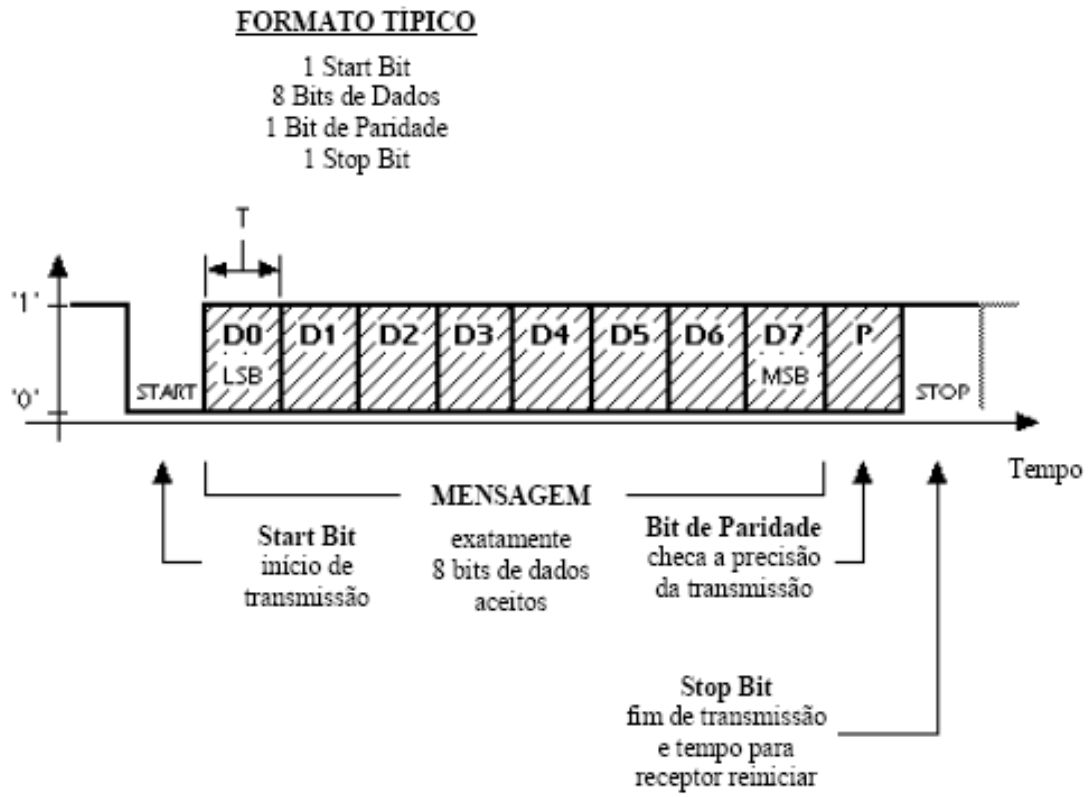


Figura 2.1 – Exemplo de Transmissão Serial Assíncrona

2.2. Interface Serial RS-232 (EIA 232)

O RS-232 é um padrão de comunicação serial, definido pelo comitê "Eletronic Industries Association" (EIA). RS é a abreviatura de "Recommended Standard". Este padrão define: tensões, temporizações, funções dos sinais, um protocolo para troca de informações e as conexões mecânicas.

As principais definições que devem ser observadas no RS 232 são a pinagem dos conectores e as tensões de transmissão. De acordo com o RS 232, em uma transmissão teremos no mínimo um DTE (Data Terminal Equipment – PC) e um DCE (Data Circit-terminating – Modem). A pinagem do DTE é diferente do DCE.

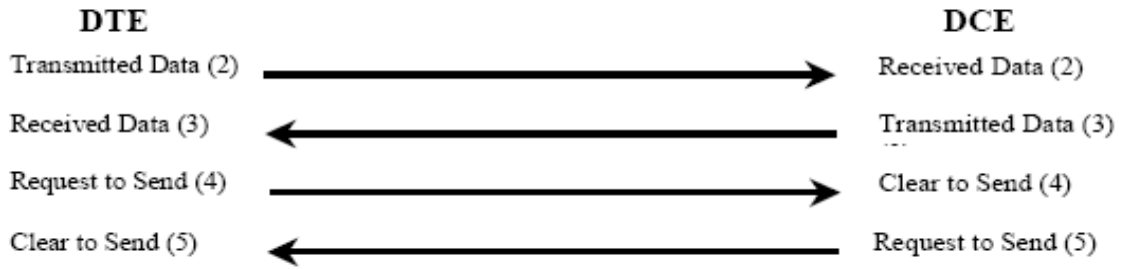


Figura 2.2 – Pinagem do DTE e DCE

O segundo aspecto importante definido pelo padrão são as tensões dos bits no canal de comunicação. Por se tratar de transmissão digital, os valores são binários (0 ou 1). Para o nível lógico alto foram definidas tensões entre -3 a -25 Volts, e para o nível baixo as tensões são de +3 a +25 Volts. As tensões de -2,9 a +2,9 Volts definem uma zona de indeterminação.

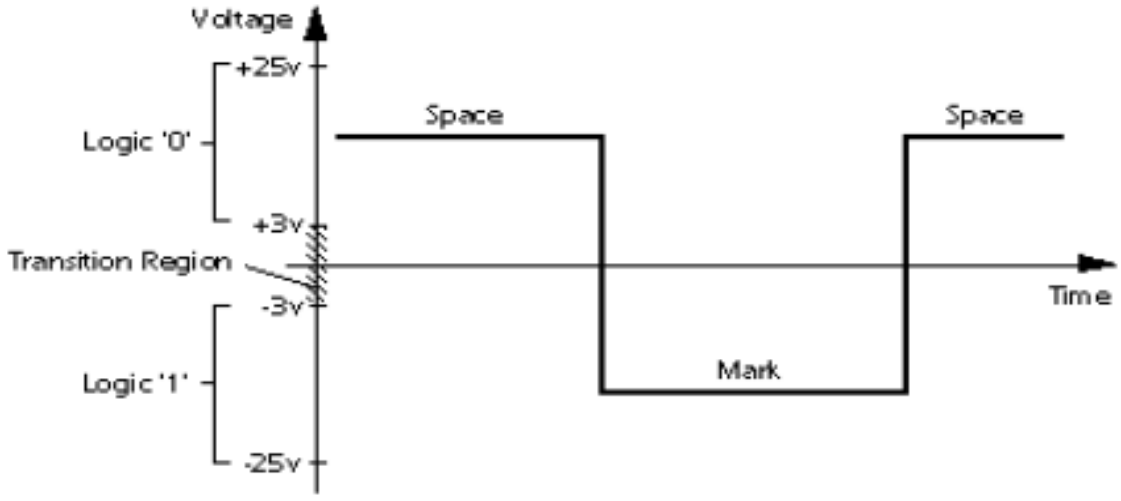


Figura 2.3 – Níveis de Tensão para RS-232

2.3. Conversão A/D

Informações sobre a natureza de um fenômeno físico podem ser veiculadas por intermédio de sinais. Na natureza estes sinais são encontrados na forma analógica.

Os olhos transformam sinais luminosos em sinais elétricos e os enviam para o cérebro. Da mesma forma análoga, os ouvidos transformam sinais sonoros em elétricos, possibilitando ao cérebro analisar algumas características desses sinais como freqüência e amplitude.

Os sinais que descrevem eventos físicos são tidos em sua forma primária como sinais contínuos ou analógicos, pois podem assumir qualquer valor de amplitude ao longo do tempo.

Para os sinais discretos tem-se variações discretas de amplitude dentro de faixas estipuladas. Um caso específico de sinais discretos são os sinais digitais, que além de possuir variações discretas de amplitude, também possuem variações discretas no tempo.

É possível se converter um sinal tanto de analógico para digital, como de digital para contínuo, desde que observadas certas restrições. No caso da conversão de analógico para digital, este processo recebe o nome de digitalização.

O processo de digitalização começa com uma etapa de amostragem, durante a qual são coletadas diversas amostras do sinal ao longo do tempo, conforme mostrado na Fig. 2-4. A taxa de amostragem deve ser no mínimo o

dobro da máxima freqüência do sinal analógico, de acordo com o Teorema de Nyquist.

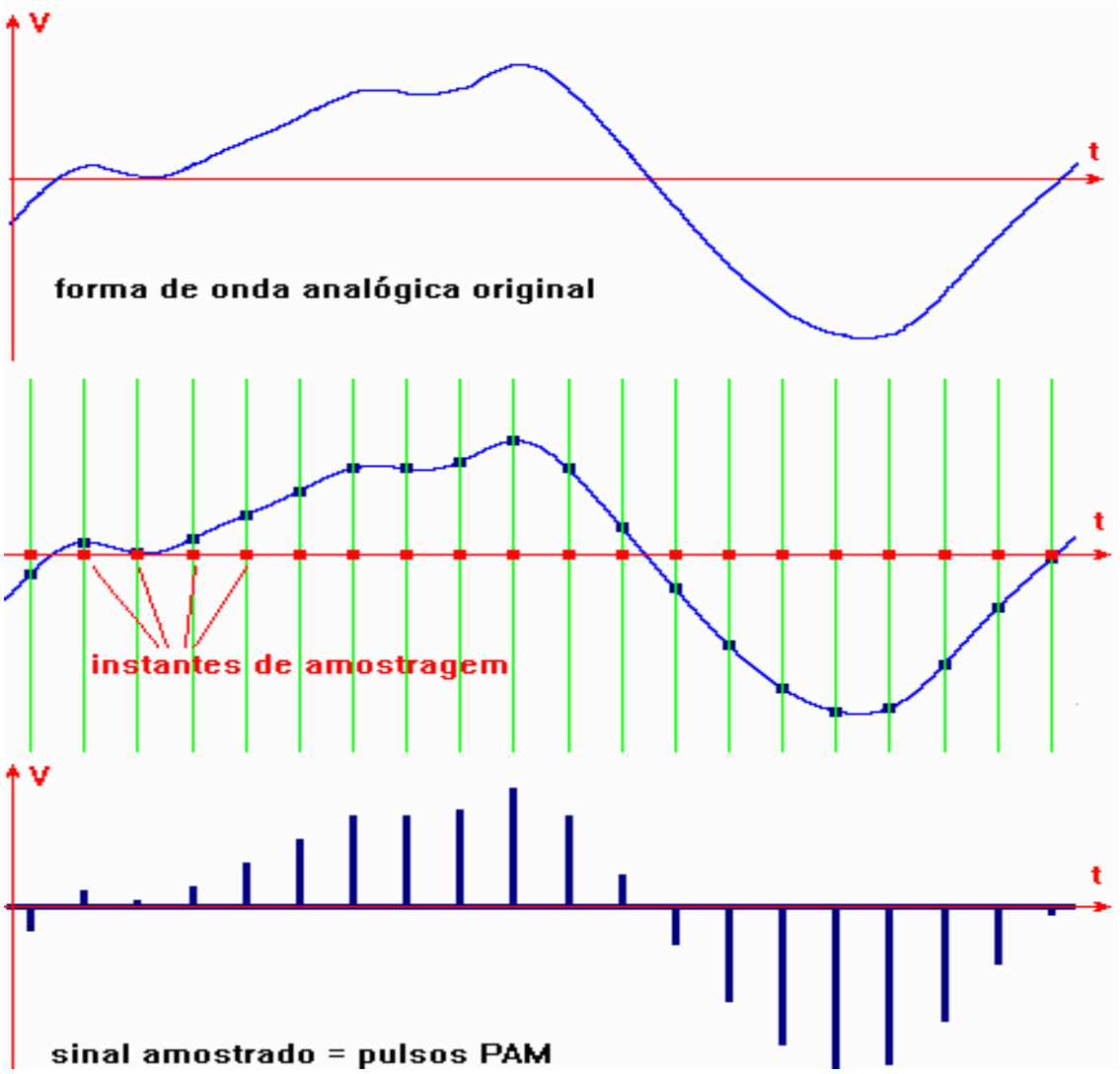


Figura 2.4 – Processo de Amostragem

A quantização é a segunda etapa do processo de digitalização, na qual são definidas faixas em que cada amostra deve ser enquadrada de acordo com o valor de sua amplitude; quanto maior for o número de faixas utilizadas na quantização, maior será a precisão do processo.

Após a etapa de quantização, realiza-se a codificação, quando o sinal quantizado é associado a uma seqüência binária que o representa. As etapas de quantização e codificação estão exemplificadas na Fig. 2-5.

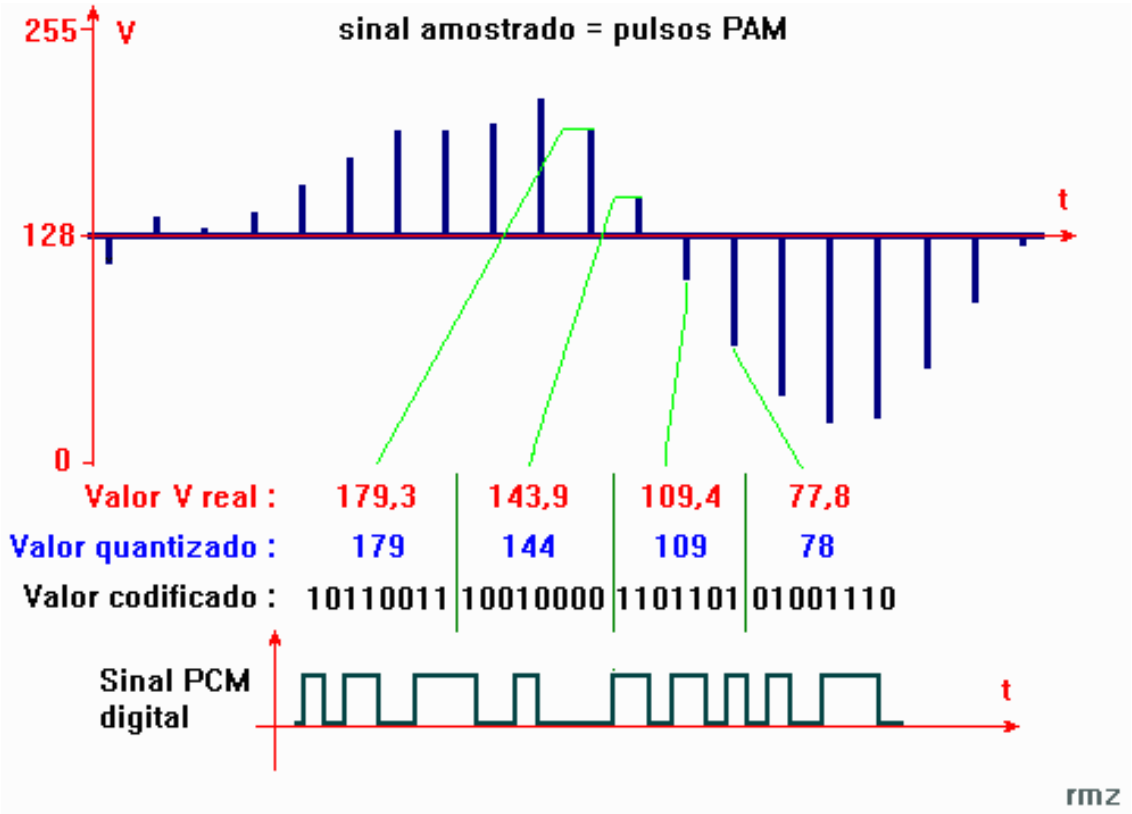


Figura 2.5 – Processo de Quantização e Codificação

CAPÍTULO 3 - OS SENSORES

A interface construída está destinada a capturar os sinais provenientes de três sensores já instalados no veículo. Estes sensores são:

- Número de rotações por minuto do motor (tacômetro);
- Velocidade do veículo (velocímetro); e
- Nível de combustível no reservatório (bóia).

Para elaboração do projeto da interface, o primeiro passo é definir as características destes sinais. Esta foi uma tarefa complexa considerando a escassez de documentação dos componentes automotivos. Assim, todos os parâmetros de comportamento destes sinais foram levantados diretamente do veículo em funcionamento, utilizando - se para isso o Multímetro modelo ET-2055 do fabricante Minipa.

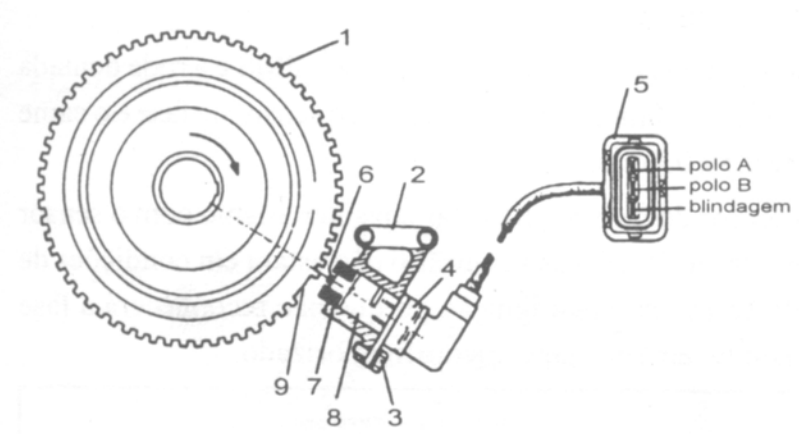
Considerando que não existe uma padronização para os sinais gerados pelos sensores automotivos, estes sinais variam de montadora para montadora e até mesmo entre modelos de uma mesma montadora.

No desenvolvimento deste trabalho foi adotado como referencial um Vectra 2.2, modelo GLS, da montadora General Motors fabricado no ano de 2000.

3.1. Sensor de Rotação do Motor

O sensor de rotação é instalado junto à “correia dentada” do motor do veículo e é do tipo eletromagnético, composto basicamente por um indutor e

um ímã permanente. A Fig. 3.1 ilustra os componentes do sensor de rotação do motor. [Ribeiro, 99]



1 - Roda dentada *2 - Suporte do sensor*
3 - Parafuso *4 - Sensor de rotação*
5 - Conector *6 - Núcleo de ferro*
7 - Indutor *8 - Imã permanente*
9 - Referência de fasagem

Figura 3.1 - Sensor de Rotação

O processo de medição do número de rotações do motor é realizado por intermédio da indução de uma corrente alternada pelo imã permanente e detectada pelo indutor. O número de rotações é diretamente proporcional a esta corrente induzida.

O sinal é então transmitido diretamente à Central de Injeção Eletrônica (CIE) através dos seus pinos 48 e 49. A CIE então processa e trata o sinal AC do sensor de rotação e, posteriormente, o disponibiliza no seu pino 43, já na forma de um sinal discreto variável com amplitudes de 0 ou 12 Volts. A freqüência deste sinal discreto é proporcional à rotação do motor.

O tacômetro do painel de instrumentos possui um mostrador do tipo analógico com escala graduada em intervalos de 1.000 rpm, limite mínimo de 0 rpm e limite máximo de 7.000 rpm. Este mostrador é ligado diretamente ao pino 43 da CIE.

Para o projeto foram coletadas 10 amostras diretamente do pino 43 da CIE. O painel de instrumentos do veículo foi utilizado como referência. Desta forma foi possível definir uma relação entre a freqüência e a rotação do motor.

Para uma melhor precisão as amostras foram coletadas a cada 500 Hz dentro de uma faixa de 1.500 Hz a 6.000 Hz e, por intermédio da média ponderada das diferenças das medições, foi possível definir o valor de 30,3 Hz para cada rpm, conforme a Tabela 3.1.

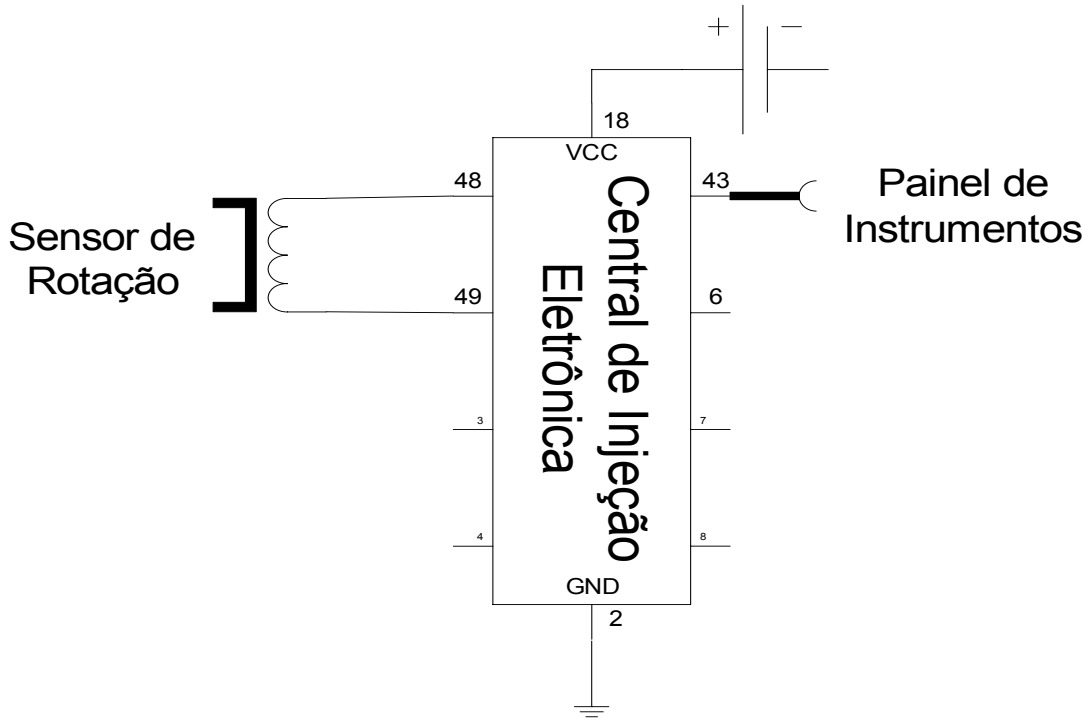


Figura 3.2 – Esquema do Sensor de Rotação

Tabela 3.1 - Relação entre rpm e freqüência

RPM	*Hz*	*Diferença em Hz*
1500	50	18
2000	68	17
2500	85	15
3000	100	20
3500	120	13
4000	133	17
4500	150	15
5000	165	15
5500	180	20
6000	200	
Média p/ 500 rpm		16,50 Hz
Média p/ 1Hz		30,30 rpm

3.2. Sensor de Velocidade

O sensor de velocidade opera de forma semelhante ao de rotação e também é do tipo eletromagnético, produzindo uma corrente alternada induzida.

Este sensor é instalado junto à roda dianteira esquerda do veículo e está conectado a um módulo exclusivo para o tratamento do sinal coletado. Este módulo é denominado de “Módulo do Velocímetro”.

O “Módulo do Velocímetro” possui cinco pinos: dois são destinados à alimentação do módulo (GND e VCC); outros dois são conectados ao sensor de velocidade (*IN* e *IN*); e o quinto pino (*OUT*) é destinado à ligação do

velocímetro, localizado no painel de instrumentos. No pino de *OUT* é disponibilizado um sinal discreto com amplitude de 12 Volts.

Considerando a semelhança entre os sinais de rotação e velocidade, o mesmo processo de amostragem elaborado para o sensor de rotação foi adotado para o sensor de velocidade.

O mostrador do painel de instrumentos também é do tipo analógico e possui divisões de 10 km/h, limite mínimo de 0 km/h e limite máximo de 220 km/h.

Foram coletadas 15 amostras a cada 10 km/h em uma faixa de 10 km/h a 150 km/h, diretamente no pino *OUT* do "Módulo do Velocímetro". A média ponderada das diferenças das medições foi de 4,23 Hz para cada quilômetro, conforme a Tabela 3.2.

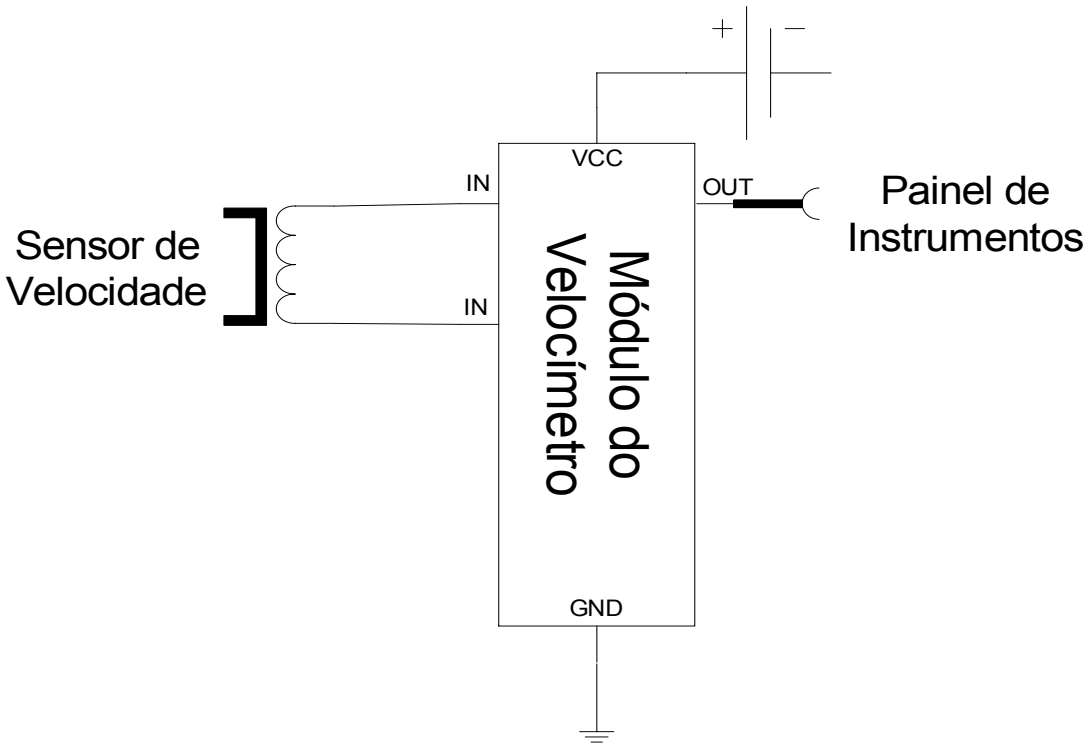


Figura 3.3 - Esquema do Sensor de Velocidade

Tabela 3.2 - Relação entre velocidade e freqüência

km/h	*Hz*	*Diferença em Hz*
10	39	35
20	74	49
30	123	41
40	164	43
50	207	40
60	247	48
70	295	43
80	338	41
90	379	39
100	418	42
110	460	45
120	505	44
130	549	43
140	592	39
150	631	
Média para 10 km/h	42,29 Hz	
Para cada km/h 4,23Hz		

3.3. Sensor de Nível de Combustível

Diferentemente dos outros sensores utilizados, o sensor de nível de combustível não é discretizado por nenhum dispositivo do veículo. Este sensor é do tipo resistivo.

O medidor de nível (bóia) está ligado a um resistor variável (potenciômetro) que possui dois terminais. Um dos terminais do resistor variável é conectado diretamente ao terminal terra (GND) e o outro ao

mostrador no painel de instrumentos. A variação neste sensor provoca uma variação na diferença de potencial (ddp) entre os terminais.

Figura 3.4 - Sensor de Nível de Combustível

O mostrador de nível de combustível no painel de instrumentos apresenta valores em litros com divisões a cada 10 litros, com limite mínimo de 0 litros e limite máximo de 60 litros. O valor de 60 litros corresponde à capacidade máxima do reservatório de combustível.

Para este sensor foram coletadas apenas cinco amostras de diferença de potencial, considerando a variação linear e a precisão das amostras de máximo e mínimo, a Tabela 3.3 apresenta os valores coletados.

Para a coleta das amostras o sensor foi retirado do reservatório de combustível e sua posição foi alterada manualmente, sendo acompanhada por um voltímetro.

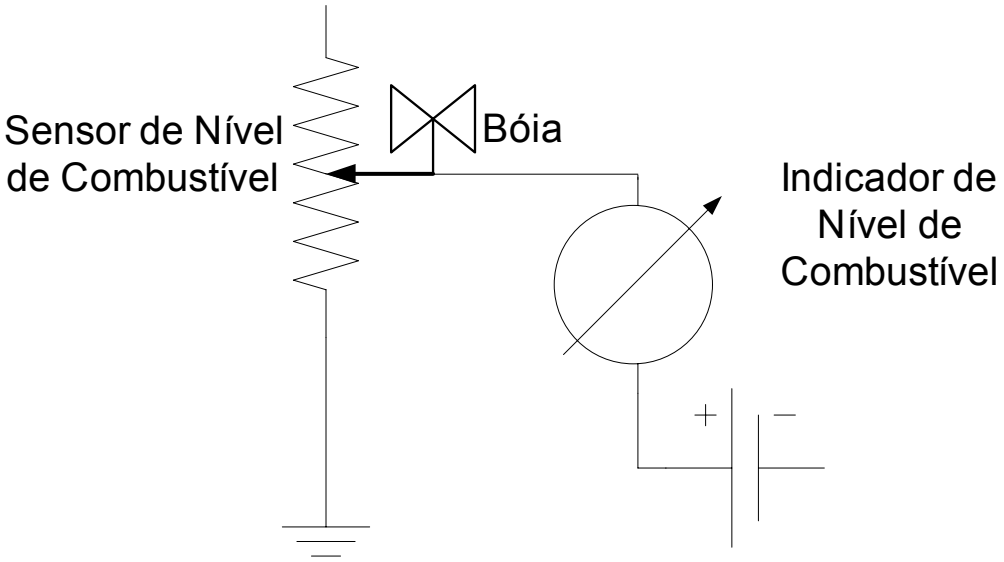


Figura 3.5 – Esquema do Sensor de Nível

Tabela 3.3 - Diferença de Potencial por Litro

ddp (V)	*Litros*
6,48	0
4,9	20
4,05	30
3	45
1,64	60

Os valores de ddp medidos foram referenciados com os valores em litros medidos pelo mostrador localizado no painel de instrumentos do veículo.

As amostras de máximo e mínimo são as mais precisas considerando que o valor mínimo de 0 litros corresponde ao reservatório vazio e o valor máximo de 60 litros corresponde à capacidade do reservatório.

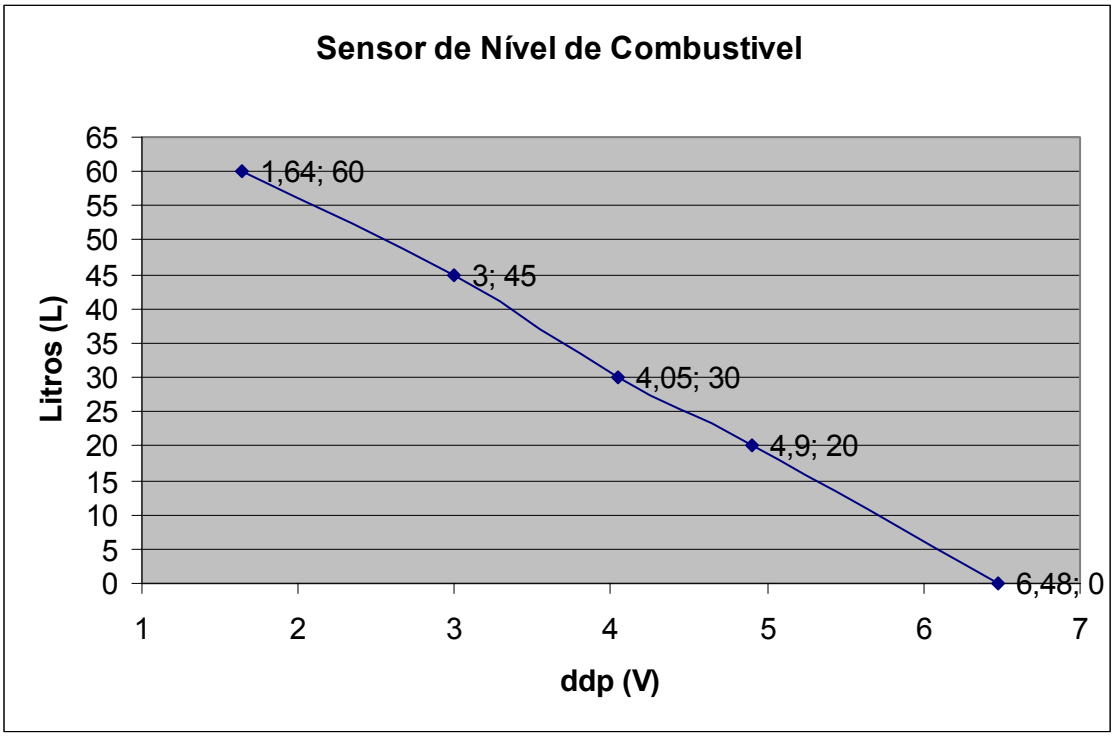


Figura 3.6 – Gráfico de Litros por ddp

Para uma maior precisão foi utilizada no equacionamento a fórmula de "Equação da Reta" tomando os pontos de máximo e mínimo como referência. Desta forma foi obtida a equação 3.1; a qual representa a quantidade de combustível em função da tensão.

$$L = \left(-12,3967 \times ddp\right) + 80,331 \qquad \textbf{(Equação 3.1)}$$

onde:

- **L** é o volume de combustível em litros; e

- **ddp** é o valor da diferença de potencial entre os terminais do sensor de combustível.

A Tabela 3.4 apresenta as diferenças em entre os valores lidos no mostrador do painel e os calculados pela Equação 3.1.

Tabela 3.4 - Diferença entre o Mostrador e o Calculado em Litros

ddp (V)	*Mostrador Litros*	*Calculado Litros*	*Diferença Litros*
6,48	0	0,00	0,00
4,9	20	19,59	0,41
4,05	30	30,12	- 0,12
3	45	43,14	1,86
1,64	60	60,00	0,00

CAPÍTULO 4 - A Interface

A Interface foi construída com o objetivo de coletar os sinais dos três sensores, processá-los e disponibilizar informações a respeito de cada entrada na porta serial padrão RS232.

A Interface foi dividida em três módulos, com funcionalidades distintas. A Fig. 4.1 apresenta um diagrama de blocos do projeto.

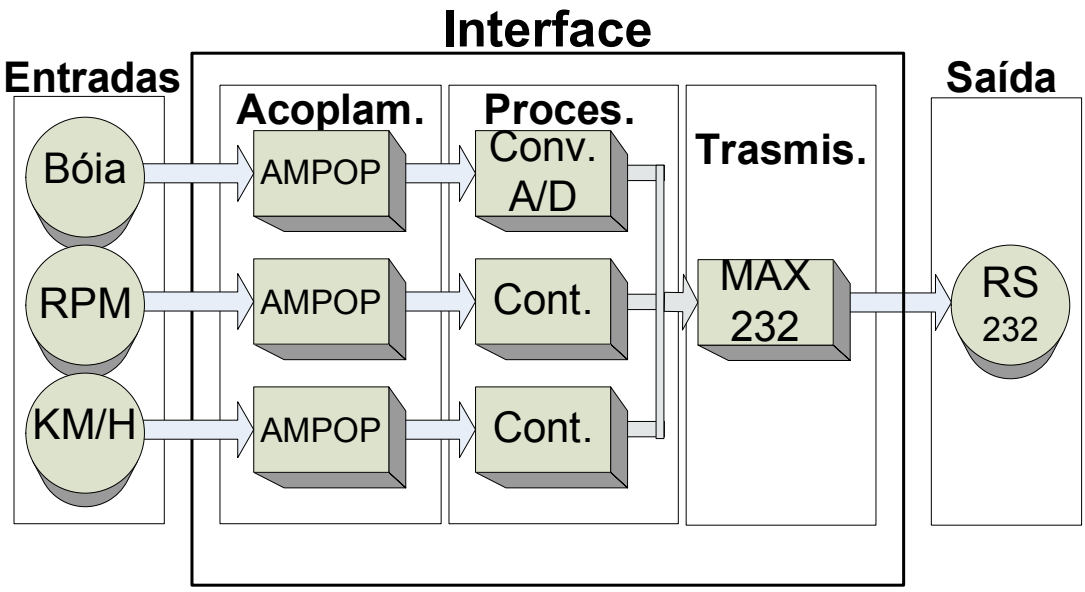


Figura 4.1 – Diagrama de Blocos da Interface

4.1. Módulo de Acoplamento

O Módulo de Acoplamento é composto por amplificadores operacionais e resistores. Este módulo possui a funcionalidade de receber os sinais dos três sensores e condicionar ao padrão Transistor Transistor Logic (TTL), tornando-os inteligíveis para o microcontrolador.

Além disso, serve para evitar que a Interface modifique qualquer um dos três sinais. A Interface não pode drenar e nem adicionar corrente aos sinais disponibilizados devido ao fato dos sensores continuarem alimentando o painel de instrumentos.

Os amplificadores operacionais foram escolhidos por apresentar duas características básicas que atendem aos pré-requisitos da Interface:

- Não drenar corrente, pois apresenta uma alta impedância nas entradas; e
- Facilidade de se trabalhar a amplitude do sinal utilizando resistores para ajustar o ganho.

Considerando a indisponibilidade de fonte simétrica no circuito, foi utilizado um divisor de tensão resistivo na entrada V+ para manter o amplificador dentro da faixa de operação linear. (Malvino 95)

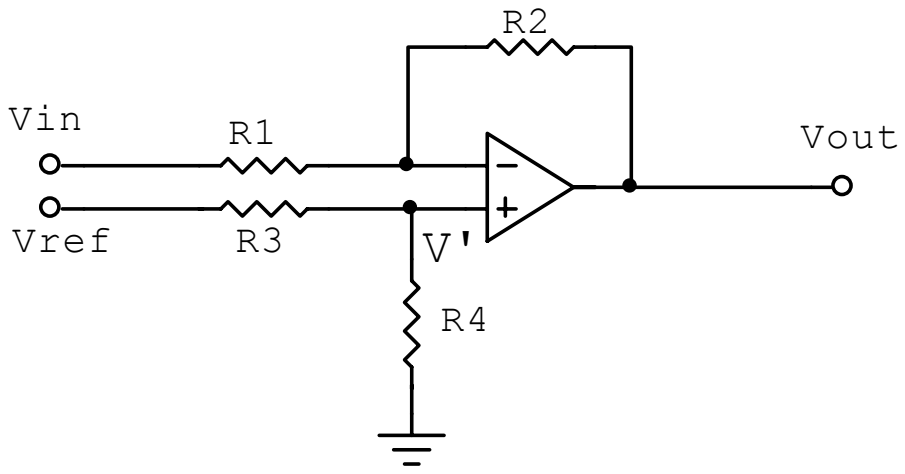


Figura 4.2 – Modelo de Utilização do AMPOP

De acordo com a Figura 4.2, obtém-se a Equação 4.1:

$$V' = \frac{R4}{R3+R4} * Vref$$

$$\frac{Vout - V'}{R2} = \frac{V' - Vin}{R1}$$

$$\therefore Vout = -\frac{R2}{R1} * Vin + \frac{R2+R1}{R1} * \frac{R4}{R3+R4} * Vref \qquad \textbf{(Equação 4.1)}$$

onde:

- **V':** tensão nas entradas do AMPOP;
- **Vout:** tensão de saída do AMPOP;
- **Vin:** tensão de entrada do circuito;
- **Vref:** tensão de referência; e
- **R1, R2, R3 e R4:** resistores.

O Vref utilizado para o acondicionamento de todos os sinais foi de 5 Volts. Outros valores de Vref implicariam na adição de componentes ao circuito. Isto considerando que o outro valor de tensão regulado no circuito é de 9 Volts, destinado a alimentação dos amplificadores operacionais.

Os sensores de rotação e velocidade apresentam sinais discretos com amplitude de 12 Volts. Estes sinais foram acondicionados a uma amplitude de 5 Volts utilizando-se valores comerciais de resistores, de acordo com a Equação 4.1. Isto não impactará na precisão devido ao fato do microcontrolador identificar apenas as bordas de transição dos sinais.

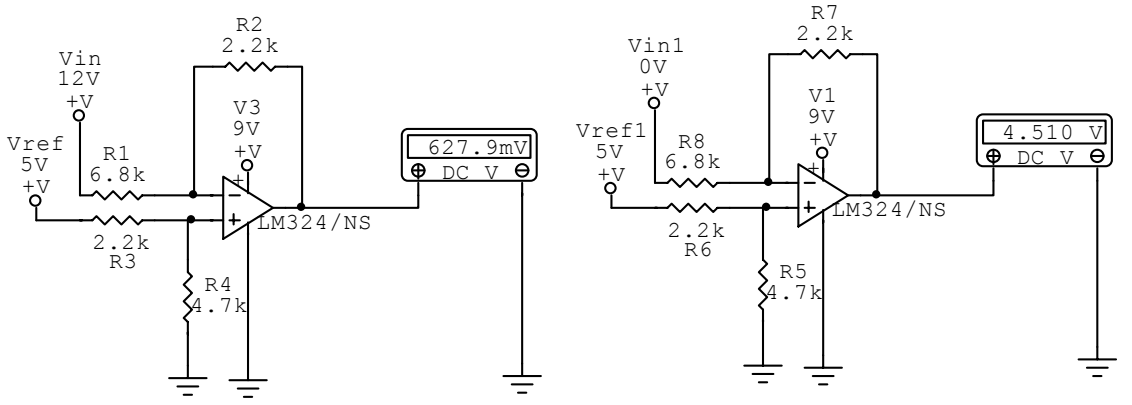


Figura 4.3 – Circuito elétrico do acoplador dos Sinais Discretos

O sinal analógico do medidor de nível de combustível apresenta uma ddp variando de 1,64 Volts a 6,48 Volts. Este sinal foi modelado para valores de 0,8 Volts a 4.8 Volts, para se adequar ao intervalo de operação do microcontrolador, proporcionando uma maior tolerância à tensão de *off set* do AMPOP.

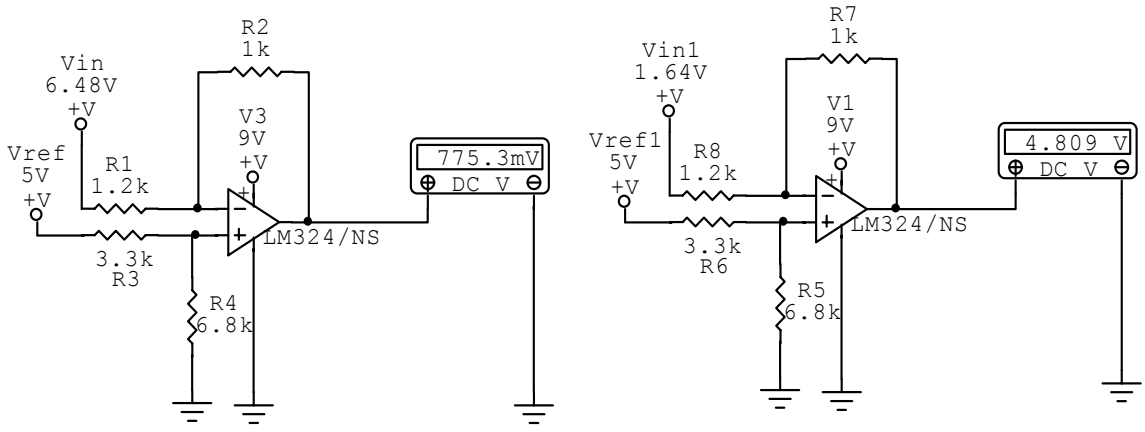


Figura 4.4 - Circuito elétrico do acoplador do Sinal Analógico

O circuito integrado (CI) adotado para o projeto foi o LM 324, principalmente porque possui quatro amplificadores contidos em um único

encapsulamento. Além disso, é largamente utilizado em aplicações de uso geral e de fácil aquisição por preços relativamente baixos.

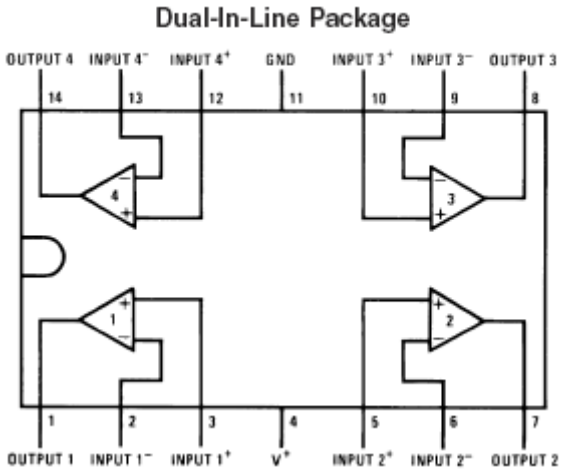


Figura 4.5 – Diagrama do LM 324N

4.2. Módulo de Processamento

Este módulo é o responsável por receber os sinais definidos pelo módulo de acoplamento, levantar a freqüência dos sinais discretos dos sensores de velocidade e rotação, quantizar o sinal do sensor de nível de combustível e disponibilizar todos os dados processados em uma saída Universal Synchronous Asynchronous Receiver Transmitter (USART).

O núcleo do módulo de processamento é o microcontrolador PIC 16F877A, fabricado pela Microchip Technology.

Este microcontrolador é o mais completo da família 16F e foi escolhido devido a duas características básicas: a existência de um conversor A/D interno de 10 bits (que possibilita 1024 níveis de quantização) e de uma porta USART (que simplifica a implementação da interface RS-232).

Os demais componentes do módulo de processamento foram dimensionados com base nas especificações de operação do microcontrolador.

A freqüência de operação do PIC 16F877A pode variar de 32 kHz a 20 MHz. Foi escolhida a freqüência de 4 MHz, conforme recomendado na bibliografia correspondente. [Pereira, 2004]

Os componentes que compõem o Módulo de Processamento são:

- O PIC 16F877A;
- Um cristal de 4 MHz; e
- Dois capacitores cerâmicos de 15 pF.

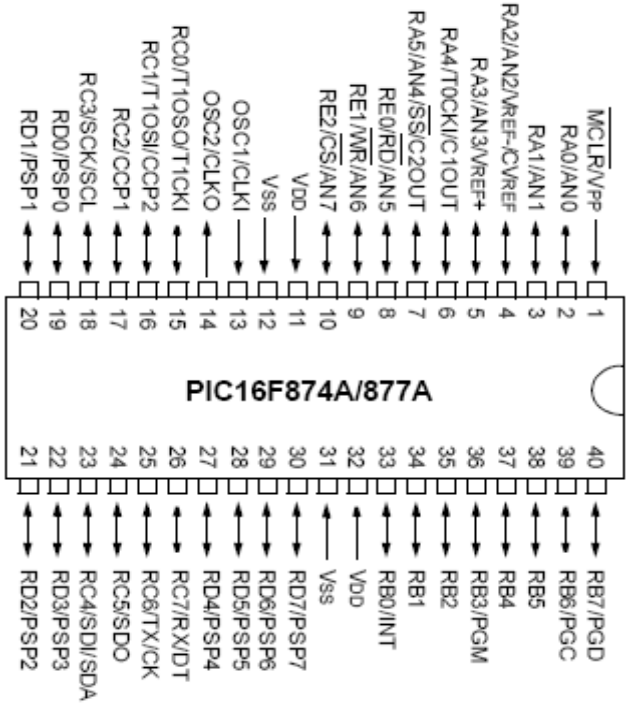


Figura 4.6 – PIC 16F877A

O cristal define a freqüência de operação do microcontrolador e deve ser ligado aos pinos 13 e 14, juntamente com dois capacitores de 15 pF, um em

cada pino. Um terminal de cada capacitor deve ser ligado a GND, a Fig. 4.7 ilustra este tipo de ligação. [Microchip]

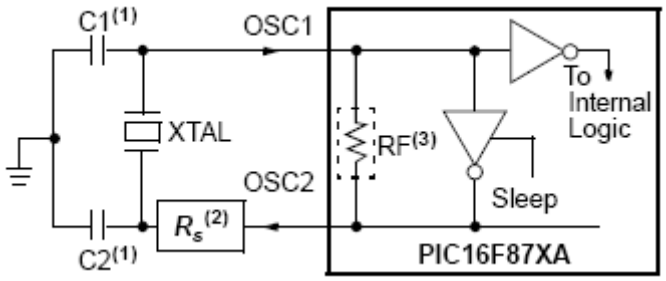


Figura 4.7 – Esquema de ligação do cristal

Tabela 4.1 - Valores de Capacitores por Freqüência de Trabalho

Tipo de Osciladores	*Freqüência do Cristal*	*Capacitor C1*	*Capacitor C2*
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	**4 MHz**	**15 pF**	**15 pF**
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF

Além da alimentação correta do microcontrolador e acoplamento do cristal, um ponto importante em sua operação é a ligação do pino MCLR. Este pino, quando em nível alto, faz com que o microcontrolador trabalhe normalmente, porém em nível baixo o mesmo é desligado.

No projeto este pino foi conectado à VDD por intermédio de um resistor de 10KΩ e a GND por intermédio de um botão do tipo *push bottom* a fim de prover um dispositivo de *reset* ao microcontrolador. Na Fig. 4.8 foi ilustrada a correta ligação do MCLR do microcontrolador. [Microchip]

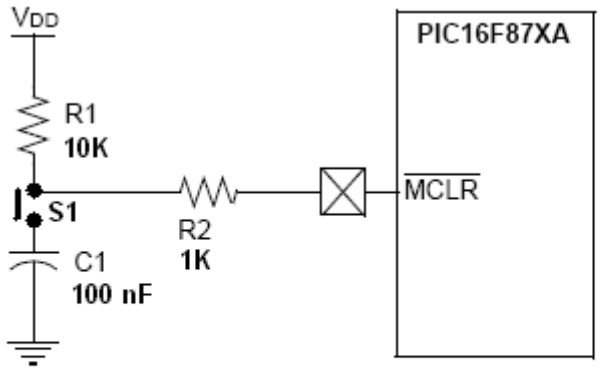


Figura 4.8 – Esquema de ligação do MCLR

4.3. Módulo de Transmissão Serial

O terceiro módulo da Interface é responsável por receber os dados da porta USART do PIC, formatar estes dados de acordo com a especificação RS 232 e disponibilizá-los para a porta serial do PC.

O microcontrolador adotado (PIC 16F877A) possui uma saída USART para comunicação com a porta serial de um PC. Porém esta saída não pode ser ligada diretamente a ela, pois, a saída do PIC apresenta níveis de tensão

TTL (0 Volt a 5 Volts) e a porta serial (de acordo com o padrão RS 232) necessita de níveis de +7,5 Volts a +15 Volts e –7,5 Volts a –15 Volts.

Para que seja realizada a conexão entre o PIC e a porta serial, é necessário que se converta os níveis de tensão TTL para os níveis padrão RS 232. Tal objetivo é alcançado lançando - se mão de um conversor de nível de tensões MAX 232.

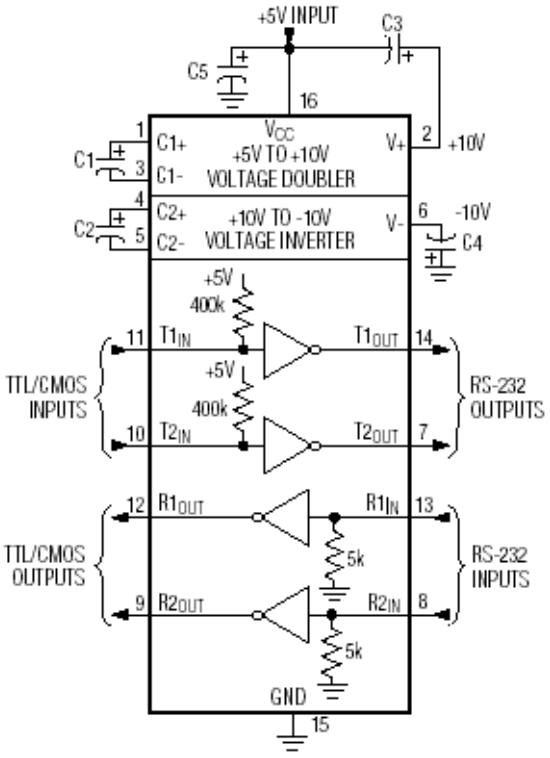


Figura 4.9 – Diagrama Esquemático do CI MAX 232

O MAX 232, fabricado pela Maxim, foi desenvolvido especificamente para converter níveis TTL, para níveis compatíveis com a porta serial do PC. Ele foi escolhido pois possui baixo custo e não necessita de fonte de alimentação simétrica.

O MAX 232 é dotado de um duplicador de tensão que disponibiliza 10 Volts, e um inversor no intuito de transformar os 10 Volts obtidos em -10 Volts. Para tanto, é necessário apenas a adição de 4 capacitores de 1 μF.

Além do MAX 232 e dos capacitores, o módulo de transmissão é composto por um conector DB 9 fêmea.

CAPÍTULO 5 - O FIRMWARE

5.1. Fluxograma do Firmware

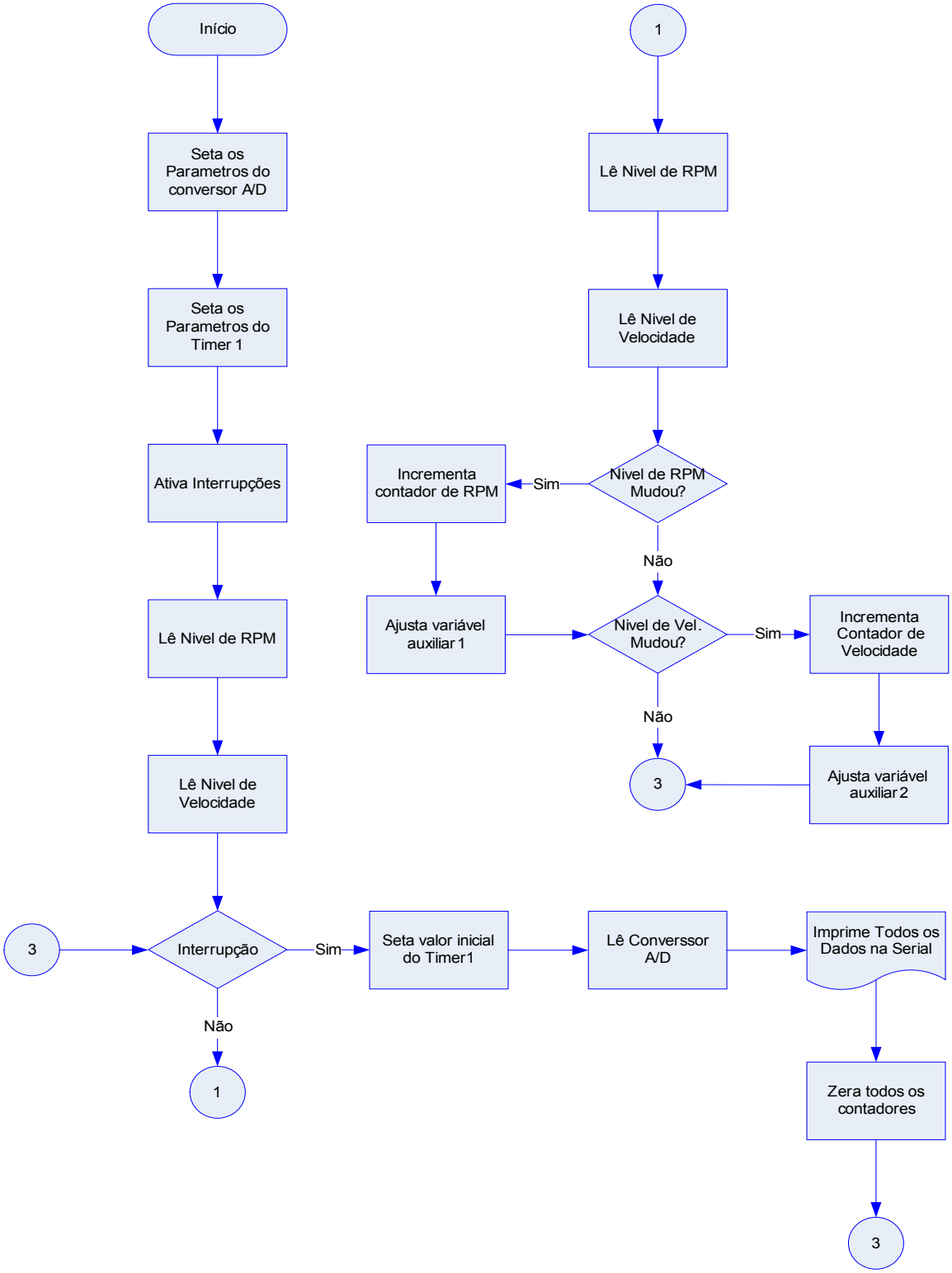


Figura 5.1 - Fluxograma do Firmware

5.2. O Desenvolvimento do Firmware

Para o desenvolvimento do firmware, foi utilizado o ambiente MPLAB IDE versão 7.21. O MPLAB IDE é um ambiente de desenvolvimento integrado *(Integrated Development Environment* - IDE) disponibilizado gratuitamente pela própria Microchip. O MPLAB IDE utiliza a linguagem de programação *Assembler* como nativa, porém foi desenvolvido pela CCS Inc. um compilador e um *plug-in* que adiciona a linguagem C no portifólio do MPLAB IDE.

Todo o Firmware deste projeto foi desenvolvido em C, o que facilita a manutenção e a portabilidade para outro microcontrolador.

5.3. Estratégia do Firmware

Para o cálculo da rotação do motor e da velocidade, lidas dos sensores, o Firmware incrementa dois contadores a fim de acumular as transições de subida e descida de nível.

Em uma freqüência de 4 Hz os valores acumulados são então enviados à USART em conjunto com o valor coletado instantaneamente pelo conversor A/D.

A freqüência de 4 Hz foi conseguida utilizando o Timer1 de 16 bits com um *prescaler* de 4.

Para calcular o valor de inicialização do Timer1, em busca de uma freqüência de 4 Hz, deve-se levar em consideração:

- a freqüência interna do microcontrolador é de ¼ da freqüência do oscilador;

- o Timer1 possui 16 bits, logo seu valor de “estouro” é de 65.535; e
- o *prescaler* do Timer1 pode ser de 2, 4 ou 8.

$$\text{Prescaler} = \frac{\frac{1}{4} \times Clock}{Freq.Desejada \times 2^{\text{bits do timer}}} = \frac{\frac{1}{4} \times 4 \times 10^6}{4 \times 2^{16}} = 3,8147$$

$$\text{Valor Inicialização} = 2^{\text{bits do timer}} - \frac{\frac{1}{4} \times Clock}{Freq.Desejada \times \text{Prescaler}}$$

$$\text{Valor Inicialização} = 2^{16} - \frac{\frac{1}{4} \times 4 \times 10^6}{4 \times 4} = 2^{16} - 62500 = 3.036$$

(Equação 5.1)

5.4. A Gravação do Firmware

O último processo para a conclusão do módulo de processamento é a gravação do firmware dentro da memória *flash* do microcontrolador. Para tanto é necessária a utilização de um gravador compatível com o microcontrolador.

No projeto proposto foi utilizado o gravador MC PLUS fabricado pela Mosaico Engenharia. A principal motivação para utilização do gravador MC PLUS foi a forte integração com a plataforma de desenvolvimento utilizada, o MP LAB IDE. Na Fig. 5.2 é exibida um foto do gravador MC PLUS.

Figura 5.2 – MC PLUS

CAPÍTULO 6 - O SOFTWARE RECEPTOR

O software receptor foi a última etapa desenvolvida no projeto. Para o desenvolvimento das etapas anteriores foi utilizado o Hiper Terminal do Windows XP, em ambiente de testes.

O software desenvolvido possui a funcionalidade de exibir os dados lidos da porta serial de forma amigável para o usuário.

6.1. Linguagem Java e IDE NetBeans

Java é uma linguagem de programação orientada a objetos extremamente poderosa e largamente utilizada.

O sucesso da linguagem Java está diretamente relacionado a dois aspectos: o primeiro é a distribuição livre realizada pela Sun Microsystems e o outro é o reaproveitamento de código.

Com o reaproveitamento de código as aplicações (classes) desenvolvidas em Java podem ser reutilizadas em novas aplicações com extrema facilidade.

O Software Receptor foi totalmente desenvolvido utilizando a linguagem de programação Java e a plataforma de desenvolvimento utilizada foi o NetBeans 4.1, distribuída gratuitamente pela Sun Microsystems.

6.2. API Java Communications

A Sun Microsystems, além da linguagem e das plataformas de desenvolvimento, também disponibiliza várias Interfaces de Programação de Aplicativos (*Application Program Interface* - API).

API é um conjunto de primitivas que integra a definição e a manipulação de objetos seguindo os padrões da linguagem de programação.

No desenvolvimento do software proposto no projeto foi utilizado a API *Java Communications*. Esta API define várias classes e métodos que possibilitam, por exemplo, o reconhecimento das portas seriais disponíveis, a abertura destas portas e a comunicação por intermédio destas mesmas portas.

6.3. Desenvolvimento do Software Receptor

O sistema possui duas telas, uma para seleção da porta serial e outra tela onde os dados enviados pela Interface são exibidos.

Figura 6.1 - Tela Para Seleção da Porta

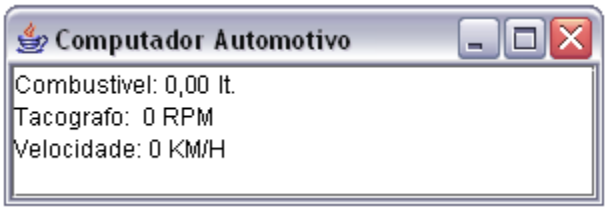


Figura 6.2 – Tela de Exibição dos Dados

6.4. O Tratamento dos dados

Todo o tratamento dos dados é feito pelo Software Receptor. A Interface envia uma única ‘string’ com todas as informações, e o Software Receptor

extrai os dados dos três sensores.

A ‘string’ é recebida na porta serial com a seguinte formatação:
{b:xx;r:xx;v:xx;}

Onde:

- **b:** Precede o valor fornecido pelo conversor AD;
- **r:** Precede o valor da freqüência referente ao sensor de rpm; e
- **v:** Precede o valor da freqüência referente ao sensor de velocidade.

Depois de receber a ‘string’ o software receptor extrai os valores dos três sensores, posteriormente os armazena em três variáveis denominadas: bóia, velocidade e rpm.

Para os sensores de rotação do motor e de velocidade veículo, é feito uma associação direta entre o valor de freqüência e os valores de velocidade e rpm, as equações 6.1 e 6.2 definem o relacionamento.

velocidade=(velocidade*100)/423 **(Equação 6.1)**

rpm=rpm*30 **(Equação 6.2)**

No tratamento da variável bóia foi necessário converter o valor lido no conversor AD para um valor em litros. Isso levando em consideração os valores de máximo e de mínimo lidos pelo conversor AD.

A equação 6.3 descreve o relacionamento da quantidade de litros com o valor fornecido pelo conversor AD.

Litros=(60*(bóia-246)/690) **(Equação 6.3)**

onde:

- **Litros:** Corresponde a variável que recebe o valor de litros;

- **60:** Valor máximo de litros;
- **bóia:** Variável com o valor fornecido pelo conversor AD;
- **246:** Valor mínimo lido pelo conversor AD; e
- **690:** Faixa de variação do conversor AD.

Devido a movimentação do veículo o sensor de nível de combustível apresentou uma grande variação. Esta variação provocou uma grande discrepância na apresentação da quantidade de litros. Este problema foi superado com a implementação de uma média móvel aritmética limite de até 120 amostras. O valor de 120 amostras foi definido com base na freqüência de atualização de 4 Hz resultando em um tempo de 30 segundos para o enchimento da fila.

CAPÍTULO 7 - CONSIDERAÇÕES FINAIS

7.1. Dificuldades Encontradas

A primeira dificuldade foi o levantamento dos sinais disponibilizados pelos sensores do veículo. A documentação do esquema elétrico do automóvel é de difícil acesso, pois é de uso exclusivo de oficinas autorizadas.

Mesmo possuindo acesso a esta documentação, foi constatado que ela não apresenta as informações necessárias de forma detalhada. Foi necessário realizar medições dos sinais com o veículo em funcionamento e seu relacionamento com as leituras do painel de instrumentos do veículo.

Além disso, durante a programação do microcontrolador foi identificada a necessidade de otimização do código a fim de obter leituras mais precisas das freqüências.

A leitura de tais freqüências apresentava resultados insatisfatórios quando do início do projeto, porque a Interface foi desenvolvida em um *protoboard*. Identificado este problema, o mesmo foi superado com a montagem da Interface em uma placa de circuito impresso.

Por fim, a *API Java Communicatons* é dotada de uma excelente documentação associada a sua utilização, porém esta documentação é deficiente no que se refere ao processo de instalação da API. Em nenhum ponto da documentação é detalhado o processo de instalação dos arquivos da API.

7.2. Resultados Obtidos

O projeto tem em sua concepção o objetivo estritamente acadêmico de construir uma interface de coleta e disponibilização de dados de vários sensores de um veículo em um computador pessoal. Este objetivo foi alcançado com êxito.

Foi construída uma interface totalmente funcional e otimizada e o software de recepção dos dados no PC, apresentando ambos um excelente tempo de resposta.

7.3. Conclusões

Apesar da escassez e deficiência da documentação referente a veículos, foi possível realizar uma serie de melhorias com o levantamento de dados diretamente no veículo.

Utilizando a mesma metodologia de coleta de dados é possível o monitoramento de qualquer parâmetro do veículo.

Além da melhoria do monitoramento de veículos o projeto realizado abre uma serie de possibilidades para opcionais que não são oferecidos para determinados modelos.

7.4. Sugestões para Trabalhos Futuros

Este trabalho fornece um grande leque de possibilidades de evolução e trabalhos futuros. Dentre elas, podemos destacar:

- Gravação dos dados em uma base histórica possibilitando uma posterior consulta a fim de facilitar manutenções corretivas e

preventivas;

- Conexão de mais sensores a interface disponibilizando mais parâmetros do veículo para monitoramento;
- Incorporação de um rádio transmissor a Interface, possibilitando a telemetria unidirecional do veículo, com a transmissão em tempo real dos parâmetros do veículo para um ponto centralizado e
- Possibilidade de telemetria bidirecional onde, além de receber os dados, a central de operações também poderia atuar diretamente nos parâmetros de funcionamento do veículo.

REFERÊNCIAS BIBLIOGRÁFICAS

Arquivo do Carro Nacional http://www.carronacional.com.br/

ANFAVEA, **Associação Nacional dos Fabricantes de Veículos** http://www.carronacional.com.br/

GOMES, Daniel Vasconcelos. **Comunicação Serial Utilizando a API da SUN**. http://www.guj.com.br/content/articles/javacommapi/JavaCommAPI.pdf

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

Microchip Inc. **Data Sheet do PIC16F877A.** http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf

PEREIRA, Fábio. **Microcontroladores PIC - Programação em C**. 3º Edição. Editora: Érica, 2004.

Ribeiro, Fábio Von Glehn. **Curso de Injeção Eletrônica**. 3º edição, Goiânia, 1999.

SILVEIRA, Jorge Luiz. **Comunicação de Dados e Sistemas de Teleprocessamento**. Ed. Makron Books, São Paulo, 1991.

SOUZA, David Jose de. **Conectando o PIC: Recursos Avançados**. 2º Edição. Editora: Érica

Sun Microsystems. **Documentação da API do Java 1.5** http://java.sun.com/j2se/1.5.0/docs/api/

Anexo A – Código Fonte do Software do Microcontrolador

```
//=========================================
//=========================================
//Firmware da Interface V1.5
//Autor: Toni Gledson
//=========================================
#include  <16f877a.h>
//Define conversor A/D de 10 bits
#device adc=10
//Velocidade do clock
#use delay(clock=4000000)
#fuses HS,NOWDT,PUT,NOLVP
//Define velocidade da porta serial
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#include <regs_16f87x.h>
//Declara variáveis globais
long int contV=0;
long int contR=0;
//Define função de tratamento do timer1
#int_timer1
// Função de tratamento do timer1
void trata_t1 (){
     //Declara variáveis da função
     long int boia=0;
     //Recarrega o valor inicial do timer1
     set_timer1(3036+get_timer1 ());
     //Lê o valor no conversor A/D
     boia = read_adc();
     //Imprime na serial
     Printf
("{b:%lu;r:%lu;v:%lu;}",boia,contR*2,contV*2);
     //Zera acumuladores
     contV=0;
     contR=0;
```

```
     }
void main() {
     //Declara variáveis locais
     short int rpm,vel,aux1,aux2;
     //Seta o bit 0 da porta 'A' como analógico
     setup_ADC_ports (RA0_analog);
     //Seta o clock do conversor A/D como interno
     setup_adc (ADC_CLOCK_INTERNAL);
     //Seta o canal de utilização do conversor A/D
     set_adc_channel (0);
     //Seta o timer1 como interno e o prescaler de 4
     setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_4);
     //Seta o valor de inicialização do timer1
     set_timer1 (3036);
     //Habilita interrupções Globais
     enable_interrupts (GLOBAL);
     //Habilita interrupções do timer1
     enable_interrupts (INT_TIMER1);
     //Aguarda 200 mili segundos
     delay_ms (200);
     //Captura os níveis dos bits 6 e 7 da porta 'B'
     aux1 = input (pin_b6);
     aux2 = input (pin_b7);
     while (true){
          rpm= input (pin_b6);
          //Verifica se ocorreu mudança de nível
          if (aux1 != rpm){
               contR++;
               aux1=rpm;
               }
          vel= input (pin_b7);
          //Verifica se ocorreu mudança de nível
          if (aux2 != vel){
               contV++;
```

```
                aux2=vel;
                }
        }
}
```

Anexo B – Código Fonte do Software Receptor

```
//*****************************************
//*****************************************
// Computador de Bordo Automotivo
//*****************************************
//Autor: Toni Gledson Dantas da Silva
//Data: 1 de Novembro de 2005, 11:40
//*****************************************

import java.io.IOException;
import java.io.InputStream;
import java.util.Enumeration;
import java.util.TooManyListenersException;
import javax.comm.CommPortIdentifier;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;
import javax.comm.UnsupportedCommOperationException;
import javax.swing.*;
import java.awt.*;
import java.text.*;
import java.awt.event.*;

//Definição da classe do Software Receptor
public class main extends JFrame implements Runnable,
SerialPortEventListener {

     //Identificador da porta para comunicação
     static CommPortIdentifier portId;

     //Lista de opções para seleção da porta
     static Enumeration portList;

     //Stream para a transmissão dos dados
     InputStream inputStream;

     //Objeto que representa a porta serial
     static SerialPort serialPort;

     //Thread da classe para monitoramento da porta
     Thread readThread;

     //Área de texto que exibirá os dados recebidos
     JTextArea area;

     //Variáveis auxiliares
     static String str = "";
     int boia, velocidade, rpm;
```

```
     float autonomia;

//Construtor da classe
public main() {
     //Abre a porta serial
     try {
          serialPort = (SerialPort)
portId.open("SoftwareReceptor", 2000);
     } catch (PortInUseException e) {
     JOptionPane.showMessageDialog(null,e.getMessage());
     }

     //Captura a Stream de dados
     try {
          inputStream = serialPort.getInputStream();
     } catch (IOException e) {
     JOptionPane.showMessageDialog(null,e.getMessage());
     }

     //Adiciona a classe na lista de listners da porta
     try {
          serialPort.addEventListener(this);
     } catch (TooManyListenersException e) {
     JOptionPane.showMessageDialog(null,e.getMessage());
     }

     //Configura para notificar quando houver dados
     serialPort.notifyOnDataAvailable(true);

     //Configura os parâmetros da comunicação
     try {
          serialPort.setSerialPortParams(19200,
          SerialPort.DATABITS_8,
          SerialPort.STOPBITS_1,
          SerialPort.PARITY_NONE);
     } catch (UnsupportedCommOperationException e) {}

     //Cria uma nova thread
     readThread = new Thread(this);
     readThread.start();

     //Configuração dos components visuais
     setTitle("Computador Automotivo");
     setBounds(100,100,300,110);
     JPanel painel = new JPanel(new BorderLayout());
     area = new JTextArea();
     area.setLineWrap(true);
     JScrollPane scroll = new JScrollPane(area);
     painel.add(scroll,"Center");
     //Rotina para encerramento do programa
     ExitWindow exit = new ExitWindow();
```

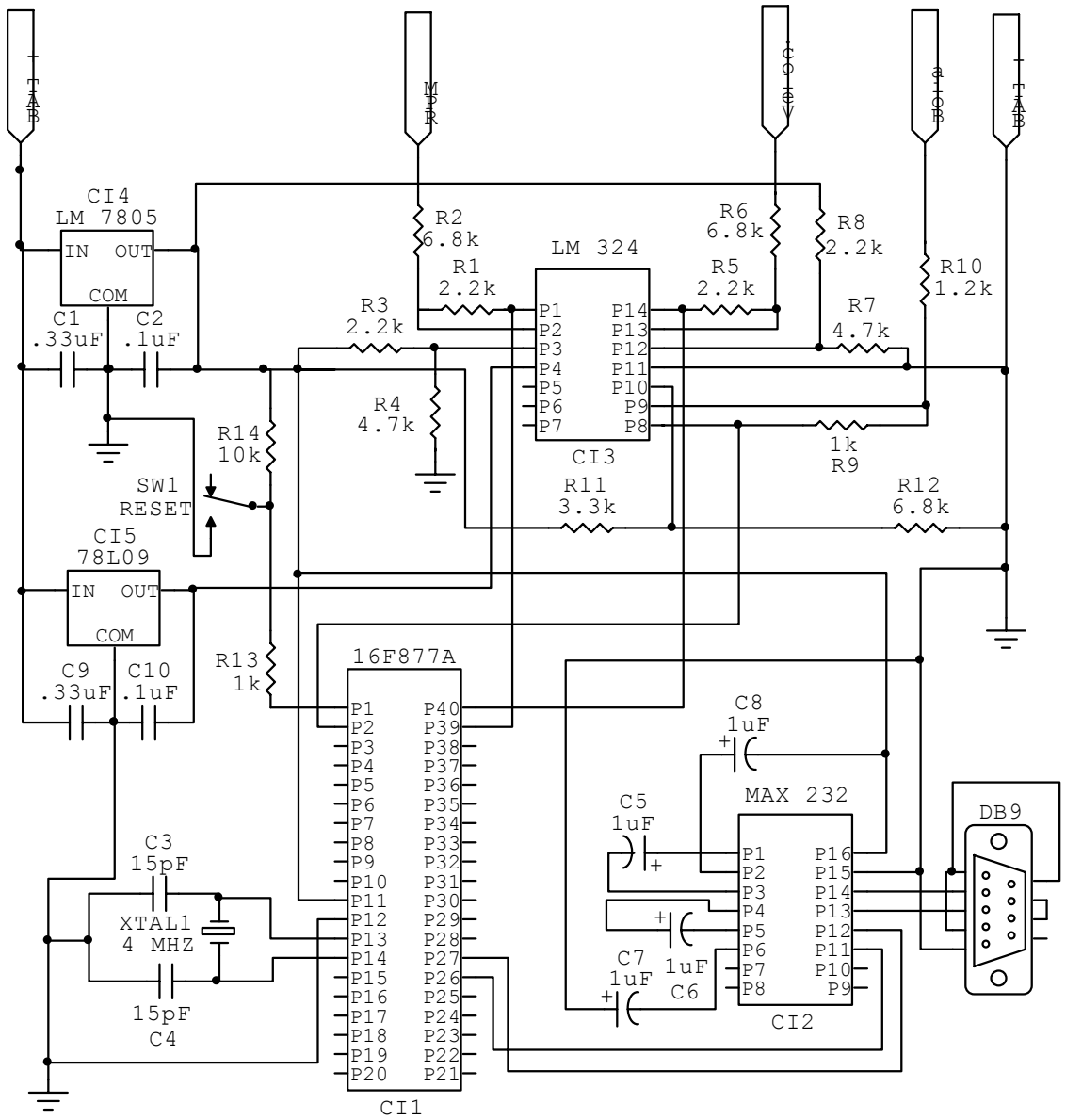
```
     this.addWindowListener(exit);
          this.setContentPane(painel);
        this.show();
}

//Ponto de entrada da thread
public void run() {
     try {
     Thread.sleep(20000);
     } catch (InterruptedException e) {}
}

//Método para tartar eventos da porta serial
public void serialEvent(SerialPortEvent event) {
     switch(event.getEventType()) {
          case SerialPortEvent.BI:
          case SerialPortEvent.OE:
          case SerialPortEvent.FE:
          case SerialPortEvent.PE:
          case SerialPortEvent.CD:
          case SerialPortEvent.CTS:
          case SerialPortEvent.DSR:
          case SerialPortEvent.RI:
          case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
               break;
          case SerialPortEvent.DATA_AVAILABLE:
               readData();
               break;
     }
}

//Ponto de entrada do programa
public static void main(String []args) {
     //Monta a lista de portas seriais
     Object[] possiveisValores =
("COM1","COM2","COM3","COM4"};

     //Mostra a caixa de seleção de porta
     String valorSelecionado =(String)
JOptionPane.showInputDialog(null,"Porta", "Selecione a
porta", JOptionPane.INFORMATION_MESSAGE, null,
possiveisValores, possiveisValores[0]);

     //Verifica se a porta selecionada existe
     if(valorSelecionado != null &&
!valorSelecionado.equals("")){
          portList =
CommPortIdentifier.getPortIdentifiers();
          while (portList.hasMoreElements()) {
               portId = (CommPortIdentifier)
portList.nextElement();
```

```
                if (portId.getPortType() ==
CommPortIdentifier.PORT_SERIAL) {
                        if
(portId.getName().equals(valorSelecionado)) {
                                //Cria a nova thread
                                main frame = new main();
                        }
                }
          }

          if(serialPort == null){
                JOptionPane.showMessageDialog(null,"Porta
não existe");
                System.exit(0);
          }
     }else{
          System.exit(0);
     }
}

//Trada o dado recebido e escreve na caixa de texto
public void readData()
{
     String str;
     int bytes;
     byte[] readBuffer = new byte[2048];
     try{
          while (inputStream.available() > 0 ){
                bytes = inputStream.read(readBuffer);
                if (bytes > 0){
                     if (bytes > readBuffer.length) {

     System.out.println(serialPort.getName()+": Input
buffer overflow!");
                     }
                     displayText(readBuffer, bytes);
          }
          }
     }
     catch (IOException ex) {
          System.out.println(serialPort.getName()+ ":
Cannot read input stream");
     }
}

//Exibe os bytes recebidos na caixa de texto
private void displayText(byte[] bytes, int byteCount) {
     int posi, posf;
     float litros;
     String temp;
     str += new String(bytes, 0, byteCount);
```

```
     //Extrai os valores de rpm, velocidade e boia da
     //string de dados recebida
    posf = str.lastIndexOf("}");
     posi = str.lastIndexOf("{", posf);
    if (posi != -1 && posf != -1){
     temp = str.substring(posi + 1, posf);
     str = str.substring(posf + 1);
     posi = temp.indexOf("b:");
     posf = temp.indexOf(";", posi);
          boia = Integer.valueOf(temp.substring(posi + 2,
posf)).intValue();
          posi = temp.indexOf("r:");
     posf = temp.indexOf(";", posi);
     rpm = Integer.valueOf(temp.substring(posi + 2,
posf)).intValue();
          posi = temp.indexOf("v:");
     posf = temp.indexOf(";", posi);
     velocidade = Integer.valueOf(temp.substring(posi + 2,
posf)).intValue();
    }

    //Trata boia, velocidade, rpm e autonomia
    if (boia<246)
        litros=0;
       else
        litros = (60*(boia-246)/690);

      if (litros >= 60)
        litros = 60;

    adicionaValorBoia(litros);
    rpm = rpm*30;
    autonomia = (litros*98)/ 10;
    velocidade = (velocidade*100)/423;

     //Escreve os dados na caixa de texto
    area.setText("Combustivel: " + litros + " lt.");
    area.append("\nTacografo:  " + rpm + " RPM");
    area.append("\nVelocidade: " + velocidade + " KM/H");
    area.append("\nAutonomia: " + autonomia + " KM");
}
//Adiciona um novo valor a lista de valores da boia
void adicionaValorBoia(float valor) {
    //Adiciona o valor recebido
    valoresBoia.addLast(new Float(valor));

    //Se após adicionar o valor o tamanho da lista for
maior que 40,
    //então retira um elemento. Serve para limitar a lista.
    if (valoresBoia.size() > 120) {
```

```
            valoresBoia.removeFirst();
        }
    }

    //Retorna a média dos valores da lista
    float mediaValorBoia() {
        //Inicializa um objeto de iteração
        Iterator i = valoresBoia.iterator();

        //Inicializa a variável soma
        float soma = 0, média;

        //Loop para percorrer a lista e acumular seus elementos
    em soma.
        while (i.hasNext()) {
            soma += ((Float) i.next()).floatValue();
        }

        //Retorna a média, dividindo a soma pelo tamanho da
    lista
        if (soma == 0) {
            media = 0;
        } else {
            media = soma/valoresBoia.size();
        }
        return media;
    }

         //Classe para destruição do programa fechado
        class ExitWindow extends WindowAdapter{
                public void windowClosing(WindowEvent e){
                        System.exit(0);
                }
        }
    }
```

Anexo C – Diagrama Elétrico Completo da Interface



Anexo D – Esquema Elétrico de Ligação da CIE do Vectra GLS 2.2