MARCOS FELIPE DE ALMEIDA ARAUJO
RENATO CLEMENTINO DE SOUSA

CUSTOMIZAÇÃO DO PAINEL DE INSTRUMENTOS AUTOMOTIVO

Santo André
2013

MARCOS FELIPE DE ALMEIDA ARAUJO
RENATO CLEMENTINO DE SOUSA

CUSTOMIZAÇÃO DO PAINEL DE INSTRUMENTOS AUTOMOTIVO

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Santo André, como exigência para a obtenção da graduação tecnológica em Eletrônica Automotiva.

.

Santo André
2013

MARCOS FELIPE DE ALMEIDA ARAUJO
RENATO CLEMENTINO DE SOUSA

CUSTOMIZAÇÃO DO PAINEL DE INSTRUMENTOS AUTOMOTIVO

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Santo André, como exigência para a obtenção da graduação tecnológica em Eletrônica Automotiva.

Orientador:
Prof. Mr Kleber Nogueira Hodel
Co-orientador:
Prof. Dr. Fabio Delatore

Santo André
2013

Fatec Santo André | CENTRO PAULA SOUZA | GOVERNO DE SÃO PAULO

LISTA DE PRESENÇA

SANTO ANDRÉ, 19 DE DEZEMBRO DE 2013.

LISTA DE PRESENÇA REFERENTE À APRESENTAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO COM O TEMA "**CUSTOMIZAÇÃO DO PAINEL DE INSTRUMENTO AUTOMOTIVO**" DOS ALUNOS DO 6º SEMESTRE DESTA U.E.

BANCA

PRESIDENTE:
PROF. MSC KLEBER NOGUEIRA HODEL_____

MEMBROS:
PROR. DR. FABIO DELATORE_____

PROF. WESLLEY M. TORRES_____

ENG. FERNANDO S. DA SILVA_____

ALUNOS:

RENATO CLEMENTINO DE SOUZA_____

MARCOS FELIPE DE ALMEIDA ARAUJO_____

www.fatecsantoandre.com.br fatecstdo@gmail.com

Rua Prefeito Justino Paixão, 150 – Centro – Santo André – SP - CEP: 09020-130

Fone (0xx11) 4437-2215

DEDICATÓRIA

Dedico este trabalho a minha família, meus pais:
Josenildo da Silva Araújo e Heloisa de Almeida Araújo
que são a base e o alicerce do meu desenvolvimento.
Marcos Felipe

Dedico este trabalho a minha família, minha esposa, meus filhos
e aos meus amigos que acreditaram na conquista
pessoal ao concretizar este trabalho.
Renato Clementino

AGRADECIMENTOS

A todos os funcionários dessa instituição de ensino, que iniciaram em 2007 com seriedade e dedicação os trabalhos da FATEC Santo André e hoje continuam contribuindo para manter a qualidade no ensino e um espaço bem estruturado.

Ao corpo docente pelo comprometimento e empenho em transmitir vosso conhecimento, em especial aos Professores: Alexsander Tressino, Carlos A. Morioka, Reginaldo C. Farias, Marco Fróes, Armando Laganá, Dirceu Fernandes, Wagner Massarope, Cleber William, Luis Kanashiro, Weslley Torres e Edson Kitani por auxiliarem também em nosso crescimento profissional e pessoal.

Aos nossos orientadores Kleber Nogueira Hodel e Fabio Delatore, que acreditaram nesse projeto e contribuíram muito para a realização do mesmo.

Aos ex-colegas de classe e hoje amigos por compartilharem de todos os momentos bons e ruins durante esse árduo e longo processo de graduação.

A todos que de alguma forma contribuíram na realização deste trabalho.

RESUMO

Com a crescente aplicação e uso das redes de comunicação de dados nos sistemas eletrônicos automotivos, uma considerável parcela de informações e dados importantes passam a ser facilmente disponibilizadas e acessadas, ao mesmo tempo, por diferentes periféricos eletrônicos dentro do veículo. Como o desenvolvimento de um veículo é realizado tendo como premissa a máxima satisfação dos clientes consumidores finais, cujo qual é altamente exigente com relação ao *design*, o painel de instrumentos é considerado como a segunda parte do veículo mais observada pelo consumidor em uma concessionária, onde a primeira acaba sendo a aparência exterior. O conceito deste projeto é proporcionar uma maior comodidade e interatividade do usuário com o painel de instrumentos. O objetivo do presente trabalho é apresentar um conceito de painel de instrumentos digital personalizado de acordo com o perfil do usuário, utilizando os dados disponíveis na rede comunicação CAN *(Controller Area Network)* do automóvel.

Palavras-chaves: Painel de Instrumentos. Comunicação CAN. Comodidade. *Design*.

ABSTRACT

With the increasing use and application of network data communication in automotive electronics systems, a considerable amount of important information and data will be easily available and accessible at the same time, for different electronic peripherals inside the vehicle. As longs as the development of a vehicle is done with that the premise the highest customer satisfaction final consumers, which is highly demanding due to the design, the instrument panel is considered as the second part of the vehicle most observed by the consumer at a car dealer, where the first is being just the outward appearance. The conception of this project is to provide greater comfort and user friendly dashboard. The objective of this work is to show a concept of tailored digital dashboard according to the user profile, using the data available on CAN communication network (ControllerArea Network) automobile.

Keywords: Dashboard. CAN Communication. Comfort. Design.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Bit* de conhecimento
BCD	*Binary Coded Decimal*
CAN	C*ontroller Area Network*
CRC	*Checksum*
DLC	Comprimento dos dados da mensagem CAN
ID	Identificador
IDE	Identificador de extensão
EOF	Fim de Frame
LabVIEW	*Laboratory Virtual Instruments Engineering Workbench*
RPM	Rotação Por Minuto
RTR	Pedido de Transmissão Remota
SAE	*Society of Automotive Engineering*
SOF	*Bit* de início da transmissão
TFT	*Thin Film Transistor*
VI	*Visual Instrument*

SUMÁRIO

1 – INTRODUÇÃO

1.1 Motivação

Quando alguém compra um automóvel, a sua decisão é fortemente influenciada pelo efeito da concepção global e as características técnicas. Após a compra, a percepção do condutor do veículo por 95 por cento do tempo consiste na percepção do *cockpit* [14].

Observando o cenário da indústria automotiva, a eletrônica embarcada está presente no desenvolvimento do veículo e é um dos fatores responsáveis por inovações, como por exemplo, o painel de instrumentos (Figura 1.1). A ideia deste projeto surgiu tendo em vista que, a tecnologia disponível nos painéis de instrumentos atuais, não atende a expectativa de todos os clientes, pois cada usuário possui uma preferência com relação ao estilo do painel de instrumentos automotivo.

Devido aos custos envolvendo *hardware* e *software* do painel de instrumentos, é inviável para as montadoras desenvolverem vários estilos de *Cluster* para um mesmo modelo de veículo.

Com o *hardware* e *software* desenvolvido no projeto, seria possível padronizar a linha de montagem do painel de instrumentos para todos os modelos de veículos, pois independente do modelo, a única modificação seria a parte gráfica do *Cluster*.

Figura 1.1 – Instrumentos Combinados do VW Polo.

1.2 Estado da arte do Painel de Instrumentos

Com o surgimento do primeiro velocímetro em 1902, seu inventor Otto Schulze lançou a pedra fundamental para todos os sistemas de informação aos motoristas. Este indicador de velocidade foi à primeira fonte de informação objetiva disponível, indo além do sentimento subjetivo do motorista para o carro (Figura 1.2).

Desde então, desenvolveu-se dentro do veículo, uma variedade de sistemas complexos que fornecem uma ampla gama de informações, desde condições de funcionamento do motor, até instruções de navegação.

Hoje, o painel de instrumentos é uma importante interface homem-máquina. Além de sua função técnica, também desempenha um papel importante na característica e estilo do veículo.

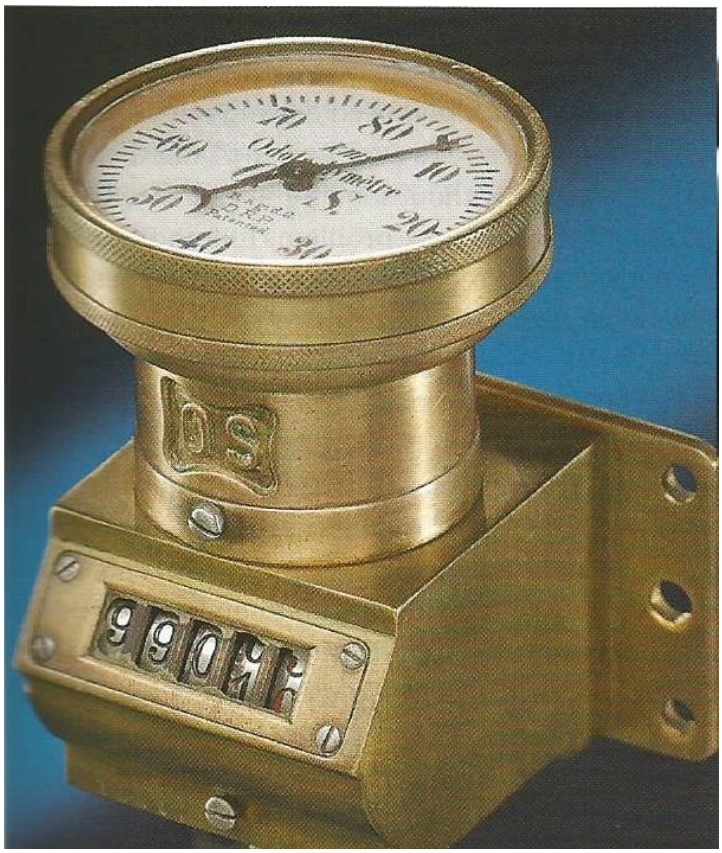


Figura 1.2 – Velocímetro O.S. de 1908 [14].

1.3 Objetivo

O desafio deste projeto é apresentar um conceito de painel de instrumentos digital personalizado de acordo com a preferência de cada usuário.
Com essa proposta é possível disponibilizar em um único painel de instrumentos vários estilos de *design* pré-definidos pelo fabricante, onde o motorista poderá alterar a interface gráfica do Cluster mantendo os parâmetros do veículo.

1.4 Metodologia

A visualização dos parâmetros que são apresentados digitalmente no painel de instrumentos do veículo, são coletados através do protocolo de comunicação CAN, analisados e tratados por um módulo eletrônico.
Esse módulo eletrônico desenvolvido no projeto é responsável por enviar os dados do automóvel para uma plataforma que possua um *software* de desenvolvimento gráfico.
Para apresentar o conceito, utilizamos o veículo Volkswagen Polo 2004 da Fatec Santo André, que possui uma rede de comunicação CAN. O *hardware* do módulo eletrônico utilizado para fazer a aquisição de dados do veículo, possui microcontrolador para o gerenciamento das informações com interfaces de comunicação CAN e serial RS232. Em questão do tempo para desenvolver o projeto, foi utilizado uma placa didática da Faculdade de Tecnologia de Santo André.
Na parte da interface gráfica, existem diversas ferramentas para desenvolvimento, a escolha da ferramenta ideal para realizar o *design* do painel de instrumentos depende de vários fatores, como custo, facilidade de utilização, quantidade de recursos, disponibilidade, entre outros. Para o desenvolvimento desse projeto que o objetivo é apresentar apenas o conceito de customização do painel de instrumentos, utilizamos a ferramenta LabVIEW da National Instruments, disponível da FATEC.

2. CONCEITOS TEÓRICOS

Os primeiros instrumentos combinados transmitiam poucas informações ao usuário. A leitura desses parâmetros era feita de forma puramente mecânica e transmitida através de cabos de aço. Com a evolução da eletrônica e a produção em larga escala, os módulos dos sistemas eletrônicos embarcados acabaram substituindo os componentes de controle mecânico, buscando maior conforto e segurança aos usuários.

Com o propósito de viabilizar a comunicação entre módulos, foi desenvolvido um protocolo de comunicação em série chamado de CAN (*Controller Area Network*). As mensagens CAN trafegam por um par de fios trançados, formando uma rede digital de comunicação em alta velocidade. Essa característica possibilita inúmeras aplicações e facilidades para o desenvolvimento de novos sistemas. Com isso, o conceito de um painel de instrumentos digital tornou-se possível, pois todos os parâmetros do automóvel estão disponíveis na rede. A utilização de uma ferramenta de design gráfico é uma alternativa para as montadoras oferecerem mais comodidade e interatividade aos clientes.

2.1 História do Painel de Instrumentos

Os primeiros veículos não possuíam um painel de instrumentos que informasse os parâmetros de funcionamento, não só do carro, mas também do motor ao condutor. Este conceito só foi introduzido nos automóveis a partir do inicio do século XX por Henry Ford com o Ford T ano 1908 e produzido em larga escala e implantando o conceito de produção em série. A Figura 2.1 apresenta o modelo em questão mencionado e a Figura 2.2, o seu respectivo painel de instrumentos.

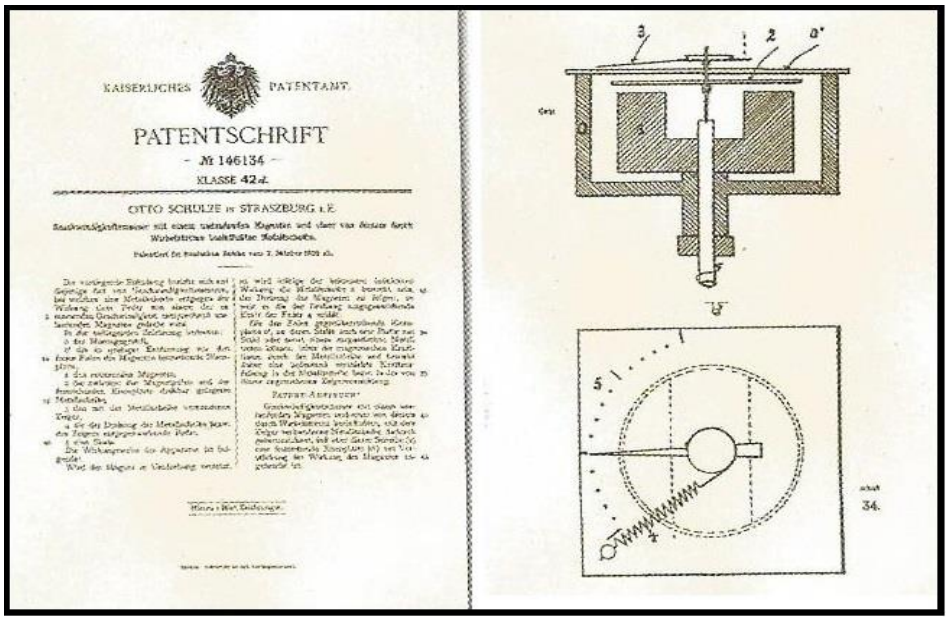
Figura 2.1 – Ford T, produzido por Henry Ford entre 1908 e 1927 [1].

Figura 2.2 – Painel do Ford T 1908 [3].

2.1.1 – Velocímetro

No conjunto de instrumentos, a função do velocímetro é indicar a velocidade do carro. Mesmo nos modelos de veículos mais novos, é usado um dispositivo que utiliza uma agulha para indicar uma velocidade específica. Já o tacômetro é o instrumento responsável por informar ao motorista as rotações do motor do veículo. Esta informação é essencial para que o motor tenha uma maior durabilidade, economia de combustível e controle do nível de emissões.

O velocímetro mecânico (velocímetro de correte de Foucault) foi inventado por Otto Schulze, de Estrasburgo (Patente em 1902) [4], apresentado pela Figura 2.3. Schulze imaginou o dispositivo revolucionário como solução para um problema cada vez maior. Os carros não apenas estavam se tornando mais populares, como também viajavam a uma velocidade cada vez maior. A velocidade máxima média do automóvel logo após a virada do século XX era de 48 Km/h [4], reduzida para os padrões atuais, porém rápida para uma época em que a maioria das pessoas no mundo deslocavam-se a passos lentos de uma carruagem puxada por cavalos. Como resultado, o número de acidentes sérios começou a crescer drasticamente. A invenção de Schulze permitiu que os motoristas observassem exatamente a velocidade com que estavam viajando e fizessem os ajustes necessários.


KAISERLICHES PATENTAMT

PATENTSCHRIFT

– № 146134 –

KLASSE 42 n.

OTTO SCHULZE in STRASZBURG i. E.

34.

Figura 2.3 – Patente de Otto Schulze, datada 7 de Outubro 1902, Adaptada [14].

Antes de ser alcançada a atual tecnologia existente nos veículos atuais, todo o acionamento e funcionamento do *Cluster* era feito de forma puramente mecânica, através de cabos de aço flexíveis (Figura 2.4). Na maioria dos veículos, o cabo flexível do tacômetro era ligado diretamente no cabeçote do motor, e o cabo do velocímetro era ligado na caixa de transmissão. As outras extremidades dos cabos eram conectadas diretamente no painel de instrumentos.

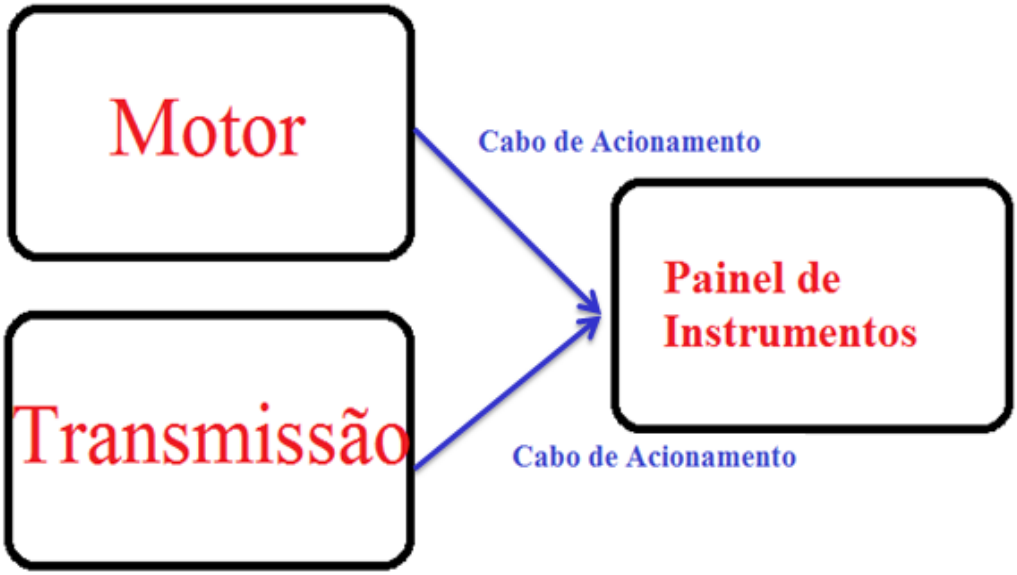


Figura 2.4 – Princípio de funcionamento do velocímetro e tacômetro mecânico.

O cálculo da velocidade de um veículo é baseado na velocidade rotacional das rodas ou da transmissão, essa medição compete ao cabo de acionamento. O cabo de acionamento consiste uma capa composta por várias molas helicoidais, sobrepostas e enroladas firmemente ao redor de uma haste central, ou mandril. Devido a sua estrutura, o cabo de acionamento é bastante flexível e pode ser inclinado, sem que se quebre, a um raio crítico. Isso é bem prático, pois o cabo deve seguir da transmissão ao conjunto de instrumentos, onde fica o velocímetro. O cabo é conectado a um conjunto de engrenagens na transmissão, para que, quando o veículo andar, as engrenagens girem o mandril dentro do eixo flexível. O mandril, então, transfere a velocidade rotacional da transmissão por todo o cabo ao "terminal de trabalho" do velocímetro, onde realmente é feita a medição da velocidade.

2.1.2 – Evolução do velocímetro para o painel de instrumentos

A partir de 1930, a produção de velocímetros subiu de forma constante. Nessa época, também, houve um desenvolvimento que se manteve até hoje com uma característica padrão. Esta unidade é estrategicamente posicionada na linha direta de visão do motorista.

Os primeiros conjuntos de instrumentos consistiam de painéis de aço tratado que serviram de placas de montagem dos instrumentos individuais instalados por trás, como mostra a Figura 2.5 [14]. No entanto, eles só podem ser chamados de painel de instrumentos no sentido moderno depois da Segunda Guerra Mundial.

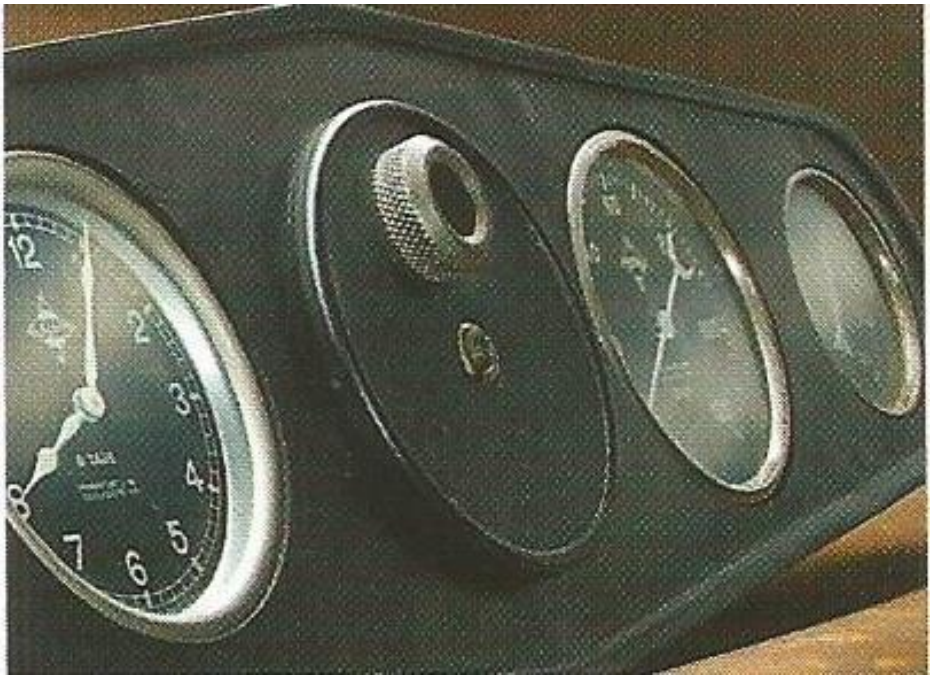


Figura 2.5 – Início do conjunto do painel de instrumentos (1925) [14].

Conteúdos de informação típicos desses conjuntos do instrumento foram acrescentados, além do velocímetro, medidor de combustível, medidor de temperatura do líquido de arrefecimento, bem como dos indicadores e luzes de advertência, como setas e indicador de carga da bateria. Inicialmente os conjuntos de instrumentos tinham um design modular. Velocímetro, medidor de combustível e outras funções de exibição foram concebidos como módulos que foram combinados para formar o painel de instrumentos. Este conceito só foi superado a partir de 1990 com a construção integrada do painel de instrumentos (Figura 2.6).

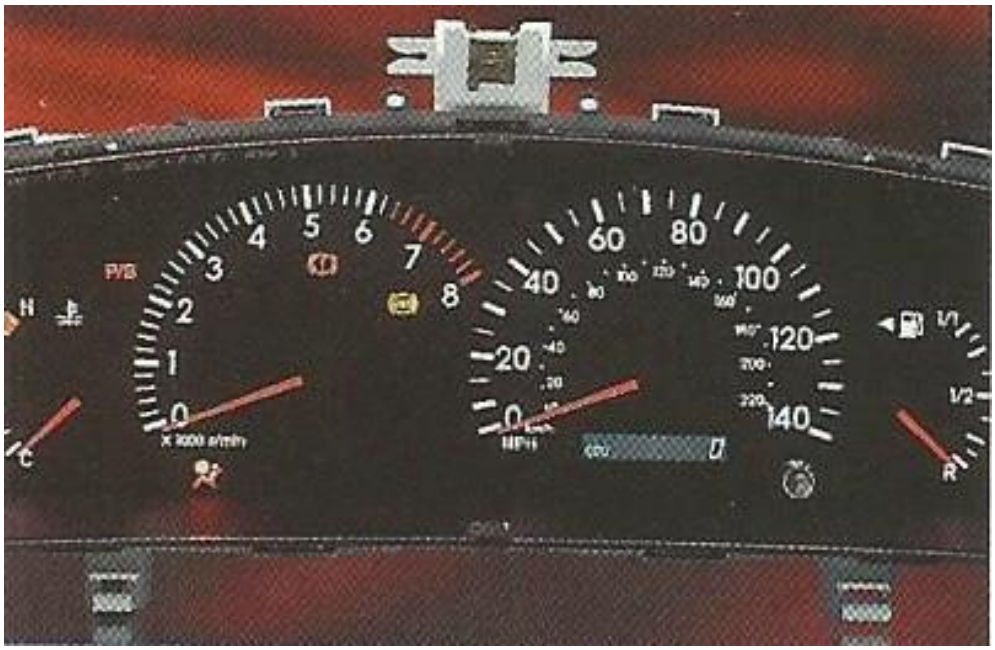


Figura 2.6 – Painel de instrumentos preto do Corolla Toyota [14].

2.1.3 – Tendência do painel de instrumentos digital

As coisas começaram a se mover para o desenvolvimento de sistemas de visualização. Embora a visão familiar de instrumentos redondos analógicos ainda seja encontrada praticamente em todos os lugares, esta primeira impressão é enganosa. Grandes avanços têm sido feitos, tanto na iluminação quanto no tipo de apresentação adequada. Talvez a tendência mais importante seja que o instrumento clássico em alguns casos, os *displays* complementam as escalas analógicas no painel de instrumentos. E estas exposições estão se tornando cada vez maiores e assumindo mais e mais funções, com monitores coloridos de alta resolução.

Estes *displays* coloridos de alta resolução foram inicialmente empregados na exibição para sistemas de navegação no *cockpit*. Mas os seus benefícios estão sendo cada vez mais utilizado no painel de instrumentos. A segunda geração do Audi A8 (1994) marcou o início deste desenvolvimento com um *display* colorido. Mas esta possibilidade de exibir as cores não é a única característica de bons gráficos. Um display gráfico de alta qualidade é obtido com a alta resolução, ou seja, pixels. Esse desenvolvimento pode continuar e é mostrado, por exemplo, nas tentativas iniciais no setor de veículos comerciais, por exemplo, a integração de um

transistor de película fina (TFT) monitor com 5 - polegadas tela diagonal no IVECO Stralis, como mostra a Figura 2.7.

Figura 2.7 – Painel de instrumentos digital do IVECO Stralis [14].

2.2 O Protocolo de Comunicação CAN

A eletrônica embarcada é um item crescente e é responsável por média 30% dos custos dos "veículos atuais" e 90% das inovações estão ligadas a sistemas eletrônicos, conforme é possível de ser observada pelo gráfico da Figura 2.8 a seguir.

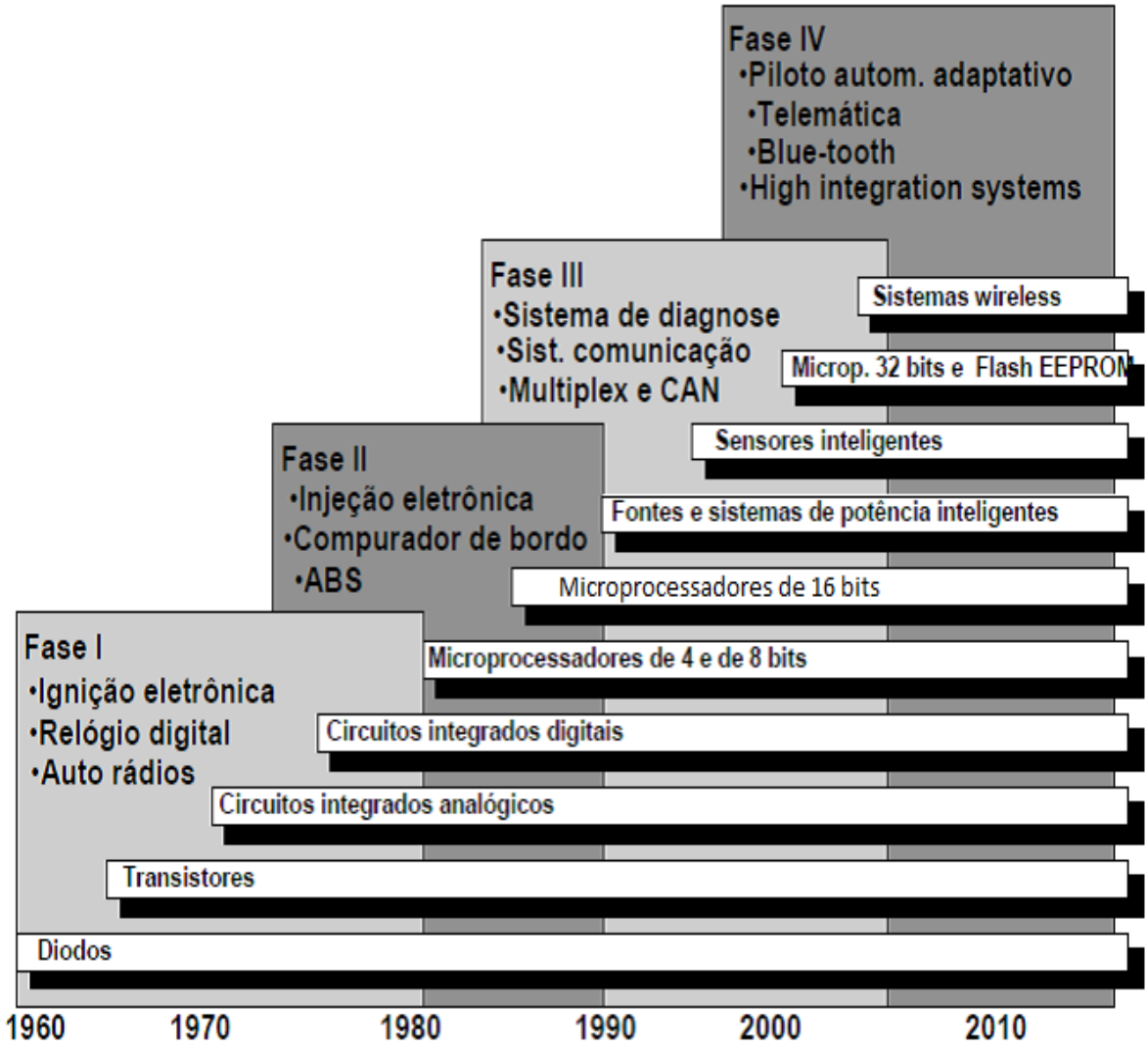


Figura 2.8 – Evolução da eletrônica embarcada [7].

No início da década de 1980, a Bosch começou a desenvolver um sistema de comunicação em série. Foi dado o nome de CAN (*Controller Area Network*) [8], utilizado até hoje nas áreas de trem de força (motor e transmissão), freios, chassis, e conveniência integrando-os em uma rede digital. A cima de tudo, a rede CAN é caracterizada pela transmissão de dados muito confiável que satisfaz os requisitos de tempo real, permitindo também uma significativa redução de cabeamento (Figura 2.9).

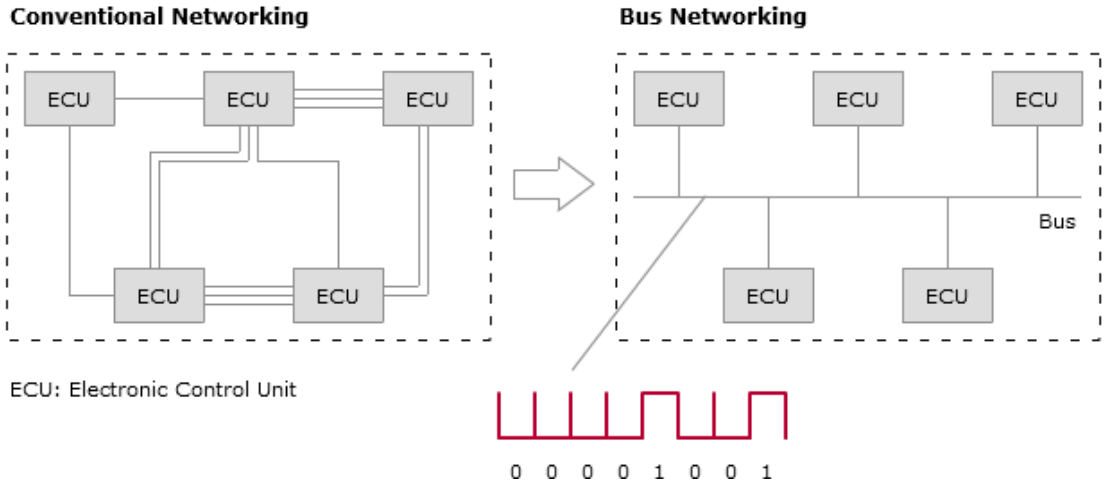


Figura 2.9 – Diagrama em blocos do protocolo CAN [8].

Uma rede CAN consiste de uma série de nós CAN que estão ligados através de um meio de transmissão físico. A rede CAN baseia-se normalmente uma topologia em linha com um canal linear, ao qual o numero de unidades de controles eletrônicos está interligada através de uma interface CAN. Nas extremidades da rede, resistores de terminação contribuem para a prevenção de fenômenos transitórios (reflexos), onde a impedância da rede tem que ser de 62,5Ω, como mostra a Figura 2.10.

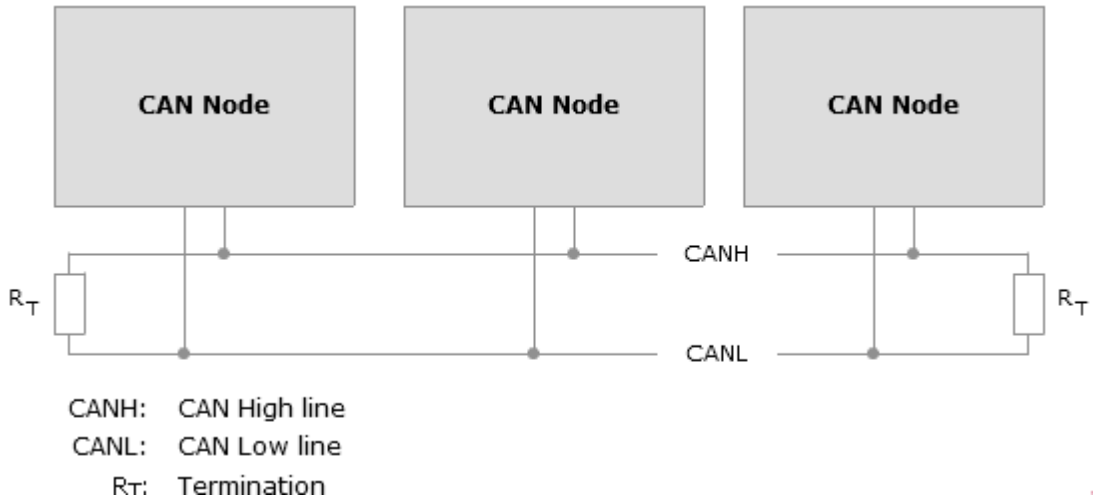


Figura 2.10 – Resistores de terminação [8].

Frames de dados assumem um papel predominante em uma rede CAN. A estrutura de dados é composta de muitos componentes diferentes. A transmissão de um *frame* de dados começa

com o bit de inicio SOF (*Start Of Frame*). É transmitido pelo módulo transmissor como um nível dominante (0), nível que é usado para sincronizar a totalidade da rede. Seguindo o SOF é identificador (ID). Isto define a prioridade do frame de dados, e em conjunto com a filtragem de aceitação prevê relações entre emissor-receptor na rede CAN. Em seguida, o bit RTR ou Pedido de Transmissão Remota é usado pelo remetente para informar os receptores do tipo de frame de dados. O bit IDE (Identificador Bit de Extensão), serve para distinguir entre o formato padrão e o formato estendido. No formato padrão o identificador tem 11 bits, e em formato estendido 29 bits.

O DLC (Comprimento dos Dados Código) transmite o número de bytes de carga útil para os receptores. Os bytes de carga útil são transportados no campo de dados. Um máximo de oito *bytes* pode ser transportado numa estrutura de dados. A carga é protegida por uma soma de verificação com uma verificação de redundância cíclica (CRC), que está encerrado por um bit de delimitador. Com base nos resultados de CRC, os receptores devem reconhecer positivamente ou negativamente na ranhura ACK (reconhecimento) que também é seguida de um delimitador. A transmissão de um frame de dados é terminada por sete bits recessivos EOF (Fim do Quadro). A Figura 2.11 apresenta os *frames* de dados de uma mensagem CAN.

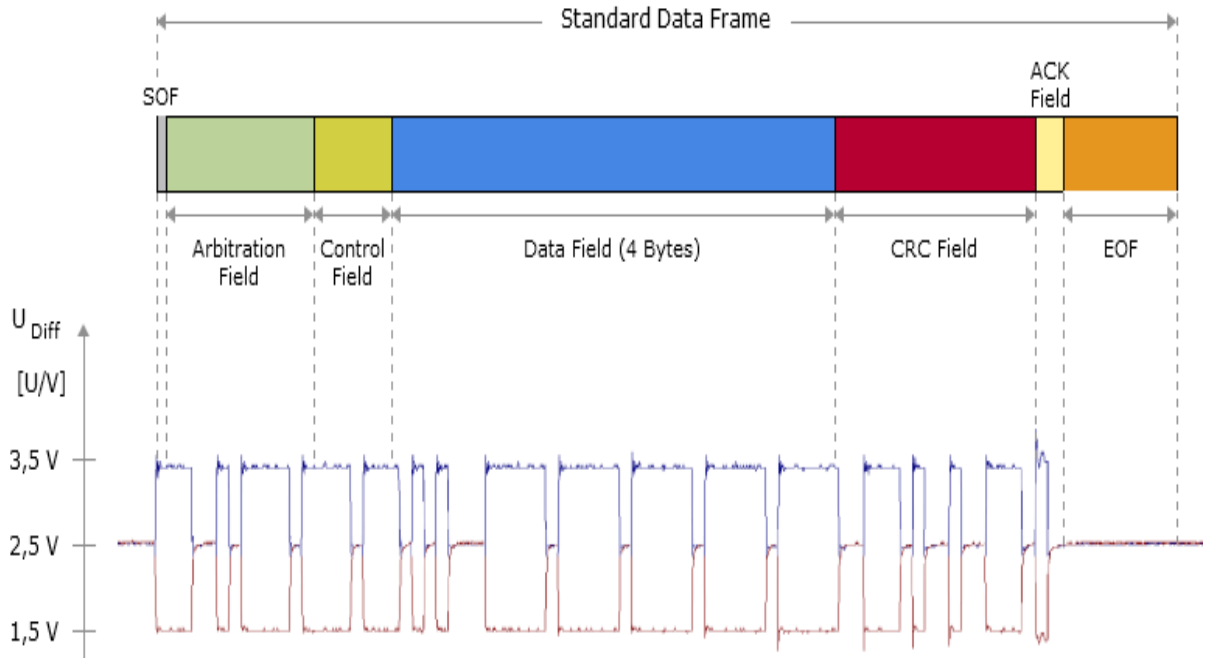


Figura 2.11 – Mensagem CAN com ID de 11 bits [8].

O principal objetivo de uma rede CAN é a troca de dados entre os módulos eletrônicos. Isso viabiliza o desenvolvimento de sistemas eletrônicos, pois um grande número de dados é disponibilizado na rede e pode ser facilmente utilizado por outros sistemas, diminuindo a necessidade de novos sensores já existentes no veiculo. Para o desenvolvimento desta customização do painel de instrumentos, as informações de velocidade e rotação são coletadas da rede CAN sem a necessidade de instalação de novos sensores.

2.3 A Escolha da Ferramenta Gráfica

A principal atividade da customização do painel de instrumentos está relacionada ao desenvolvimento da interface gráfica. Esse é o principal resultado do projeto, que diretamente desperta a percepção do cliente, ou seja, do usuário do veículo.

Existem diversas ferramentas para desenvolvimento de trabalhos gráficos, a escolha da ferramenta ideal para realizar o *design* do painel de instrumentos depende de vários fatores, como custo, facilidade de utilização, quantidade de recursos, disponibilidade, entre outros. Para o desenvolvimento desse projeto que o objetivo é apresentar apenas o conceito de customização do painel de instrumentos, utilizamos a ferramenta LabVIEW da National Instruments, disponível da FATEC.

O *software* LabVIEW – Laboratory Virtual Instruments Engineering Workbench é uma linguagem gráfica de programação (linguagem G) criada pela National Instruments, que utiliza ícones ao invés de linhas de texto para criar as aplicações. Em contraste com as linguagens de programação baseadas em texto, o LabVIEW usa fluxo de dados (*dataflow*) para determinar a forma de execução.

Sua forma de programação é altamente produtiva e propicia a construção de sistemas voltados para aquisição, análise e apresentação de dados, conforme o fluxo ilustrado na Figura 2.12.

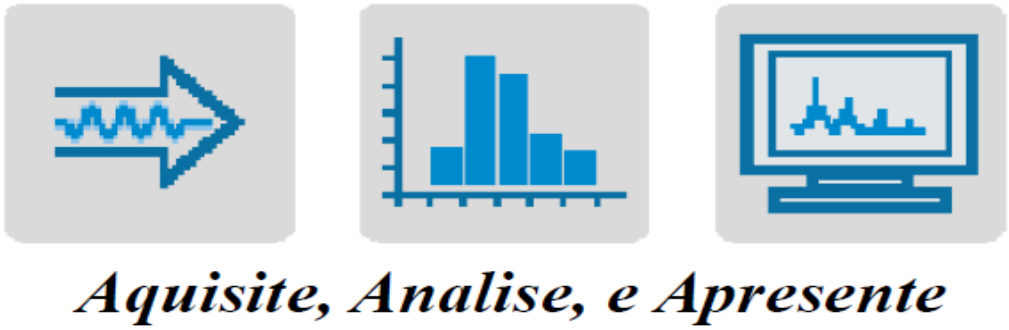


Figura 2.12 – LabVIEW – Aquisição, análise e apresentação de dados [9].

Os programas em LabVIEW possuem extensões chamadas de VI (*Virtual Instruments*) e SubVIs (sub rotinas de programação).

As VIs fornecem duas interfaces: uma interface para o fluxo de dados que é o código fonte chamado de Diagrama de Blocos, onde se desenvolve toda a lógica do *software* e podemos associá-lo á um fluxograma (A Figura 2.13 detalha essa interface.). O Painel Frontal que também permite a programação, a visualização de variáveis e interação do usuário com o programa.

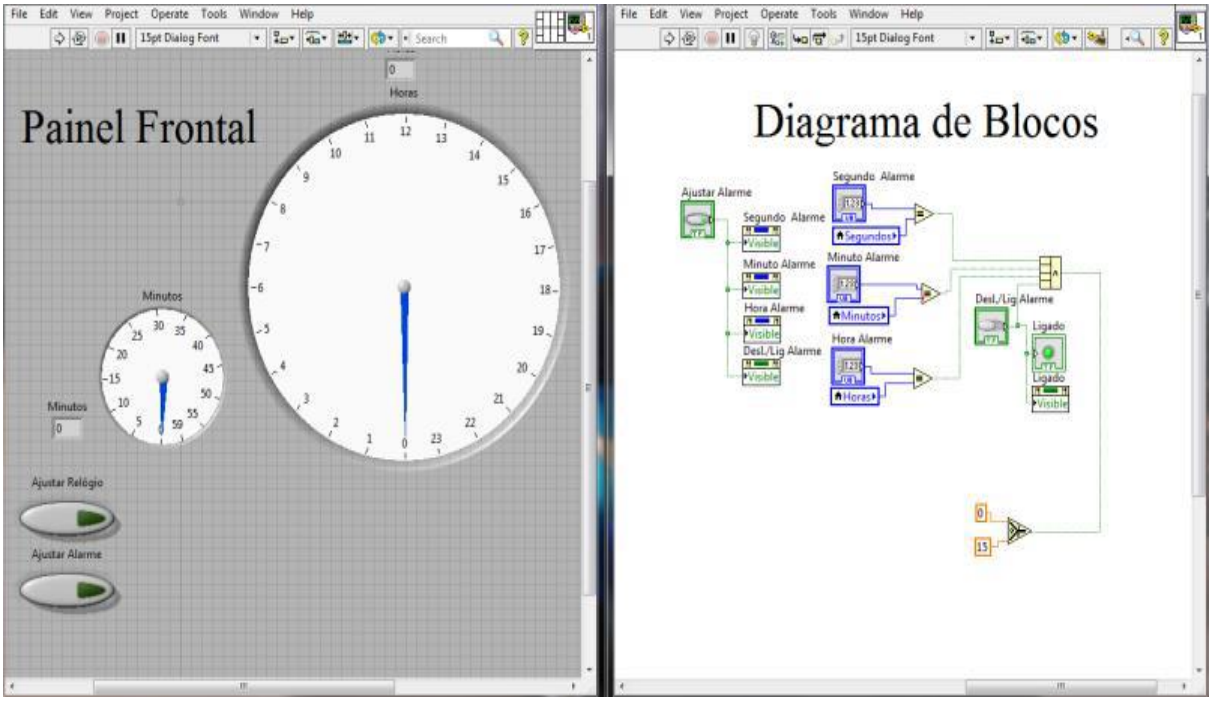


Figura 2.13 – Painel frontal e diagrama de blocos do LabVIEW.

Seu modo de operação possui uma lógica peculiar que é o Fluxo de Dados, na qual, diferentemente do conceito de fluxo linear da Linguagem C, as funções executam em paralelo ("da esquerda para a direita da tela") com a dependência apenas do fluxo de dados entre as funções.

- SubVIs são sub-rotinas do LabVIEW que executam trechos repetidos, muito utilizados ou funções específicas (ex. Leitura da porta serial do computador). Com isso temos um programa mais robusto resultando em uma melhor visualização.
- Polimorfismo é uma característica do LabVIEW, onde o programa consegue realizar funções/equações mesmo se as variáveis de entrada possuírem precisões (extensões) diferentes. Por default, o resultado terá sempre a extensão de maior precisão.

3. DESENVOLVIMENTO DO PROJETO

A proposta do presente trabalho é de apresentar um conceito de painel de instrumentos digital e personalizado de acordo com o perfil de cada usuário, proporcionar um ambiente dinâmico, utilizando os dados disponíveis na rede de comunicação CAN do automóvel. Como existe uma grande variedade de estilos de painel de instrumentos e perfis de usuários, no modelo atual não é possível satisfazer todos os clientes, pois para isso seria necessário desenvolver vários painéis diferentes, envolvendo o hardware e software. No conceito de customização, teremos um único hardware com uma tela gráfica de cristal líquido, LED ou outro tipo de tecnologia, que seja possível demonstrar o painel de instrumentos digitalmente.

3.1 Aquisição de Dados

Para o modelo proposto de customização, as informações do veículo, como, por exemplo, rotação do motor e velocidade, são coletadas da rede CAN do automóvel. Para apresentar o conceito, utilizamos o veículo Volkswagen Polo 2004 da Fatec Santo André, como mostra a Figura 3.1.

Figura 3.1 – VW Polo da FATEC Santo André.

Para coletar as informações via CAN do automóvel, foi preciso fazer uma adaptação no chicote elétrico para conectar os cabos CAN High e CAN Low na placa didática.

O VW Polo fornecido pela Faculdade possui uma rede de comunicação CAN de alta velocidade com taxa de transmissão de 500 Kbps. No painel de instrumentos deste veículo, possui tacômetro eletrônico com máxima de 7500 RPM, e o velocímetro eletrônico com máxima de 260 KM/h. Como mostra a Figura 3.2.

Figura 3.2 – Painel de instrumentos do VW Polo.

As informações que alimentarão o painel de instrumentos estão disponíveis na linha CAN do veículo, como por exemplo, velocidade de veículo e rotação do motor. Existem dois tipos de dados na rede CAN, os privados desenvolvidos especificamente para cada empresa e os abertos protocolos banco de dados conhecidos. O veículo utilizado para aplicação deste conceito de painel de instrumento possui uma rede CAN privada.

Para identificação dos dados em uma rede CAN privada, é necessário utilizarmos uma metodologia para descobrir essas informações.

Procedimento utilizado para interpretar os dados da rede CAN:

- Conectamos o hardware VN 1630 da Vector, como mostra a Figura 3.3.

Figura 3.3 – Aquisição de dados com o aparelho VN1630.

- Conectamos um *software* da empresa Vector chamado de CANOE no barramento CAN do veículo;

- O *software* disponibiliza todos as dados que estão trafegando na rede CAN na tela do computador, conforme Figura 3.4.

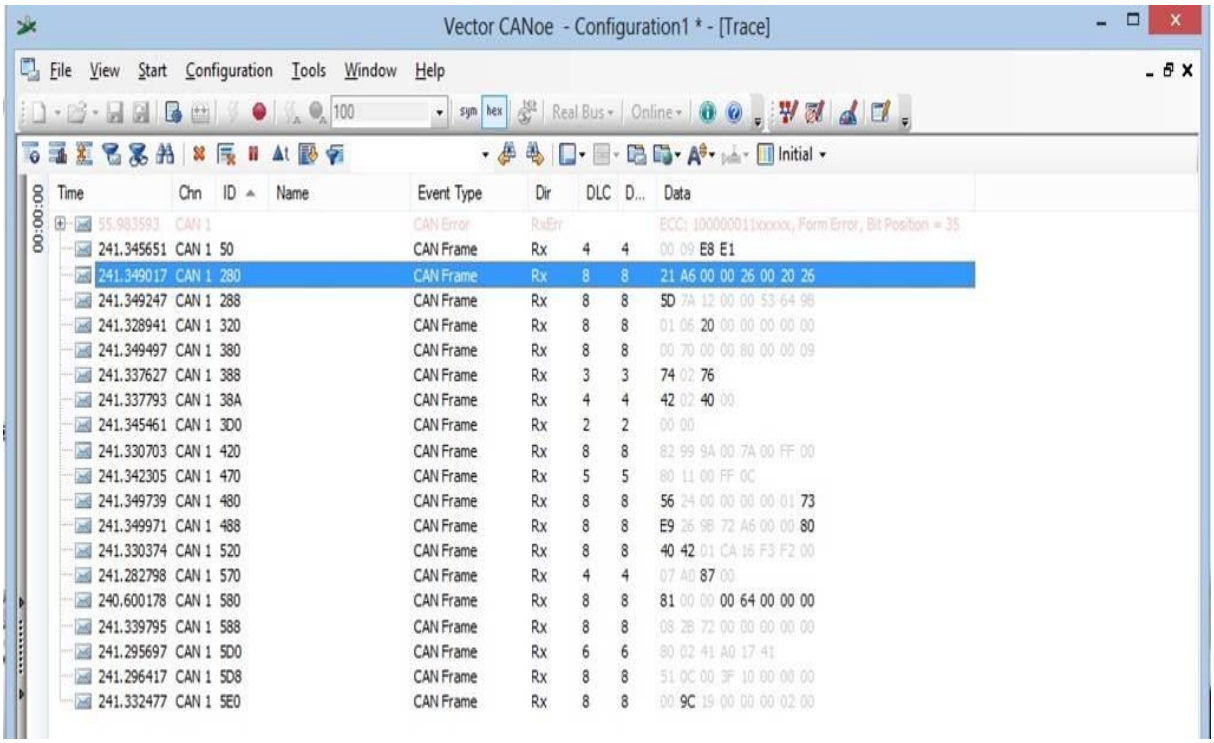


Figura 3.4 – Software CANOE da Vector.

As colunas mais importantes para o nosso monitoramento são:

- ID – identificador da mensagem

- DLC – quantidade de numero de byte de dados

- Data – as informação correspondente a cada um dos seus Ids, ou seja, cada uma das mensagens.

Para descobrir qual era a mensagem de rotação do motor, foi ligada a ignição do veículo, neste instante as mensagens da rede CAN começam a trafegar, registramos as mensagens em que os *bytes* de dados tinham seus valores em zero, pois o motor desligado o valor da rotação permanece nula, foi ligado o motor e monitorado quais das mensagens registradas alteram seus valores, as mensagens que não tiveram, sobrando poucas mensagens para identificarmos, variamos a rotação do motor através do acionamento do pedal de acelerador e mensagem que sofreu alteração é a mensagem da rotação.

Para descobrimos a escala da mensagem de rotação, anotamos o valor com o veículo em marcha lenta, aumentamos a rotação do motor para mil rotações por minutos e anotamos o valor correspondente da mensagem CAN, conforme Tabela 1, através de um cálculo

descobrimos a escala da rotação do motor, ou seja, os dois bytes dados da rotação recebidos da mensagem CAN será unidos e divido por 4 (Estratégia adotada pela montadora).

Tabela 3.1 – Dados coletados e tratados da mensagem CAN

ID		Receptor	DLC	Campo de Dados				
Decimal	Hexadecimal							
640	280	Rx	8	21	18	8C	0C	15 00 1B 16 Length
648	288	Rx	8	5D	9B	12	0	00 51 6B 18 Length
896	380	Rx	8	0	79	0	0	80 00 00 00 Length
1416	588	Rx	8	8	7A	72	0	00 00 00 00 Length
1160	488	Rx	8	AA	16	18	72	A6 00 00 70 Length
800	320	Rx	8	5	2	20	0	00 00 00 00 Length
640	280	Rx	8	21	18	8C	0C	15 00 1B 15 Length
648	288	Rx	8	5D	9B	12	0	00 51 6B 18 Length
896	380	Rx	8	0	79	0	0	80 00 00 00 Length
1152	480	Rx	8	56	4	B0	10	00 00 02 F0 Length
1160	488	Rx	8	5A	16	18	72	A6 00 00 80 Length
640	280	Rx	8	21	18	8C	0C	15 00 1B 15 Length
648	288	Rx	8	8F	9B	12	0	00 51 6B 18 Length
896	380	Rx	8	0	79	0	0	80 00 00 00 Length
1416	588	Rx	8	8	7A	72	0	00 00 00 00 Length
	RPM							
	Velocidade							
	Byte Correspondente							

Todos os dados da rede CAN podem ser descobertos, através de rotinas especificas para cada tipo de informação, exemplo velocidade, temperatura, etc., conforme mostrado o exemplo da rotação do motor.

3.2 Hardware Utilizado

No mercado automotivo existem aparelhos capazes de fazer aquisição e tratamento de dados via CAN e demonstrá-los em uma plataforma gráfica, como por exemplo, o NI 9862 da National Instruments (Figura 3.5), porém possuem custos elevados.

Figura 3.5 – Aparelho da NI 9862.

Devido ao custo e o tempo de desenvolvimento do conceito, foi utilizada a placa didática Kit CAN da FATEC Santo André, para fazer a aquisição e tratar os dados da rede CAN, como mostra a Figura 3.6.

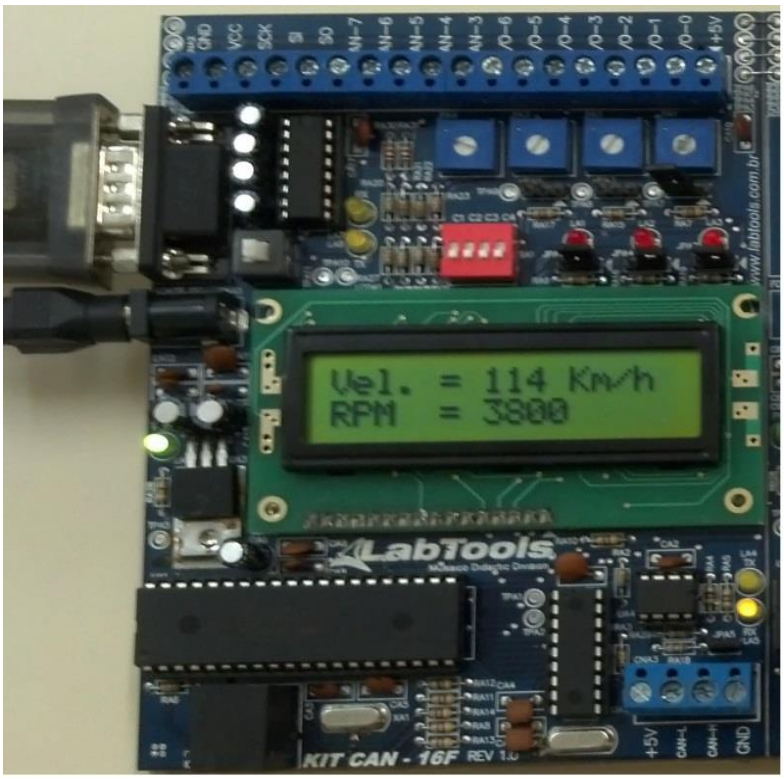


Figura 3.6 – Placa didática Kit CAN da FATEC Santo André.

A placa didática é composta por um transmissor e receptor CAN PCA82C251, um controlador CAN MCP2515, microcontrolador PIC 16F877A, transmissor e receptor serial RS-232 MAX232D e um LCD 16x2.

3.3 Software Desenvolvido em Linguagem C

Com a necessidade de coletar e tratar os dados de rotação e velocidade do veículo via CAN e transmitir para uma plataforma gráfica, foi preciso desenvolver um *software* em linguagem C que filtrasse as mensagens que possuíam dados de rotação e velocidade. Os dados coletados passaram a ser transmitidos para uma plataforma gráfica via serial RS-232.

O programa desenvolvido em linguagem C para a aquisição e tratamento dos dados de rotação e velocidade, possui o fluxograma, como apresenta a Figura 3.7.

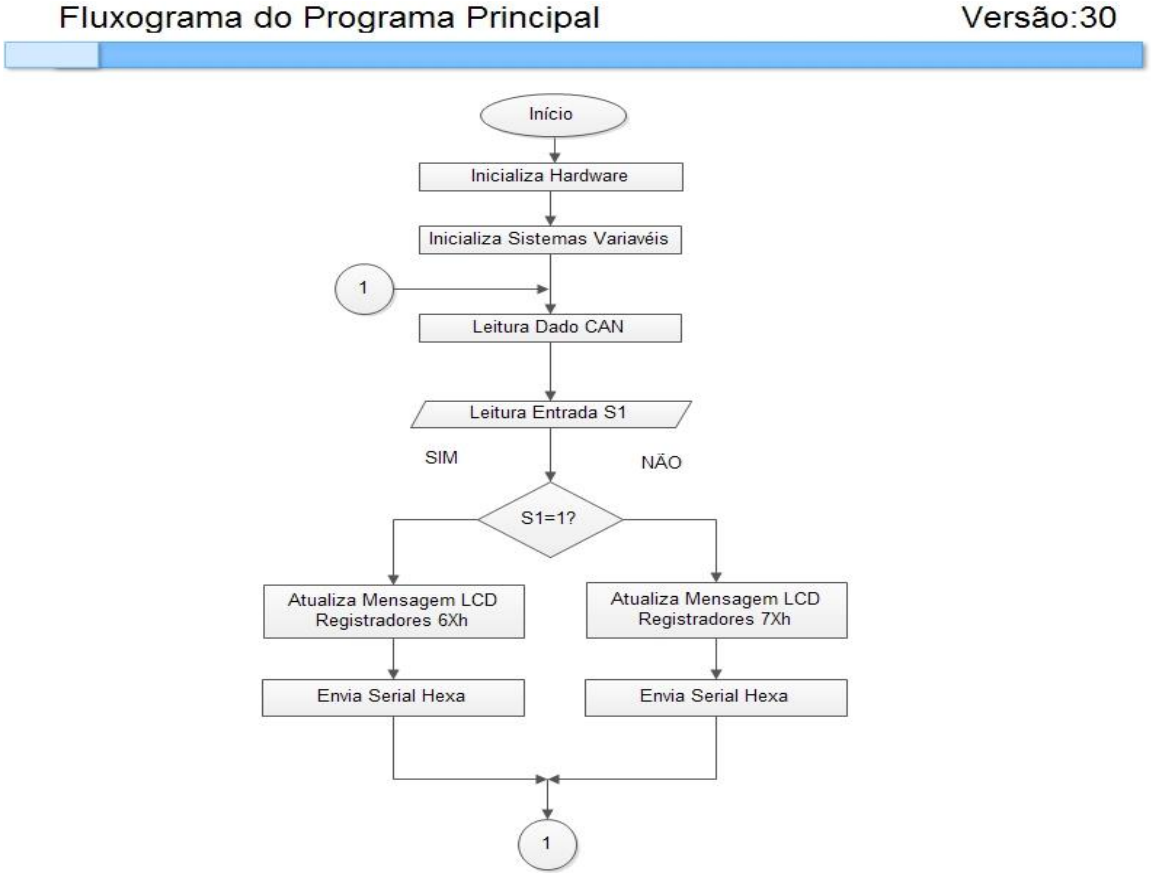


Figura 3.7 – Fluxograma do programa principal em linguagem C.

O *software* desenvolvido em linguagem C foi separado em partes para o programa ter uma maior flexibilidade, são elas:

- ProgramaPrincipal.c;
- InicializaHardware.h;
- ComandosCAN.h;
- Display4bits.h;
- MatrizHexa.h.

O programa "InicializaHardware.h" é composto por funções que configuram o modo de funcionamento do PIC 16F887A, como entradas e saídas, base de tempo (1ms) e comunicação SPI.

O controlador MCP2515 é configurado pela comunicação SPI e a parte do programa em linguagem C responsável é "ComandosCAN.h". A Tabela 3.2 mostra as configurações gerais para o MCP2515 atuar com a mesma taxa de transmissão do VW Polo.

Tabela 3.2 – Configurações gerias do MCP2515

Taxa de Transmissão			500Kbps
Frequência de entrada	foscilação		20MHz
Toscilação	1/ foscilação	1/20MHz	50ns
Tempo de bit	1/taxa de transmissão	1/500Kbps	2us
Tempo Quantum	Tq=2x(BRP+1)xToscilação (BRP=2)	Tq=2x2x50ns	200ns
Quantidade de Tq	Tempo de bit/Tq	Tq=2us/200ns	10
Segmento de Sincronismo			1 Tq
Segmento de Propagação			1 Tq
Segmento de Fase 1			4 Tq's
Segmento de Fase 2			4 Tq's
ID da Rotação (RPM)	640	280h	10100000000
ID da Velocidade	800	320h	11001000000

Nesse mesmo programa, foi habilitado Máscara e Filtro para receber apenas as mensagens CAN com o ID 280h (rotação do motor) e 320h (velocidade).

Para mostrar os dados recebidos pela CAN, utilizamos um LCD 16x2 utilizando 4 vias de comunicação com o PIC 16F877A. A parte do programa "Display4bits.h" é responsável pela configuração do LCD, e o programa "MatrizHexa.h" é uma matriz que contêm números em

hexadecimal que são mostrados no *display* de acordo com os valores recebidos pela rede CAN.

O "ProgramaPrincipal.c" é composto pelos programas citados anteriormente (Figura 3.8), e a função desse programa é receber os dados da mensagem CAN com ID 280h (rotação do motor) e 320h (Velocidade) e mostrar no LCD os 8 *bytes* de dados de cada mensagem, e enviar para a serial RS-232 somente os dois *bytes* de dados da rotação e um *byte* de dados da velocidade. Com a chave seletora S1, pode-se escolher qual das duas mensagens (280h ou 320h) será mostrada no *display*.

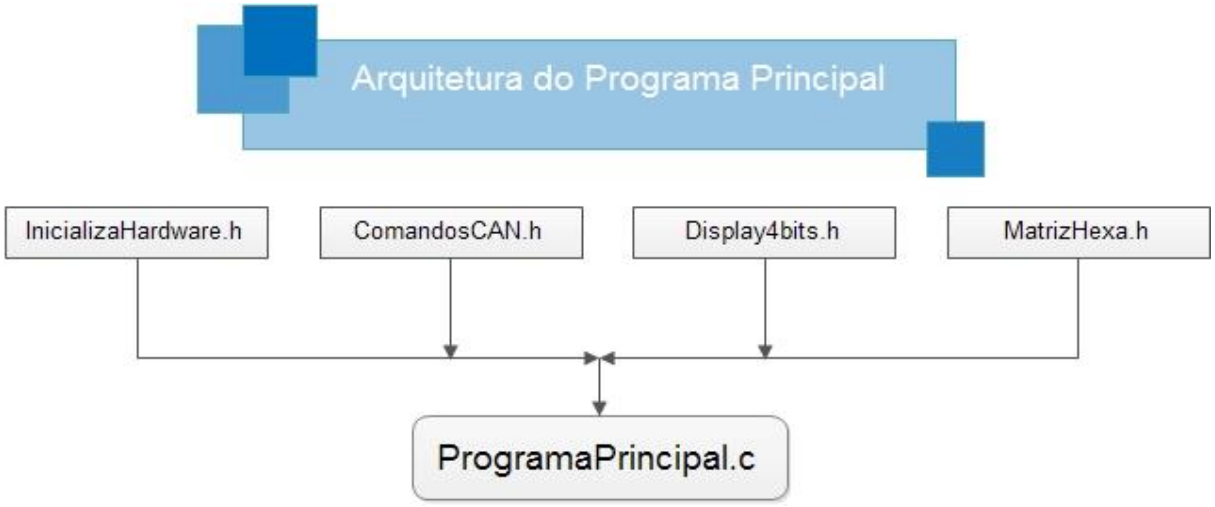


Figura 3.8 – Arquitetura do programa principal.

3.4 Software Desenvolvido em Linguagem G

Utilizando a ferramenta gráfica LabVIEW, a programação da interface gráfica foi desenvolvida no diagrama em blocos e o fluxograma da Figura 3.9, mostra toda a sequência de execução do programa.

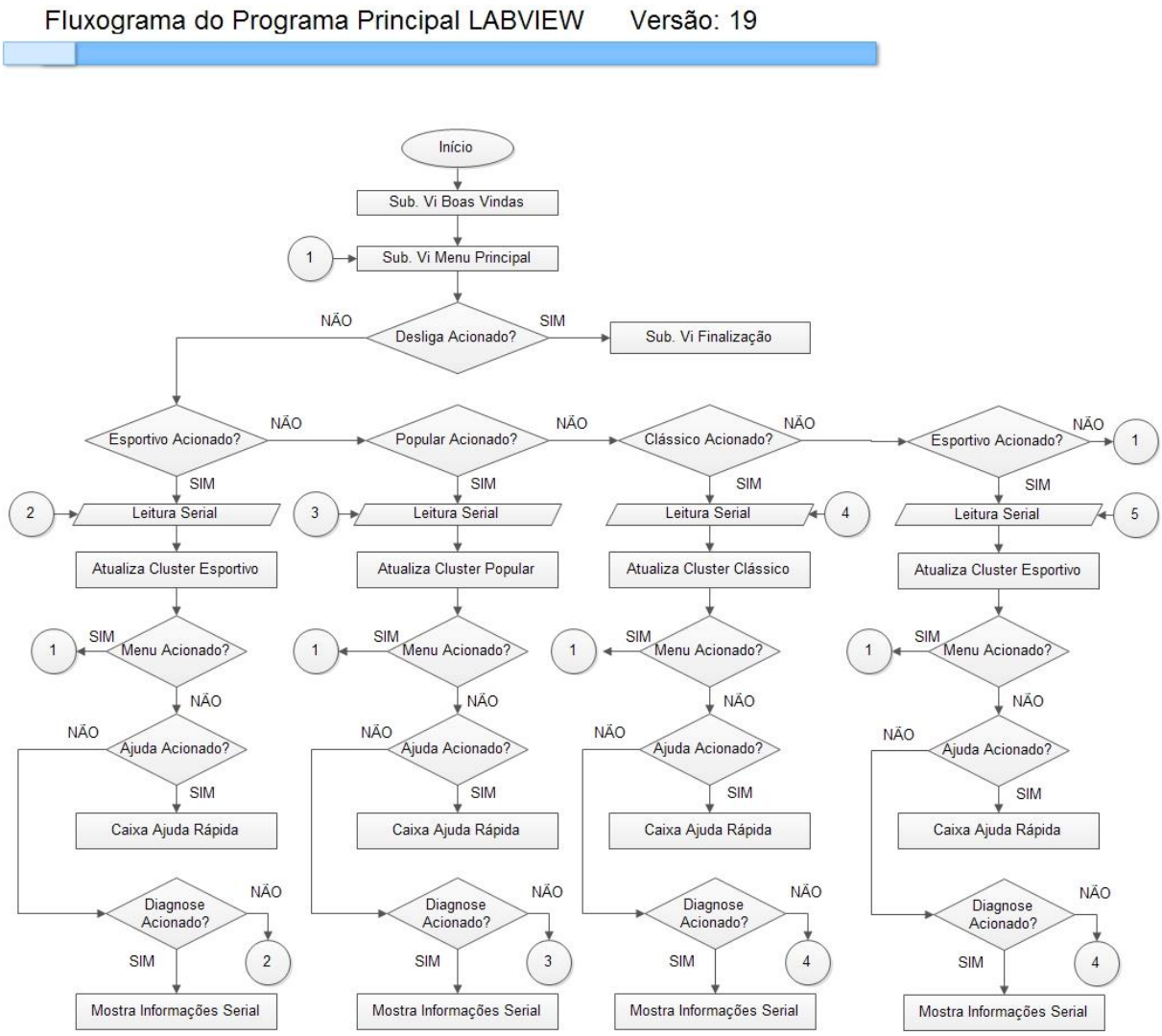


Figura 3.9 – Fluxograma de funcionamento do software desenvolvido em LabVIEW.

O programa possui uma VI principal e oito SubVIs:

- VI_Principal;

-Sub_VI_Bem_Vindos;

- Sub_VI_Menu_Principal;

- Sub_VI_Mercedes;

- Sub_VI_Clio;

- Sub_VI_Fusca;

- Sub_VI_Esportivo;

- Sub_VI_Diplay_7_Segmentos;

- Sub_VI_Finalizacao.

Quando executar a VI_Principal, a Sub_VI_Bem_Vindos será chamada e terá um tempo de amostragem de sete segundos.

Em seguida a Sub_VI_Menu_Principal será executada aguardando a ação do usuário. Caso o usuário pressione o botão “Desligar”, a Sub_VI_Finalizacao será chamada e logo após o sistema desligará.

Nos quatro estilos de painéis de instrumentos desenvolvidos, a lógica de programação em linguagem G foi à mesma, onde utilizou a função VISA para ler dados da Serial RS-232.

A função VISA do software LabVIEW é responsável por receber os dados de rotação e velocidade via serial RS 232. Esses dados são convertidos em *bytes* e no caso da rotação, os *bytes* são concatenados para serem lidos como um único dado divide-se este valor por quatro, e em seguida, divide-se o valor por mil para se chegar aos mesmos parâmetros originais do veículo.

Já para obter-se os parâmetros de velocidade, o dado é multiplicado por 1.5 e o resultado subtrai-se 5, tendo como resultado o valor da velocidade original do veículo, como apresenta a Figura 3.10.

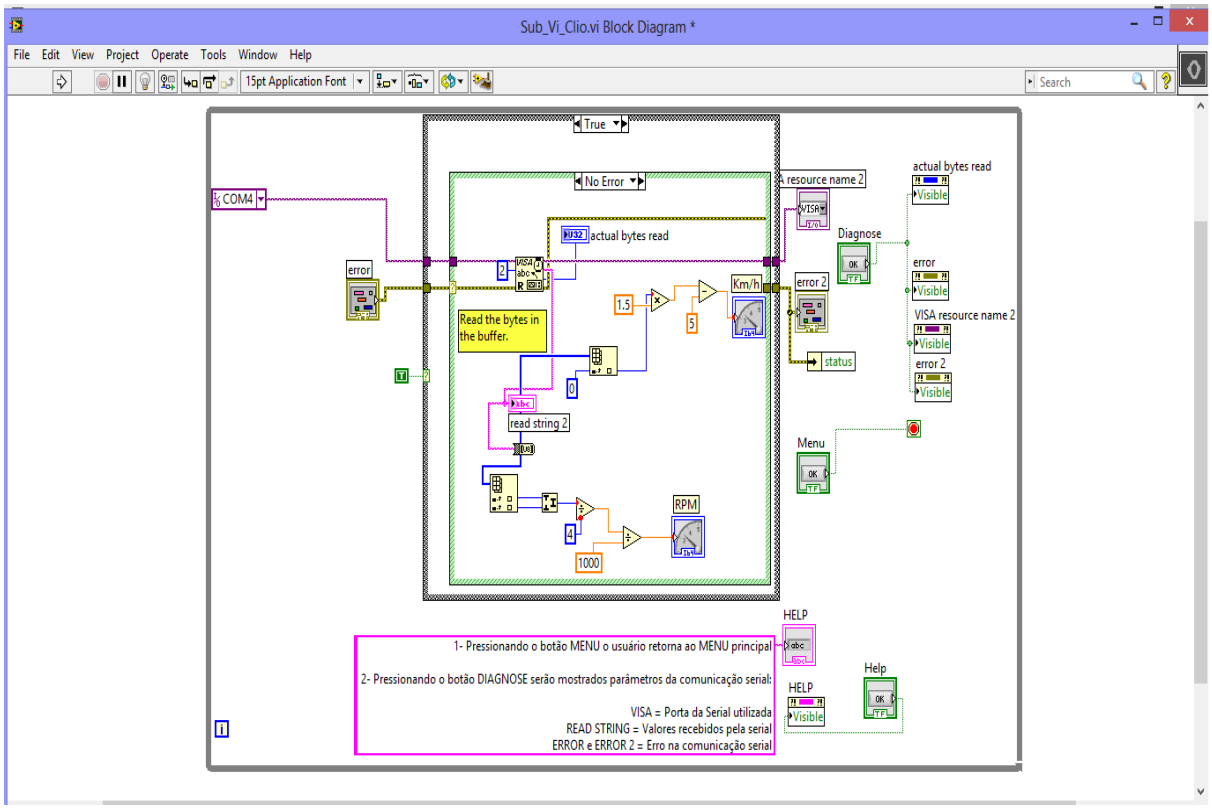


Figura 3.10 – Diagrama em blocos do tratamento de dados da rotação e velocidade.

O painel de instrumento da opção “Esportivo” teve uma modificação na lógica de programação por se tratar de um velocímetro digital, onde foi preciso transformar o dado da velocidade do formato *byte* para um código BCD, como mostra a Figura 3.11.

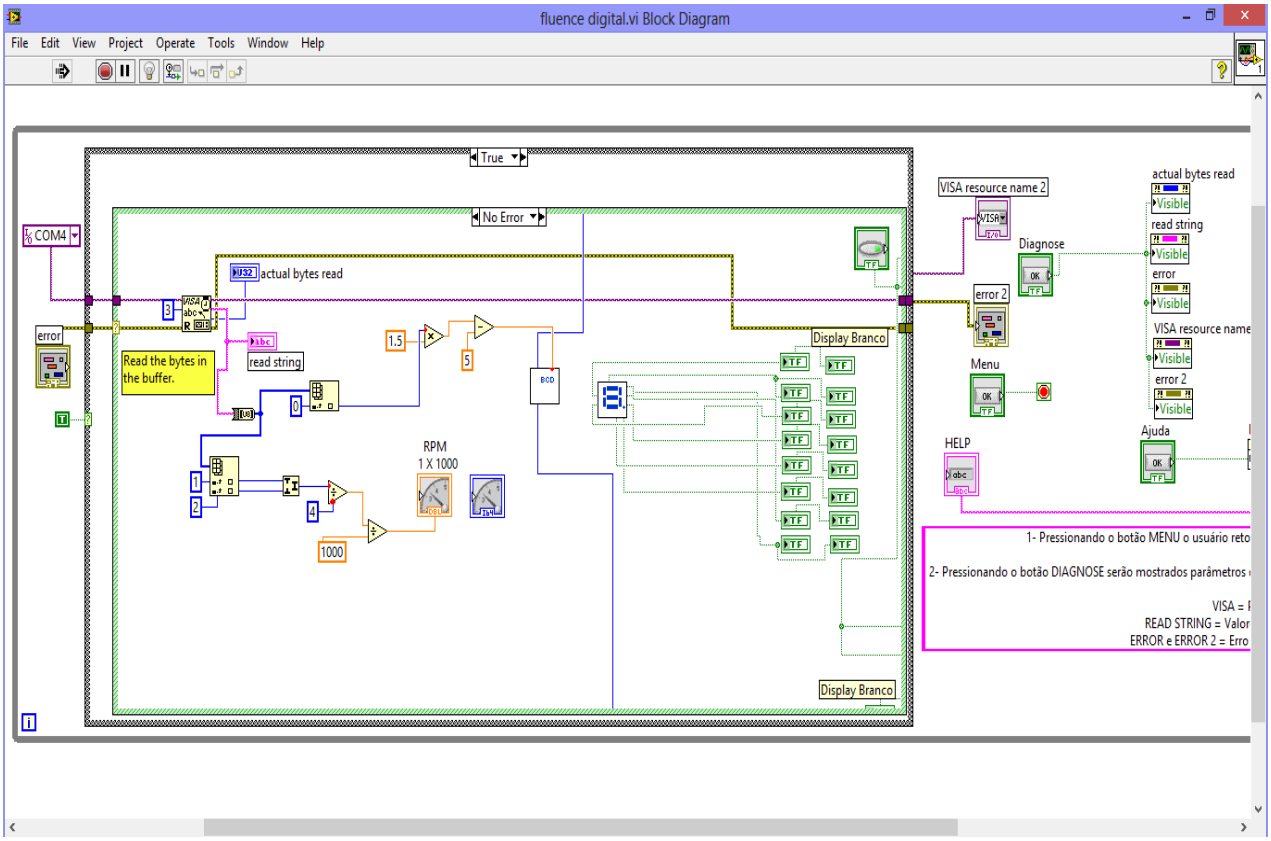


Figura 3.11 – Diagrama em blocos da Sub_VI_Esportivo.

4 – RESULTADOS OBTIDOS

Após obter êxito nos teste em bancada, o conceito de painel de instrumento foi introduzido no veículo VW Polo. (Para simular os parâmetros do veículo em bancada, foi desenvolvido um *software* em linguagem C e utilizada outra placa didática Kit CAN da FATEC Santo André). Para mostrar o conceito do painel de instrumento digital, o VW Polo foi colocado no dinamômetro do laboratório da FATEC Santo André e para demostrar melhor os estilos de

painéis digitais, a imagem foi projetada na TV do próprio laboratório, como mostra a Figura 4.1.

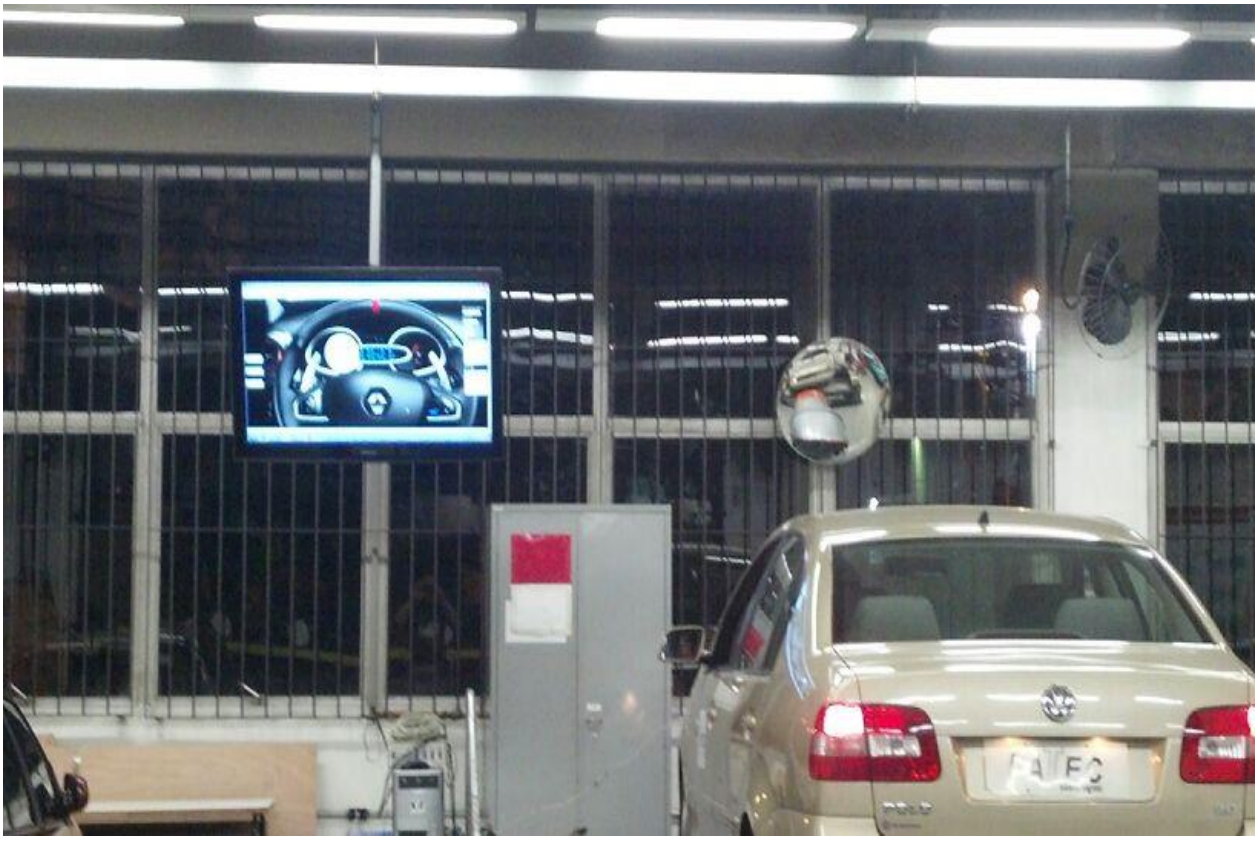


Figura 4.1 – Teste no VW Polo.

Ao iniciar o sistema, será apresentada uma mensagem de “Boas Vindas” (Figura 4.2) para o usuário. Em seguida surgirá o menu principal constituído por botões (opções), onde o usuário poderá escolher o estilo de *Cluster* de sua preferência ou desligar o sistema, como mostra a Figura 4.3.

Figura 4.2 – Tela de "Boas Vindas".

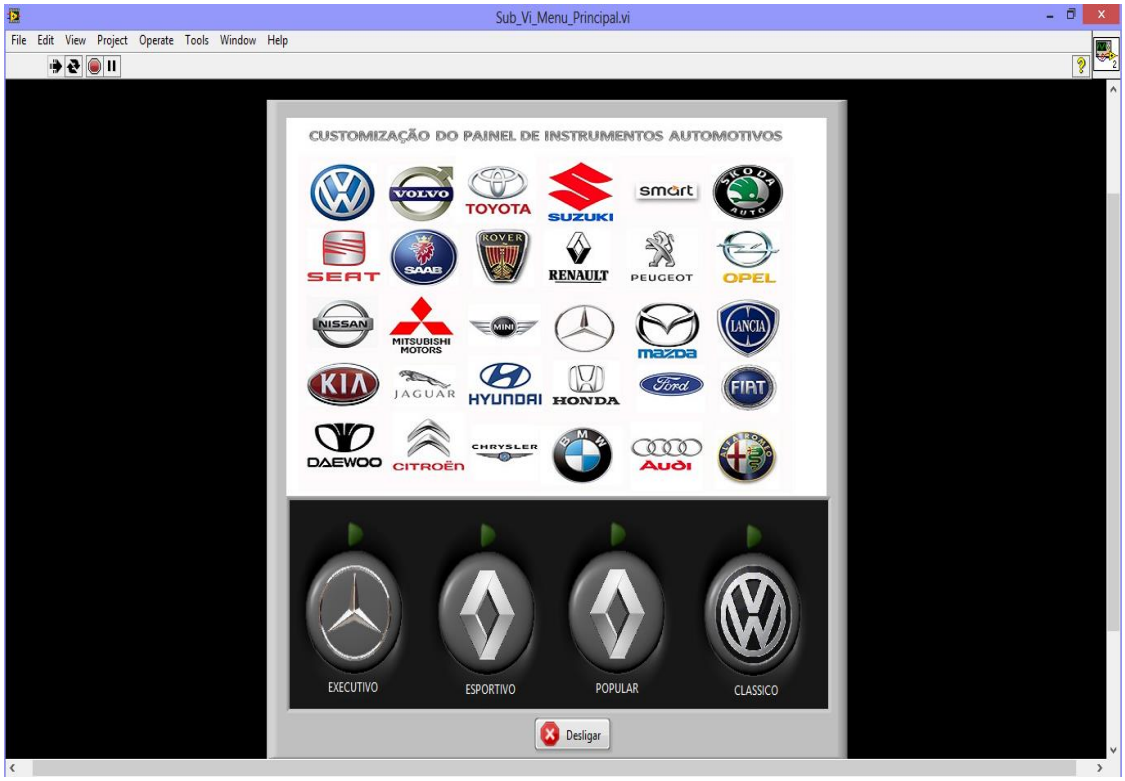


Figura 4.3 – Menu principal do painel de instrumentos.

Ao selecionar a opção “Executivo” mostrará ao usuário um painel de instrumentos do modelo Mercedes Benz E350, como mostra a Figura 4.4

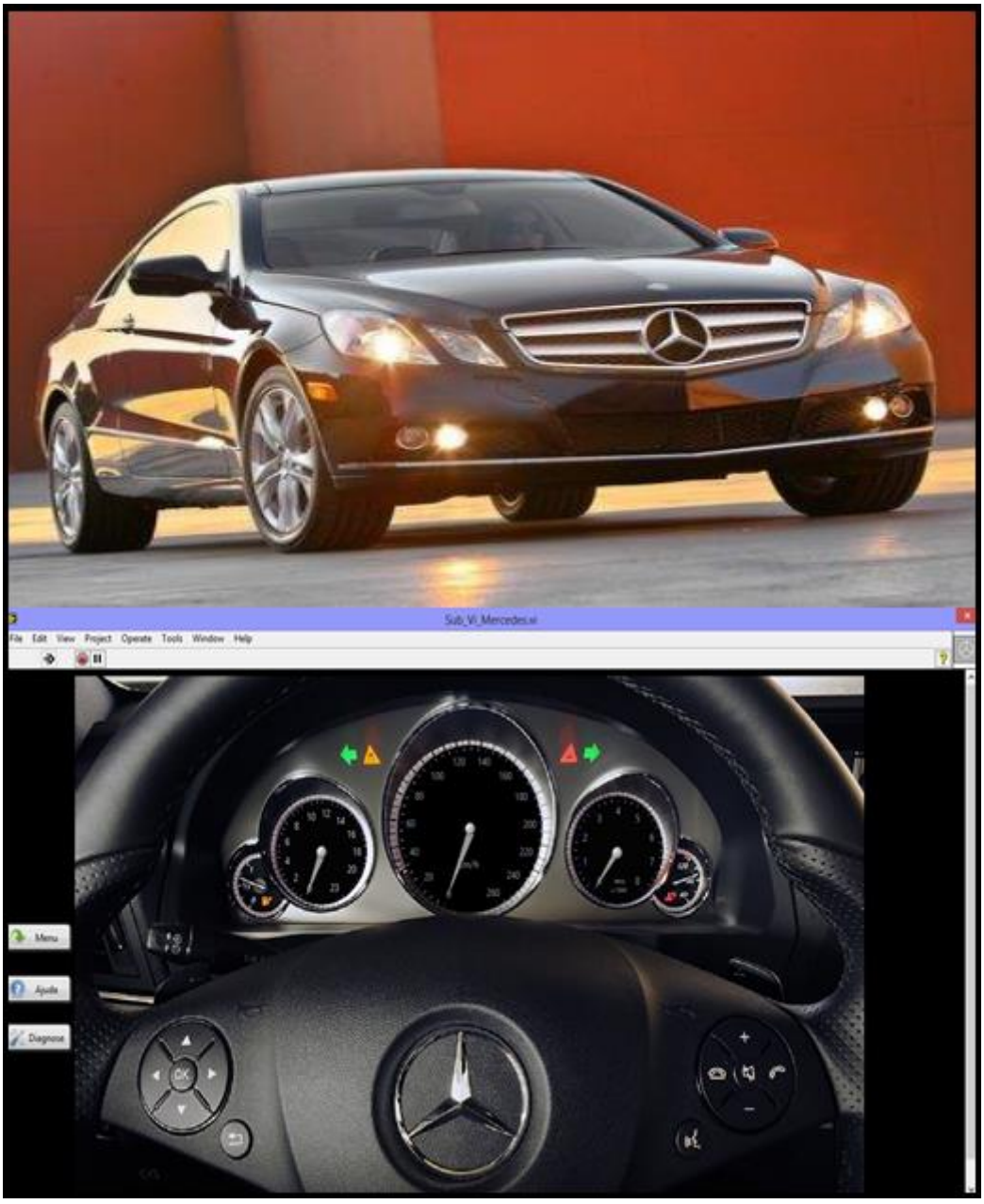


Figura 4.4 – Opção “Executivo”.

Ao selecionar a opção “Esportivo” mostrará ao usuário um painel de instrumentos do Renault modelo Clio RS, como mostra a Figura 4.5. Esse modelo de *Cluster* tem um botão (no volante), onde o usuário poderá alterar a cor dos dígitos do velocímetro (branco ou azul).

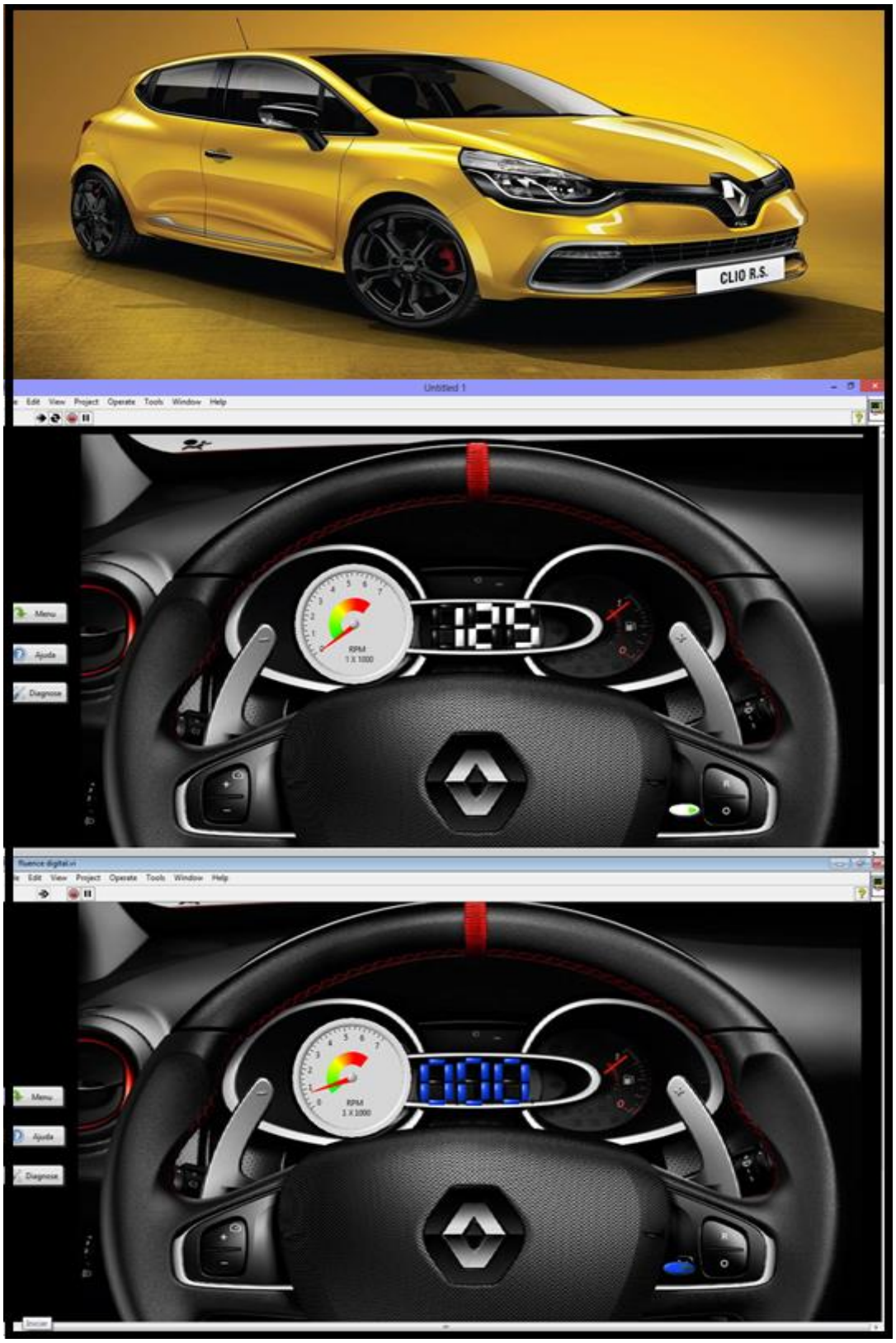


Figura 4.5 – Opção “Esportivo”.

Ao selecionar a opção “Popular” mostrará ao usuário um painel de instrumentos do modelo Renault Clio 2013, como mostra a Figura 4.6.

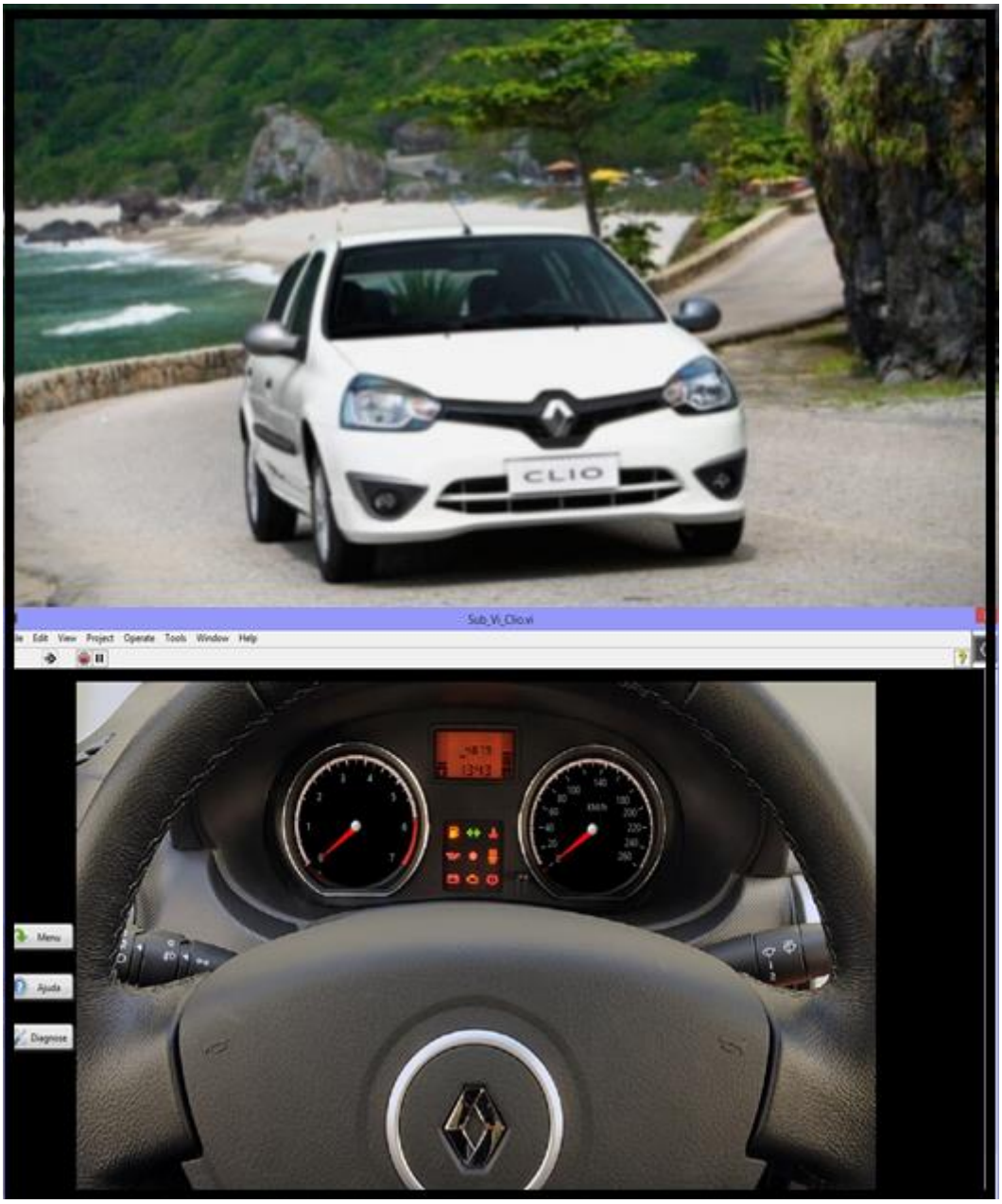


Figura 4.6 – Opção “Popular”.

Ao selecionar a opção “Clássico” mostrará ao usuário um painel de instrumentos do Volkswagen modelo Fusca 1969, como mostra a Figura 4.7.

Figura 4.7 – Opção "Clássico".

Em cada estilo de painel selecionado será mostrado ao usuário (lado esquerdo da tela) uma barra de ferramentas contendo três botões, apresentada pela Figura 4.8.

- Menu: Propicia ao usuário o retorno imediato ao menu principal do sistema.
- Ajuda: Apresenta-se ao usuário um box com um tutorial de ajuda rápida.
- Diagnose: Apresenta informações sobre a comunicação serial.

Figura 4.8 – Opção "Clássico" com botões Menu, Ajuda e Diagnose pressionados.

Todos os estilos de painéis mostrados anteriormente possuem além dos instrumentos combinados, a imagem do volante e outros acessórios do veículo, pois a função foi de mostrar a visão que o motorista iria ter ao entrar no automóvel. Porém a imagem que será mostrada no painel de instrumentos digital é apenas da parte dos instrumentos combinados, como apresenta a figura 4.9.

Figura 4.9 – Instrumentos combinados da opção "Executivo".

5 - CONCLUSÃO

O objetivo do presente trabalho foi conquistado com êxito.
O avanço da eletrônica embarcada trouxe facilidades e soluções em que painéis com sistemas mecânicos não conseguiam solucionar. Um desses problemas é no desenvolvimento do veículo, onde a premissa é satisfazer o maior número de clientes. A indústria automotiva percebeu que o painel de instrumentos e o design externo do veículo são fatores preponderantes na venda do automóvel e a proposta desse trabalho apresenta uma alternativa de projeto que proporciona forte flexibilidade da customização do painel de instrumentos, onde usuário poderá alterar a interface gráfica pré-definida pelo fabricante.

As sugestões para melhorias futuras são:

- Realizar um estudo de materiais para viabilizar o projeto, pois o conceito de uma customização do painel de instrumentos automotivo é possível de ser realizado;
- Colocar outros parâmetros no painel de instrumentos digitalmente (Ex.: Setas, indicador do nível óleo e combustível, etc);
- Colocar um sistema de antifurto integrando com o painel de instrumentos digitalmente;
- Integrar no painel de instrumentos uma comunicação GPRS para informar a concessionária os parâmetros do veículo;
- Integrar o painel de instrumentos com outras plataformas (Ex. Android, MAC).
- Informar ao usuário um sistema de diagnóstico simplificado em caso de avarias.
- Estudo de outras ferramentas gráficas e hardware.

6 – REFERÊNCIAS BIBLIOGRÁFICAS

[1] BARBOSA, S.B. (2011), **Painel Central de Mostradores para Automóveis: analise históricas e tendências futuras do produto**. Universidade Federal de Santa Catarina.

[2] **E o começo de tudo... História do Automóvel,** http://velocimetrosp.blogspot.com.br, acessado em 22 de Fevereiro de 2013.

[3] SANTOS, Cleiton Gonçalves; FERNANDES, Fernando de França (2012), **Uma Contribuição ao Estudo e Desenvolvimento do Cluster Automotivo Reconfigurável,** Monografia, FATEC Santo André, São Paulo.

[4] HARRIS, William. **Como funcionam os velocímetros,** http://carros.hsw.uol.com.br, acessado em 3 de maio 2013.

[5] **Painel de instrumentos mecânicos,** http://carrosantigos.wordpress.com/2012/01/16 /1939-ford-deluxe-station-wagon-2/, acessado em 12 de março 2013.

[6] **Painel de instrumentos digital,** http://carplace.virgula.uol.com.br, acessado em 12 de março 2013.

[7] HODEL, Kleber N. (2011), **Notas de aula da matéria de Redes de Comunicação** – FATEC Santo André, São Paulo.

[8] **Rede CAN,** www.vector-elearning.com acessado em dezembro de 2012.

[9] KITANI, Edson Caoru (2013), **Notas de aula da matéria de Ferramentas Computacionais** – FATEC Santo André, São Paulo.

[10] BOSCH. (2005), **Manual de Tecnologia Automotiva,** 25ª Edição, São Paulo, Editora Edgard Blucher.

[11] LAGANÁ, A.A.M. (2011), **Apostila de Sensores.** Equipe da FATEC - Santo André.

[12] GUIMARÃES, Alexandre de Almeida. **Eletrônica Embarcada Automotiva.** 1ª ed. São Paulo: Erica, 2007.

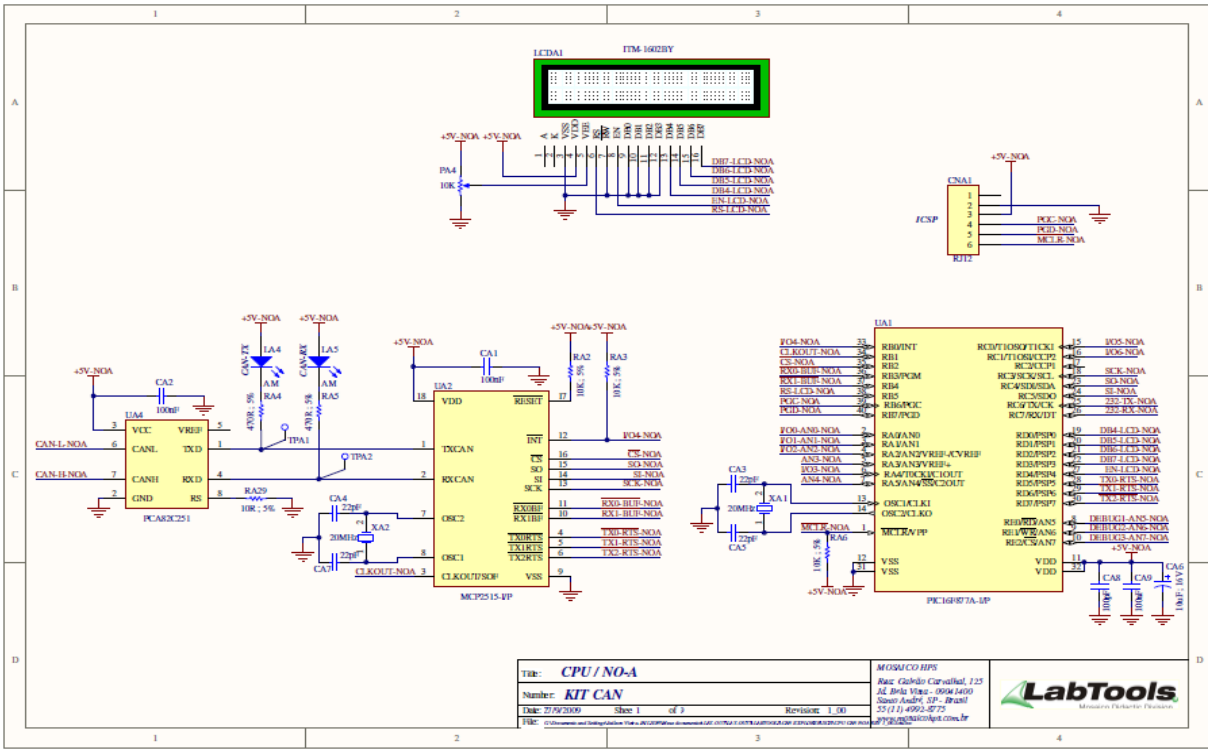
.

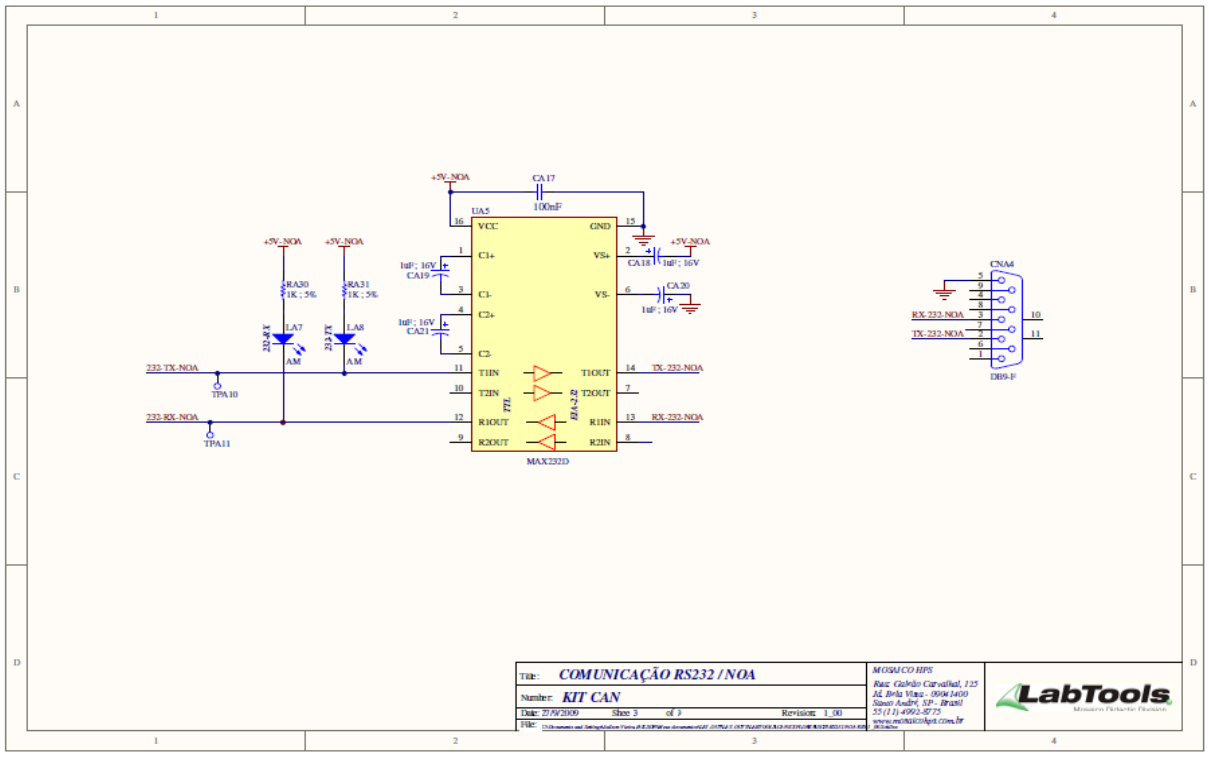
[13] HODEL, Kleber N. (2008), **Limites do Protocolo CAN (Controller Area Network) para Aplicações que Exigem Alto Grau de Confiabilidade Temporal.** Tese de Mestrado, Escola Politécnica da Universidade de São Paulo, São Paulo.

[14] TUBINGEN, Jorg Christoffel (2002), **The History and Future of Driver Information.** Siemens VDO Automotive AG, Regensburg, Scwalbach.

7 – ANEXOS

7.1 Esquema elétrico da placa didática Kit CAN





7.2 Programa Principal.c

```
//*************************************************************************
**//
//   FATEC SANTO ANDRÉ                                          //
//   TCC Customização do Painel de Instrumentos Automotivo      //
//   PROFESSOR ORIENTADOR: Kleber N. Hodel                      //
//   PROFESSOR CO-ORIENTADOR: Fabio Delatore                    //
//   SOFTWARE:  Simulador VW Polo                               //
//   ALUNOS:                                                    //
//            Marcos Felipe de A. Araujo    R.A.: 1013003       //
//            Renato Clementino de Sousa    R.A.: 1012020       //
//*************************************************************************
**//
//              Definições Iniciais do Programa                 //
//*************************************************************************
**//
#INCLUDE "16F877A.h"          //Microcontrolador utilizado PIC 16F877A
#INCLUDE "stdio.h"            //Biblioteca
#INCLUDE "stdlib.h"           //Biblioteca
#FUSES NOWDT                  //No Watch Dog Timer
#FUSES HS                     //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES PUT                    //Power Up Timer
#FUSES NOPROTECT              //Code not protected from reading
#FUSES NODEBUG                //No Debug mode for ICD
#FUSES NOBROWNOUT             //No brownout reset
#FUSES NOLVP                  //No low voltage prgming, B3(PIC16) or B5(PIC18) used for
I/O
#FUSES NOCPD                  //No EE protection
#FUSES NOWRT                  //Program memory not write protected
#use delay(clock=20000000)     //Frequencia do Cristal
```

```
#include <InicializaHardware.h> //Definições do Hardware
#include <Comandos_CAN.h>       //Definições do Controlador CAN MCP2515
#include <matriz.h>             //Matrizes de Rotação e Velocidade
#include <Display_4bits.h>      //Definições do Display LCD
#use rs232(baud=2400,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)//Definições da Serial
//*************************************************************************
**//
//              Definições Iniciais do Programa                          //
//*************************************************************************
**//
#DEFINE    VERSAO_SOFTWARE   30     //Versão do Software em Linguagem C
#DEFINE    VERSAO_HARDWARE   1      //Versão do Hardware
#DEFINE    VERSAO_LABVIEW    14     //Versão do Software Labview
#DEFINE    LED_B1          PIN_E0   //LED B1 do Hardware
#DEFINE    LED_B2          PIN_E1   //LED B2 do Hardware
#DEFINE    LED_B3          PIN_E2   //LED B3 do Hardware
#DEFINE    Botao_IO2       PIN_A2   //Entrada Digital I/O-2  do Hardware
#DEFINE    Botao_IO3       PIN_A4   //Entrada Digital I/O-3  do Hardware
#DEFINE    Botao_IO5       PIN_C0   //Entrada Digital I/O-5  do Hardware
#DEFINE    Botao_IO6       PIN_C1   //Entrada Digital I/O-6  do Hardware
//*************************************************************************
**//
//              Declarações de Varaiáveis                                //
//*************************************************************************
**//
unsigned int Tempo_Display;      //Tempo de Atualização do Display
unsigned int Tempo_Leitura_RPM;  //Tempo de Leitura do Potenciometro da RPM
unsigned int Tempo_Leitura_VEL;  //Tempo de Leitura do Potenciometro da Velocidade
unsigned int Tempo_Leitura_Botao;//Tempo de Leitura das Entradas Digitais
unsigned int Tempo_Leitura_Dado; //Tempo de Leitura do Dado Recebido pelo MCP2515
unsigned int Tempo_Serial;       //Tempo da Porta Serial
```

```
unsigned int rpm_hexa;          //Direciona a Matriz Hexa da RPM
unsigned int rpm;               //Direciona a Matriz Rotação
unsigned int vel_hexa;          //Direciona a Matriz Hexa Velocidade
unsigned int vel;               //Direciona a Matriz Velocidade
unsigned int LCD;               //Direciona em qual case será utilizado
unsigned int Dado_CAN66;        //Guarda valor lido do Registrador 66 do MCP2515
unsigned int Dado_CAN67;        //Guarda valor lido do Registrador 67 do MCP2515
unsigned int Dado_CAN68;        //Guarda valor lido do Registrador 68 do MCP2515
unsigned int Dado_CAN69;        //Guarda valor lido do Registrador 69 do MCP2515
unsigned int Dado_CAN6A;        //Guarda valor lido do Registrador 6A do MCP2515
unsigned int Dado_CAN6B;        //Guarda valor lido do Registrador 6B do MCP2515
unsigned int Dado_CAN6C;        //Guarda valor lido do Registrador 6C do MCP2515
unsigned int Dado_CAN6D;        //Guarda valor lido do Registrador 6D do MCP2515
unsigned int Dado_CAN76;        //Guarda valor lido do Registrador 76 do MCP2515
unsigned int Dado_CAN77;        //Guarda valor lido do Registrador 77 do MCP2515
unsigned int Dado_CAN78;        //Guarda valor lido do Registrador 78 do MCP2515
unsigned int Dado_CAN79;        //Guarda valor lido do Registrador 79 do MCP2515
unsigned int Dado_CAN7A;        //Guarda valor lido do Registrador 7A do MCP2515
unsigned int Dado_CAN7B;        //Guarda valor lido do Registrador 7B do MCP2515
unsigned int Dado_CAN7C;        //Guarda valor lido do Registrador 7C do MCP2515
unsigned int Dado_CAN7D;        //Guarda valor lido do Registrador 7D do MCP2515

unsigned int teste;             //Verifica se está desocupada a serial do MCP2515
unsigned int rpm1;              //Valor recebido da Matriz
unsigned int rpm2;              //Valor recebido da Matriz
unsigned int rpm3;              //Valor recebido da Matriz
unsigned int rpm4;              //Valor recebido da Matriz
unsigned int rpm5;              //Valor recebido da Matriz
unsigned int rpm6;              //Valor recebido da Matriz
unsigned int rpm7;              //Valor recebido da Matriz
unsigned int rpm8;              //Valor recebido da Matriz
```

```
unsigned int rpm9;            //Valor recebido da Matriz
unsigned int rpm10;           //Valor recebido da Matriz
unsigned int rpm11;           //Valor recebido da Matriz
unsigned int rpm12;           //Valor recebido da Matriz
unsigned int rpm13;           //Valor recebido da Matriz
unsigned int rpm14;           //Valor recebido da Matriz
unsigned int rpm15;           //Valor recebido da Matriz
unsigned int rpm16;           //Valor recebido da Matriz
//*************************************************************************
**//
//                  Interrupções - Timer                        //
//*************************************************************************
**//
#int_timer0
void Timer_0(void)                    //Base de tempo 1ms
{
  set_timer0(100+get_timer0());              //Carrega o Timer0 com 632ms
  if (Tempo_Display) Tempo_Display--;         //Decrementa o Tempo Display
  if (Tempo_Leitura_RPM) Tempo_Leitura_RPM--;   //Decrementa o Tempo Leitura RPM
  if (Tempo_Leitura_VEL) Tempo_Leitura_VEL--;   //Decrementa o Tempo Leitura VEL
  if (Tempo_Leitura_Botao) Tempo_Leitura_Botao--;//Decrementa o Tempo Leitura VEL
  if (Tempo_Serial)      Tempo_Serial--;       //Decrementa o Tempo Serial
}
//*************************************************************************
**//
//                  Carregar Variáveis                          //
//*************************************************************************
**//
void Carrega_Variaveis (void)
{
  can_reset ();         //Reseta o Controlador CAN MCP2515
```

```
  configuracao ();       //Configura o Controlador CAN MCP2515
  Tempo_Display=100;     //Carrega a variável Tempo_Display com 500ms
  Tempo_Leitura_RPM=70;  //Carrega a variável Tempo_Leitura_RPM com 70ms
  Tempo_Leitura_VEL=100; //Carrega a variável Tempo_Leitura_VEL com 100ms
  Tempo_Leitura_Botao=50;
}
//*************************************************************************
**//
//                    Versão do Software                    //
//*************************************************************************
**//
void Versao_Display (void)
{
  envia_byte_lcd(0, 0b10000000);          //Escreve na Posição 80 do Display
  printf(escreve_lcd,"TCC  FATEC   SA");  //Escreve no LCD "TCC FATEC SA"
  envia_byte_lcd(0, 0b11000000);          //Escreve na Posição C0 do Display
  printf(escreve_lcd,"Versao:");          //Escreve no LCD "Versao:"
  envia_byte_lcd(0, 0b11001101);          //Escreve na Posição C8 do Display
  printf(escreve_lcd,"%2d", VERSAO_SOFTWARE); //Escreve no LCD o número da versão
do Software
  envia_byte_lcd(0, 0b11001000);          //Escreve na Posição CA do Display
  printf(escreve_lcd,"%1d.", VERSAO_HARDWARE);//Escreve no LCD o número da
versão do Hardware
  envia_byte_lcd(0, 0b11001010);          //Escreve na Posição CC do Display
  printf(escreve_lcd,"%2d.", VERSAO_LABVIEW); //Escreve no LCD o número da versão
do LabView
  delay_ms(5000);                  //Tempo que permanece as informações no LCD
}
//*************************************************************************
**//
//                  Atualiza Mensagens no LCD Hexa              //
```

```
//*************************************************************************
**//
void Atualiza_Mensagem_LCD_Hexa(void)
{
  if(!Tempo_Display)                    //Se o Tempo Display for igual a 0
  {
    Tempo_Leitura_Dado=500;              //Carrega a variável Tempo_Display com 500ms
    vel_hexa = MVelocidade_Hexa[Dado_CAN7D];//Condiciona valor obtido
    envia_byte_lcd(0, 0b11001001);       //Escreve na Posição 80 do Display
    printf(escreve_lcd,"Vel.=");         //Mostra no LCD Vel.=
    envia_byte_lcd(0, 0b11001110);       //Escreve na Posição 80 do Display
    printf(escreve_lcd,"%2X", vel_hexa);   //Mostra no LCD o valor da Velocidade em Hexa
    rpm_hexa = MRotacao_Hexa[Dado_CAN76];  //Condiciona valor obtido
    envia_byte_lcd(0, 0b11000011);       //Escreve na Posição C0 do Display
    printf(escreve_lcd,"=%2X   ", rpm_hexa);//Mostra no LCD o valor da RPM em Hexa
  }
}
//*************************************************************************
**//
//                    Atualiza Mensagens no LCD                    //
//*************************************************************************
**//
void Atualiza_Mensagem_LCD(void)
{
  if(!Tempo_Display)                    //Se o Tempo Display for igual a 0
  {
    Tempo_Display=100;                  //Carrega Tempo Display com 100ms
    vel = MVelocidade[Dado_CAN7D];       //Condiciona valor obtido
    envia_byte_lcd(0, 0b10000000);       //Escreve na Posição 80 do Display
    printf(escreve_lcd,"Vel. = ");       //Mostra no LCD Vel.=
    envia_byte_lcd(0, 0b10000111);       //Escreve na Posição 87 do Display
```

```
    printf(escreve_lcd,"%3u ", vel);       //Mostra no LCD o valor da Velocidade
    envia_byte_lcd(0, 0b11000011);         //Escreve na Posição 80 do Display
    rpm = MRotacao[Dado_CAN76];            //Condiciona valor obtido
    printf(escreve_lcd,"  = %2u", rpm);    //Mostra no LCD o valor da RPM
    envia_byte_lcd(0, 0b10001010);         //Escreve na Posição 8A do Display
    printf(escreve_lcd," Km/h   ");        //Mostra no LCD Km/h
    envia_byte_lcd(0, 0b11001001);         //Escreve na Posição C9 do Display
    printf(escreve_lcd,"00     ");         //Mostra no LCD 00
    rpm_hexa = MRotacao_Hexa[Dado_CAN76];  //Condiciona valor obtido
    vel_hexa = MVelocidade_Hexa[Dado_CAN7D];//Condiciona valor obtido

  }
}
//*************************************************************************
**//
//                    Atualiza Mensagens no LCD registradores          //
//*************************************************************************
**//
void Atualiza_Mensagem_LCD_Registradores6(void)
{
  if(!Tempo_Display)                   //Se o Tempo Display for igual a 0
  {
    Tempo_Display=50;
    envia_byte_lcd(0, 0b10000000);       //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/", rpm1);    //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b10000011);       //Escreve na Posição 83 do Display
    printf(escreve_lcd,"%2X/", rpm2);    //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b10000110);       //Escreve na Posição 86 do Display
    printf(escreve_lcd,"%2X/", rpm3);    //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b10001001);       //Escreve na Posição 89 do Display
    printf(escreve_lcd,"%2X/Reg.", rpm4);//Mostra no LCD valor recebido no registrador
```

```
    envia_byte_lcd(0, 0b11000000);      //Escreve na Posição C0 do Display
    printf(escreve_lcd,"%2X/", rpm5);   //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b11000011);      //Escreve na Posição C3 do Display
    printf(escreve_lcd,"%2X/", rpm6);   //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b11000110);      //Escreve na Posição C6 do Display
    printf(escreve_lcd,"%2X/", rpm7);   //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b11001001);      //Escreve na Posição C9 do Display
    printf(escreve_lcd,"%2X    6", rpm8);//Mostra no LCD valor recebido no registrador
  }
}
//*************************************************************************
**//
//                    Atualiza Mensagens no LCD registradores         //
//*************************************************************************
**//
void Atualiza_Mensagem_LCD_Registradores7(void)
{
  if(!Tempo_Display)              //Se o Tempo Display for igual a 0
  {
    Tempo_Display=50;
    envia_byte_lcd(0, 0b10000000); //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/", rpm9);
    envia_byte_lcd(0, 0b10000011); //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/", rpm10);
    envia_byte_lcd(0, 0b10000110); //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/", rpm11);  //Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b10001001);      //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/Reg.", rpm12);//Mostra no LCD valor recebido no registrador
    envia_byte_lcd(0, 0b11000000); //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/", rpm13);
    envia_byte_lcd(0, 0b11000011); //Escreve na Posição 84 do Display
```

```
    printf(escreve_lcd,"%2X/", rpm14);
    envia_byte_lcd(0, 0b11000110); //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X/", rpm15);
    envia_byte_lcd(0, 0b11001001); //Escreve na Posição 84 do Display
    printf(escreve_lcd,"%2X    7", rpm16);
  }
}
//*************************************************************************
**//
//                    Atualiza Mensagens CAN Hexadeciamal          //
//*************************************************************************
**//
void Atualiza_Mensagem_CAN_Hexa(void)
{
  rpm1 = MRotacao_Hexa[Dado_CAN66];    //Condiciona valor obtido
  rpm2 = MRotacao_Hexa[Dado_CAN67];    //Condiciona valor obtido
  rpm3 = MRotacao_Hexa[Dado_CAN68];    //Condiciona valor obtido
  rpm4 = MRotacao_Hexa[Dado_CAN69];    //Condiciona valor obtido
  rpm5 = MRotacao_Hexa[Dado_CAN6A];    //Condiciona valor obtido
  rpm6 = MRotacao_Hexa[Dado_CAN6B];    //Condiciona valor obtido
  rpm7 = MRotacao_Hexa[Dado_CAN6C];    //Condiciona valor obtido
  rpm8 = MRotacao_Hexa[Dado_CAN6D];    //Condiciona valor obtido
  rpm9 = MRotacao_Hexa[Dado_CAN76]; // Condiciona valor obtido
  rpm10 = MRotacao_Hexa[Dado_CAN77]; // Condiciona valor obtido
  rpm11 = MRotacao_Hexa[Dado_CAN78]; // Condiciona valor obtido
  rpm12 = MRotacao_Hexa[Dado_CAN79];  //Condiciona valor obtido
  rpm13 = MRotacao_Hexa[Dado_CAN7A]; // Condiciona valor obtido
  rpm14 = MRotacao_Hexa[Dado_CAN7B]; // Condiciona valor obtido
  rpm15 = MRotacao_Hexa[Dado_CAN7C]; // Condiciona valor obtido
  rpm16 = MRotacao_Hexa[Dado_CAN7D]; // Condiciona valor obtido
}
```

```
//*************************************************************************
**//
//                    Entrada Digital Botão                    //
//*************************************************************************
**//
void Entrada_Digital(void)
{
  if (!Tempo_Leitura_Botao)
  {
     if (input(Botao_IO5))
     {
       Tempo_Leitura_Botao=50;
       LCD=1;
     }
     if (!input(Botao_IO5))
     {
      Tempo_Leitura_Botao=50;
      LCD=2;
     }
  }
}
//*************************************************************************
**//
//                    Leitura Dado CAN                    //
//*************************************************************************
**//
void Leitura_Dado_CAN(void)
{
  read(0x2C);              //Só envia mensagem caso linha esteja desocupada
  teste = spi_read(0);      //Ler a Serial do MCP2515 e move para teste
  output_high (CS);         //Comando para o MCP2515
```

```
  delay_us (10);            //Tempo de Estabilização do MCP2515
  if (teste)//Verifica se tem dado na serial e se o Tempo Leitura Dado igual a 0
  {
    Tempo_Leitura_Dado=500;    //Carrega o Tempo Leitura Dado com 100ms
    write (0x0C, 0x00);        //Comando para o MCP2515
    read(0x66);                                                    //Registradores de leitura do
reciver 66h
    Dado_CAN66 = spi_read(0);    //Valor lido no Registrador 66h é passado para
Dado_CAN66h
    output_high (CS);          //Comando para o MCP2515
    delay_us (10);             //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);        //Comando para o MCP2515
    read(0x67);                //Registradores de leitura do reciver 6Dh
    Dado_CAN67 = spi_read(0);    //Valor lido no Registrador 6Dh é passado para
Dado_CAN6Dh
    output_high (CS);          //Comando para o MCP2515
    delay_us (10);             //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);        //Comando para o MCP2515
    read(0x68);                //Registradores de leitura do reciver 76h
    Dado_CAN68= spi_read(0);       //Valor lido no Registrador 76h é passado para
Dado_CAN76h
    output_high (CS);          //Comando para o MCP2515
    delay_us (10);             //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);        //Comando para o MCP2515
    read(0x69);                //Registradores de leitura do reciver 7Dh
    Dado_CAN69= spi_read(0);       //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);          //Comando para o MCP2515
    delay_us (10);             //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);        //Comando para o MCP2515
    read(0x6A);                //Registradores de leitura do reciver 7Dh
```

```
    Dado_CAN6A= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);
    write (0x0C, 0x00);       //Comando para o MCP2515
    read(0x6B);               //Registradores de leitura do reciver 7Dh
    Dado_CAN6B= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);
    write (0x0C, 0x00);       //Comando para o MCP2515
    read(0x6C);               //Registradores de leitura do reciver 7Dh
    Dado_CAN6C= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);
    write (0x0C, 0x00);       //Comando para o MCP2515
    read(0x6D);               //Registradores de leitura do reciver 7Dh
    Dado_CAN6D= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);
//            Leitura Dado CAN Registradores 76h - 7Dh                //
    write (0x0C, 0x00);       //Comando para o MCP2515
    read(0x76);               //Registradores de leitura do reciver 66h
    Dado_CAN76 = spi_read(0);   //Valor lido no Registrador 66h é passado para
Dado_CAN66h
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);            //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);       //Comando para o MCP2515
    read(0x77);               //Registradores de leitura do reciver 6Dh
```

```
    Dado_CAN77 = spi_read(0);  //Valor lido no Registrador 6Dh é passado para
Dado_CAN6Dh
    output_high (CS);        //Comando para o MCP2515
    delay_us (10);           //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);      //Comando para o MCP2515
    read(0x78);              //Registradores de leitura do reciver 76h
    Dado_CAN78= spi_read(0);      //Valor lido no Registrador 76h é passado para
Dado_CAN76h
    output_high (CS);        //Comando para o MCP2515
    delay_us (10);           //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);      //Comando para o MCP2515
    read(0x79);              //Registradores de leitura do reciver 7Dh
    Dado_CAN79= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);        //Comando para o MCP2515
    delay_us (10);           //Tempo de Estabilização do MCP2515
    write (0x0C, 0x00);      //Comando para o MCP2515
    read(0x7A);              //Registradores de leitura do reciver 7Dh
    Dado_CAN7A= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);        //Comando para o MCP2515
    delay_us (10);
    write (0x0C, 0x00);      //Comando para o MCP2515
    read(0x7B);              //Registradores de leitura do reciver 7Dh
    Dado_CAN7B= spi_read(0);      //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);        //Comando para o MCP2515
    delay_us (10);
    write (0x0C, 0x00);      //Comando para o MCP2515
    read(0x7C);              //Registradores de leitura do reciver 7Dh
```

```
    Dado_CAN7C= spi_read(0);       //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);
    write (0x0C, 0x00);       //Comando para o MCP2515
    read(0x7D);               //Registradores de leitura do reciver 7Dh
    Dado_CAN7D= spi_read(0);       //Valor lido no Registrador 7Dh é passado para
Dado_CAN7Dh
    output_high (CS);         //Comando para o MCP2515
    delay_us (10);


  }
}
//*************************************************************************
**//
//                    Enviar Serial Hexa                          //
//*************************************************************************
**//
void Envia_Serial_Hexa(void)
{
  if (!Tempo_Serial)     //Se o Tempo Serial for igual a 0
  {
    Tempo_Serial=500;//Carrega a variável Tempo_Serial com 200ms
    //putc (rpm3);     //Manda para Serial o valor da rpm hexa
    //putc (rpm4);     //Manda para Serial o valor da rpm hexa
    putc (rpm15);    //Manda para Serial o valor da velocidade hexa
    write(0x2C, 0x00);  //Comando para o MCP2515


  }
}
```

```
//*************************************************************************
**//
//                    Limpar Serial                          //
//*************************************************************************
**//
void Limpar_Serial(void)
{
  putc (0b00000000);    //Manda para Serial o valor da rpm hexa
  putc (0b00000000);    //Manda para Serial o valor da rpm hexa
  putc (0b00000000);   //Manda para Serial o valor da velocidade hexa
}
//*************************************************************************
**//
//                    Função Principal                       //
//*************************************************************************
**//
void main(void)
{
  InicializaMicrocontrolador(); //Inicializa as Definições do Microcontrolador
  Carrega_Variaveis();        //Carrega as variáveis com valores pré-definidos
  Inicializa_LCD();           //Inicializa as Definições do LCD
  Versao_Display();           //Envia para o Display as Versões dos Softwares
  Limpa_LCD();                //Limpa o LCD
  Mascara_Display_Hexa();     //Escreve Velocidade: e Rotacao no LCD
  Limpar_Serial();            //Limpa Serial
  while(true)                 //Laço While (Enquanto for verdadeiro)
  {
    Entrada_Digital();
    Leitura_Dado_CAN();
   Atualiza_Mensagem_CAN_Hexa();
    switch (LCD)
```

```
    {
      case 1:  Atualiza_Mensagem_LCD_Registradores6();
               Envia_Serial_Hexa();
               break;
      case 2:  Atualiza_Mensagem_LCD_Registradores7();
               Envia_Serial_Hexa();
               break;
    }
  }
}
```

7.3 InicializaHardware.h

```
//*************************************************************************
**//
//                  Inicializa Microcontrolador                          //
//*************************************************************************
**//
void InicializaMicrocontrolador(void)
{
  setup_adc_ports(AN0_AN1_AN3);
  setup_adc(ADC_CLOCK_DIV_8);
  set_tris_a(0b11111111);
  set_tris_b(0b11011011);
  set_tris_c(0b10110111);
  set_tris_d(0b11111111);
  set_tris_e(0b00000000);
  output_high(pin_e0);    //Apagar o LED
  output_high(pin_e1);    //Apagar o LED
  output_high(pin_e2);    //Apagar o LED
```

```
  setup_spi(SPI_MASTER | SPI_L_TO_H | SPI_XMIT_L_TO_H | SPI_CLK_DIV_16); // Configura a comunicação SPI como Master, com uma atuação na borda de subida e com uma divisão de 16 no clock
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32);//20MHz/4=5MHz;  Pré-Escaler=32; 5MHz/32=156250Hz; 1/156250=6,4us; 6,4us*255=1,632ms, Base de tempo 1 ms é necessário carregar com 100.
  enable_interrupts(global|int_timer0); // Habilita interrupções
}
```

7.4 ComandosCAN.h

```
//*************************************************************************
**//
//   FATEC SANTO ANDRÉ                                          //
//   TCC Customização do Painel de Instrumentos Automotivo      //
//   PROFESSOR: Kleber N. Hodel                                 //
//   SOFTWARE:  Comandos CAN                                    //
//   ALUNOS:                                                    //
//           Marcos Felipe de A. Araujo    R.A.: 1013003        //
//           Renato Clementino          R.A.: 1012020           //
//
//*************************************************************************
**//
//              Definições Iniciais do Programa                //
//*************************************************************************
**//

#DEFINE    CS            PIN_B2

//*************************************************************************
**//
```

```
//                Configuração da Escrita no Transiver                  //
//*************************************************************************
**//
void write (int end, int dado)
{
  output_high (CS);  // Configuração do Chip Select- Saída CS=RB2
  output_low(CS);    // Configuração do Chip Select- Saída CS=RB2
  output_high(CS);   // Configuração do Chip Select- Saída CS=RB2
  delay_us (10);
  output_low(CS);       // Inicia a comunicação(SPI) entre o PIC e o Controlador CAN-
Configuração do Chip Select- Saída CS=RB2(Nível Baixo)
  spi_write(0b00000010); // Manda a instrução de escrita para o transiver(Controlador CAN)
  spi_write(end);        // Envia o endereço no qual se deseja guardar o dado.
  spi_write(dado);       // Envia o dado
  output_high(CS);     // Termina a comunicação(SPI) entre o PIC e o Controlador CAN-
Configuração do Chip Select-Saída CS=RB2(Nível ALto)
  delay_us (10);
}
//*************************************************************************
**//
//                Configuração da Leitura no Transiver                  //
//*************************************************************************
**//
void read (int end)
{
  output_high (pin_b2); // Configuração do Chip Select
  output_low(PIN_b2); // Configuração do Chip Select
  output_high (pin_b2); // Configuração do Chip Select
  delay_us (10);
  output_low(PIN_b2); // Configuração do Chip Select
  spi_write(0b00000011); // Manda a instrução de leitura para o transiver
```

```
  spi_write(end); // Envia o endereço no qual deseja receber a informação- Reg. RXF1SIDH-
00000100
}
//*************************************************************************
**//
//                Configuração do Modo do Controlador CAN                //
//*************************************************************************
**//
void configuracao ()
{
//////////////*Configuração do Modo de Operação*//////////////////

  write (0x0F, 0b10000000); // CANCTRL, coloca em Modo de Configuração pg58
  delay_ms (100);

//////////////*Configuração dos Modos de Sincronismo*/////////////

  write (0x28, 0b01000011); //CNF3, Filtro desabilitado, Clock Out Habilitado, Fase2 5Tq.
  write (0x29, 0b10011000); //CNF2, Segmento de Propagação 1Tq, Fase1 5Tq.
  write (0x2A, 0b00000001); //CNF1, SJW 1Tq,  Baud Rate 500Kbps, Osc 20Mhz -Total Tq
= 10

//////////////*Configuração do RXB0 e RXB1*////////////////////

  write (0x60, 0b00000000); //Controle do RXB0
  write (0x70, 0b00000000); //Controle do RXB1

//////////////*Configuração do Filtro Rotação*////////////////////

  write (0x00, 0b01010000); //Mais Significativo Rotação
  write (0x01, 0b00000000); //Menos Significativo Rotação
```

```
//////////////*Configuração da Máscara*//////////////////

   write (0x20, 0b11111111); //Mais Significativo Rotação
   write (0x21, 0b11100000); //Menos Significativo Rotação

//////////////*Configuração do Filtro Velocidade*//////////////

   write (0x18, 0b01100100); //Mais Significativo Rotação
   write (0x19, 0b00000000); //Menos Significativo Rotação

//////////////*Configuração da Mascara da Velocidade*//////////////////

   write (0x24, 0b01100100); //Mais Significativo Rotação
   write (0x25, 0b11100000); //Menos Significativo Rotação

//////////////*Configuração das Interrupções*//////////////////

   write (0x2B, 0b00010000); //  Interrupções Desabilitadas

//////////////*Configuração Modo de Funcionamento Normal*////////

   write (0x2C, 0b00000000); // Zera as interrupções e libera a mensagem para ser enviada
   write (0x0F, 0b00000000); //CANCTRL, coloca em modo de trabalho
}
//*************************************************************************
**//
//                Configuração para Reset do Controlador CAN              //
//*************************************************************************
**//
void can_reset()
```

```
{
  output_high (CS); // Configuração do Chip Select
  output_low(CS); // Configuração do Chip Select
  output_high(CS); // Configuração do Chip Select
  delay_us (10);
  output_low(CS); // Configuração do Chip Select
  spi_write(0b11000000);
  output_high(CS); // Configuração do Chip Select
  delay_us (10);
}
```

7.5 Display4bits.h

```
//*************************************************************************
**//
//              Definições Iniciais do Programa                          //
//*************************************************************************
**//
#DEFINE    DISPLAY_4       PIN_D0
#DEFINE    DISPLAY_5       PIN_D1
#DEFINE    DISPLAY_6       PIN_D2
#DEFINE    DISPLAY_7       PIN_D3
#DEFINE    LCD_E           PIN_D4
#DEFINE    LCD_R           PIN_B5
//*************************************************************************
**//
//                        Definições LCD                                 //
//*************************************************************************
**//

//*************************************************************************
**//
```

```
//                Envia de Nibble para o LCD                        //
//*************************************************************************
**//
void envia_nibble_lcd(int dado)
{
  //Carrega as vias de dados (pinos) do LCD de acordo com o nibble lido
  output_bit(DISPLAY_4, bit_test(dado,0)); //Carrega DB4 do LCD com o bit DADO<0>
  output_bit(DISPLAY_5, bit_test(dado,1)); //Carrega DB5 do LCD com o bit DADO<1>
  output_bit(DISPLAY_6, bit_test(dado,2)); //Carrega DB6 do LCD com o bit DADO<2>
  output_bit(DISPLAY_7, bit_test(dado,3)); //Carrega DB7 do LCD com o bit DADO<3>
  //Gera um pulso de enable
  output_high(LCD_E); // ENABLE = 1
  delay_us (1); // Recomendado para estabilizar o LCD
  output_low(LCD_E); // ENABLE = 0
}

//*************************************************************************
**//
//                Envio de Byte para o LCD                          //
//*************************************************************************
**//
  //Esta rotina irá enviar um dado ou um comando para o LCD conforme abaixo:
  // ENDEREÇO = 0 -> a variável DADO será uma instrução
  // ENDEREÇO = 1 -> a variável DADO será um caractere
void envia_byte_lcd(boolean endereco, int dado)
{
  output_bit(LCD_R,endereco);// Seta o bit RS para instrução ou caractere
  delay_us(100); // Aguarda 100 us para estabilizar o pino do LCD
  output_low(LCD_E); // Desativa a linha ENABLE
  envia_nibble_lcd(dado>>4); // Envia a parte ALTA do dado/coamndo
```

```
  envia_nibble_lcd(dado & 0B00001111);// Limpa a parte ALTA e envia a parte BAIXA do dado/comando
  delay_us(40); // Aguarda 40us para estabilizar o LCD
}

//*****************************************************************************
**//
//                  Envio de Caractere para o LCD                          //
//*****************************************************************************
**//
  // Esta rotina serve apenas como uma forma mais fácil de escrever um caractere no display.
  //Ela pode ser eliminada e ao invés dela usaremos diretamente a função envia_byte_lcd(1,"<caractere a ser mostrado no lcd>");
  //ou envia_byte_lcd(1,<código do caractere a ser mostrado no lcd>);
void Escreve_LCD(char c)
{
  envia_byte_lcd(1,c);       // Envia caractere para o display
}

//*****************************************************************************
**//
//                    Função para limpar o LCD                             //
//*****************************************************************************
**//

void Limpa_LCD()
{
  envia_byte_lcd(0,0B0000001); // Envia instrução para limpar o LCD
  delay_ms(2); // Aguarda 2ms para estabilizar o LCD
}
```

```
//*************************************************************************
**//
//                    Inicializa o LCD                          //
//*************************************************************************
**//
void Inicializa_LCD()
{
  output_low(DISPLAY_4); // Garante que o pino DB4 estão em 0 (low)
  output_low(DISPLAY_5); // Garante que o pino DB5 estão em 0 (low)
  output_low(DISPLAY_6); // Garante que o pino DB6 estão em 0 (low)
  output_low(DISPLAY_7); // Garante que o pino DB7 estão em 0 (low)
  output_low(LCD_R); // Garante que o pino RS estão em 0 (low)
  output_low(LCD_E); // Garante que o pino ENABLE estão em 0 (low)
  delay_ms(15); // Aguarda 15ms para estabilizar o LCD
  envia_nibble_lcd(0b00000011); // Envia comando para inicializar o display
  delay_ms(5); // Aguarda 5ms para estabilizar o LCD
  envia_nibble_lcd(0b00000011); // Envia comando para inicializar o display
  delay_ms(5); // Aguarda 5ms para estabilizar o LCD
  envia_nibble_lcd(0b00000011); // Envia comando para inicializar o display
  delay_ms(5); // Aguarda 5ms para estabilizar o LCD
  envia_nibble_lcd(0b00000010);// CURSOR HOME - Envia comando para zerar o contador
de caracteres e retornar à posição inicial (0x80).
  delay_ms(1); // Aguarda 1ms para estabilizar o LCD
  envia_byte_lcd(0,0b00101000); // FUNCTION SET - Configura o LCD para 4 bits,2 linhas,
fonte 5X7.
  envia_byte_lcd(0,0b00001100); // DISPLAY CONTROL - Display ligado, sem cursor
  Limpa_LCD(); // Limpa o LCD
  envia_byte_lcd(0,0b00000110); // ENTRY MODE SET - Desloca o cursor para a direita
}
```

```
//*************************************************************************
**//
//                    Máscara Display                    //
//*************************************************************************
**//
void Mascara_Display_Hexa(void)
{
  envia_byte_lcd(0, 0b10000000); //Escreve na Posição 80 do Display
  printf(escreve_lcd,"Vel. =");
  envia_byte_lcd(0, 0b11000000); //Escreve na Posição 80 do Display
  printf(escreve_lcd,"RPM  =");
}
```