Felipe Augusto Vieira Fagundes

Gustavo Luiz da Silva

Marco Aurélio Scomparim Assis

Sistema de monitoramento automotivo remoto

Santo André – São Paulo

2015

Felipe Augusto Vieira Fagundes

Gustavo Luiz da Silva

Marco Aurélio Scomparim Assis

Sistema de monitoramento automotivo remoto

Monografia apresentada ao Curso de Tecnologia Eletrônica Automotiva da FATEC Santo André, como Requisito parcial para conclusão do curso em Tecnologia em Eletrônica Automotiva.

Orientador: Prof. Weslley Torres

Santo André – São Paulo

2015

LISTA DE PRESENÇA

Fatec CENTRO PAULA SOUZA GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Santo André

LISTA DE PRESENÇA

SANTO ANDRÉ, 03 DE JULHO DE 2015.

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

BANCA

PRESIDENTE:
PROFº. MSC. WESLLEY MEDEIROS TORRES

MEMBROS:
PROFª. MSC. SUELY MIDORI AOKI

PROFº. MSC. CLEBER WILLIAN GOMES

ALUNOS:

MARCO AURELIO SCOMPARIM ASSIS

FELIPE AUGUSTO VIEIRA FAGUNDES

GUSTAVO LUIZ DA SILVA

www.fatecsantoandre.com.br fatecstdo@gmail.com
Rua Prefeito Justino Paixão, 150 – Centro – Santo André – SP - CEP: 09020-130
Fone (0xx11) 4437-2215

"Penso, logo existo"

Descartes, René

AGRADECIMENTOS

Eu agradeço primeiramente a Deus por me conceder sabedoria e paciência nesta etapa da minha via, agradeço a minha esposa, sogra e filha por me compreender e me ajudar em todos os momentos e agradeço a todos meus amigos e colegas que me auxiliaram nos momentos mais difíceis.

Marco A. S. Assis.

Agradeço principalmente a minha família e amigos que sempre ajudaram e incentivaram e a todos envolvidos direta e indiretamente no desenvolvimento desse projeto.

Gustavo L. da Silva

Meus sinceros agradecimentos a todos aqueles que de alguma forma doaram um pouco de si para que a conclusão deste trabalho se tornasse possível

Felipe A. V. Fagundes

RESUMO

Este projeto visa a simulação dos dados de um veículo (velocidade, rotação, temperatura e TPS) a partir da leitura do sistema OBD (*On-Board Diagnosis*), tratamento dos dados através de um microcontrolador para envio a partir de um sistema de telemetria via rede de dados móvel (GSM) e leitura a partir de um browser de internet. Com esse sistema o usuário será capaz de monitorar o veículo em tempo real verificando o comportamento do veículo naquele instante.

Key words: ODB, Telemetria, EVK2, GSM;

ABSTRACT

This project aims to the simulation data of a vehicle (speed, RPM, temperature, load and TPS) from the reading of the OBD (On-Board Diagnosis) of data processing through a microcontroller for sending from system telemetry via mobile data network (GSM) and reading from an internet browser. With this system the user will be able to monitor the real-time vehicle checking vehicle behavior at that moment.

Keywords: OBD, Telemetry, EVK2, GSM;

SUMÁRIO

INDICE DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIAÇOES

TPS - *Throttle position sensor*

GPS – *Global position system*

OBD – *On Board Diagnosis System*

RPM – *Revolutions per minute*

GSM - *Group Special Mobile*

ECU – *Electronic control unit*

M2M – *Machine to machine*

SAE - *Society of Automotive Engineers*

ISO - *International Organization for Standardization*

GND – *Ground*

CI – Circuito integrado

UMTS - *Universal Mobile Telecommunication System*

HSDPA - *High-Speed Downlink Packet Access*

CDMA - *Code Division Multiple Access*

USB – *Universal serial Bus*

EEC-V - *Electronic Engine Control V*

VPW - *Variable pulse width*

PWM - *Pulse width modulation*

CRC - *Cyclical Redundancy Check*

LGA - *Land grid array*

GPRS - *General Packet Radio Services*

LTE - *Long Term Evolution*

HSPA - *High Speed Packet Access*

EVDO - *Evolution-Data Optimized*

HSPA - *High Speed Packet Access*

UMTS - *Universal Mobile Telecommunication System*

GPIO - *General Purpose Input/Output*

UART - *Universal asynchronous receiver/transmitter*

ADC - *Application delivery controller*

DAC - *Digital-to-Analog Converter*

SIM - *subscriber identity module*

I/O – *In/Out*

SPI - *Serial Peripheral Interface*

I2C - *Inter-Integrated Circuit*

UDP - *User Datagram Protocol*

FTP - *File Transfer Protocol*

SMTP - *Simple Mail Transfer Protocol*

TCP - *Transmission Control Protocol*

TTFF - *Time To First Fix*

EEPROM – *Electrically Erasable Programmable Read Only Memory*

NMEA - *National Marine Electronics Association*

dBm - *decibel miliwatt*

EEC - *Electronic Engine Control*

PCM - *Powertrain Control Module*

DTC – *Diagnostic Trouble Codes*

VSS – *Vehicle Speed Sensor*

NTC - *Negative Temperature Coefficient*

MAF – *Mass Air Flow*

1. Introdução

Atualmente a maior parte de veículos na frota nacional possuem módulos de controle que conseguem prover informações sobre o seu funcionamento através de um canal de comunicação destinado a um serviço de diagnóstico, canal este que é utilizado principalmente com o objetivo de prover informações e ajustes destinados à manutenção do veículo. Nosso projeto consiste na utilização das informações providas por este canal para aplicação em um sistema de telemetria, através de uma interface que efetua a leitura dos dados de funcionamento do veículo via OBD II, parametriza e faz o envio dessas informações por GSM, para que as mesmas possam vir a ser utilizadas em diversas aplicações que necessitam destas.

1.1. Justificava

A maioria dos sistemas de telemetria faz a leitura dos dados de forma invasiva, tendo a necessidade de fazer corte em chicotes e atuadores para conseguir efetuar as leituras dos sinais dos mesmos e assim invalidando garantias de fábrica e comprometendo a confiabilidade do sistema elétrico e eletrônico do veículo, além do sistema ter a necessidade de ser desenvolvido de acordo com a arquitetura de cada veículo, já que lendo diretamente nos sensores a leitura terá de ser feita com base nos valores do mesmo por não existir um padrão para as leituras devido a cada sensor ter sua característica especifica, assim, por sua vez encarecendo muito ter que desenvolver uma interface diferente para cada sistema de injeção de um carro. Por desconhecermos a existência de um sistema de telemetria que faça a leitura dos dados via tomada de diagnostico, decidimos desenvolver uma interface que efetuasse a leitura dos sensores de uma forma menos invasiva e que pudesse ser feita na maioria dos veículos sem a necessidade de alterar o hardware e software e por isso a leitura foi feita via padrão OBDII.

1.2. Objetivos

Desenvolver uma plataforma que faça a leitura dos dados de funcionamento de um veículo, interprete, condicione esses dados e transmita os mesmos sem a utilização de fios, para que estes dados possam ser utilizados por qualquer tipo de plataforma destinada a utilizar essas informações para possíveis recursos disponíveis com as mesmas.

2. Referencial teórico

2.1. Introdução à telemetria

Telemetria é a transmissão e recepção sem fio de grandezas e medidas com a finalidade de monitorar remotamente condições ambientais ou parâmetros do equipamento para como exemplo melhor sua aplicação.

Existem diversas formas de transmissão e recepção de dados sem a utilização de fios de um local para outro utilizando ondas de rádio (Ondas eletromagnéticas que se formam devido à oscilação simultânea de um campo elétrico e de um campo magnético perpendicular entre si) e cada uma terá suas características especificas, seja a distância de transmissão, a velocidade em que os dados trafegam, a segurança dos dados trafegados, a confiabilidade do sistema de transmissão, entre outras, sendo necessário observar qual será a aplicação e necessidade destinada para que se possa selecionar a correta.

As principais tecnologias utilizadas atualmente são:

- Wireless

Também conhecida como *Wi-fi*, permite a criação de ondas de rádio em frequências não licenciadas, dispensando qualquer tipo de licença ou autorização do agente regulador das comunicações para operar (brasilescola.com).

- Bluetooth

É uma especificação industrial para a comunicação em curta distância de redes sem fio (10 metros) com um baixo custo e alta operabilidade para dados ou voz com largura da banda de 1MBit/s e velocidade máxima de 783KBit/s. A tecnologia Bluetooth opera na faixa de frequência de 2.4 GHz com conexão ponto a ponto ou ponto a rede, a segurança na troca de informações é garantida trocando a frequência 1600 vezes por segundo entre 79 canais (2402 MHz a 2480 MHz), o algoritmo de troca é randômico e baseado no endereço do máster, todos os 79 canais (1 MHz cada) são usados antes de um canal ser reutilizado.

- GSM

O sistema GSM é um sistema celular digital de segunda geração, concebido com o propósito de resolver os problemas de fragmentação dos primeiros sistemas celulares na Europa. O GSM é o primeiro sistema celular no mundo a especificar modulação digital e arqui-

teturas de serviços de nível de rede. Antes do GSM, os países da Europa utilizavam padrões diferentes dentro do continente e não era possível a um usuário utilizar um único terminal em toda a Europa (Redes GSM e GPRS - Prof. Dr. Omar Branquinho).

No local remoto onde esses dados serão obtidos existem vários sensores que serão neste caso a fonte de dados da nossa aplicação. A saída do sensor é convertida em dados digitais por algum dispositivo ou computador, esse dispositivo é ligado a um modem que converte esses dados digitais para um sinal analógico que pode ser transmitido através do ar. O sinal é enviado para um receptor e esse processo é executado inversamente. O modem converte o sinal analógico recebido para um sinal digital para que possa ser processado ou visualizado por algum equipamento (Figura 1).

Figura 1– Topologia de um sistema de telemetria

Os sistemas de telemetria podem apresentar as seguintes vantagens e desvantagens

- Vantagens:

Flexibilidade: comunicação sem restrições e alcance onde redes com fios estariam impossibilitadas de chegar.

Robustez: não depende de meio físico para transporte das mensagens

Redução do Custo e facilidade: Como uma rede GSM não necessita de fios sua construção é muito mais barata

- Desvantagens:

Qualidade de serviço: por depender da qualidade do sinal emitido uma rede sem fio pode se tornar muito lenta.

Segurança: redes sem fio são mais suscetíveis a ataques externos não desejados. O uso de ondas para comunicação pode ocasionar interferência nos demais equipamentos

Baixa transferência de dados.

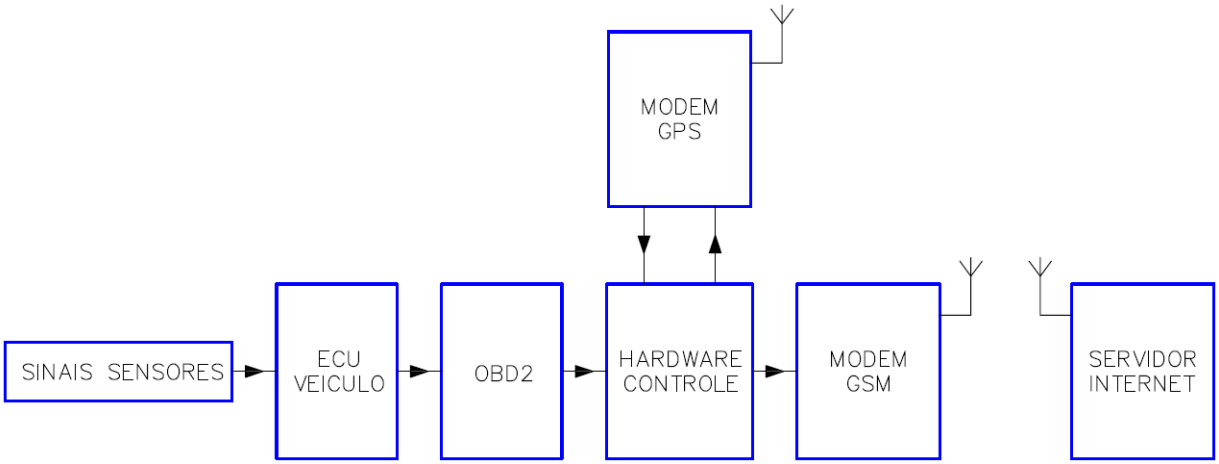


Figura 2 - Diagrama em Blocos do Projeto

A aplicação do projeto consiste na simulação dos dados para ECU, esses dados serão recebidos pelo scanner ODBII e o hardware de controle tem a função de receber, tratar e enviar esses dados via conexão GSM juntamente com os dados de localização recebidos pelo modem GPS. Esses dados poderão ser visto a partir de um servidor que pode ser um computador pessoal ou um celular conforme diagrama em blocos na figura acima.

2.2. Interface de comunicação veicular (OBD II)

Todos os veículos produzidos no Brasil a partir de 2010 devem ter as especificações OBDII instaladas para controle de emissões. Este padrão controla as atividades do veículo a fim de minimizar as emissões ocasionadas por falhas no motor e periféricos. A maior diferença entre o sistema construído para OBDI e para OBDII, é que o sistema OBDI monitora falha nos sensores, enquanto que o sistema OBDII monitora o desempenho de funcionamento dos mesmos, além de outros detalhes como o aumento do número de códigos de falha e a padronização na localização da tomada de diagnostico. Esta nova lógica de trabalho permite que o motor mantenha um nível de emissões o mais limpa possível, dentro de todo ciclo de vida útil do motor e componentes do sistema de emissões.

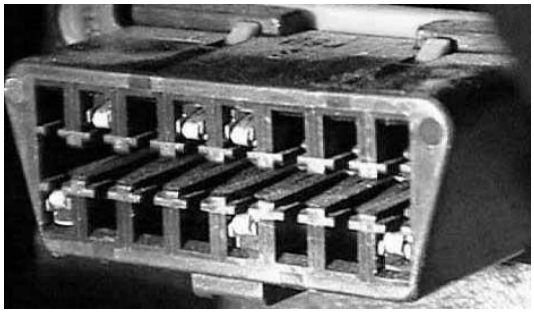


Figura 3 - Conector OBDII fêmea - Fonte: http://www.connector.pinoutsguide.com/ (29/04/2015)

A especificação OBDII define a padronização do conector de diagnostico conforme figura acima, o conector possui 16 pinos de acordo com a norma SAE J1962 ou ISO 15031-3.

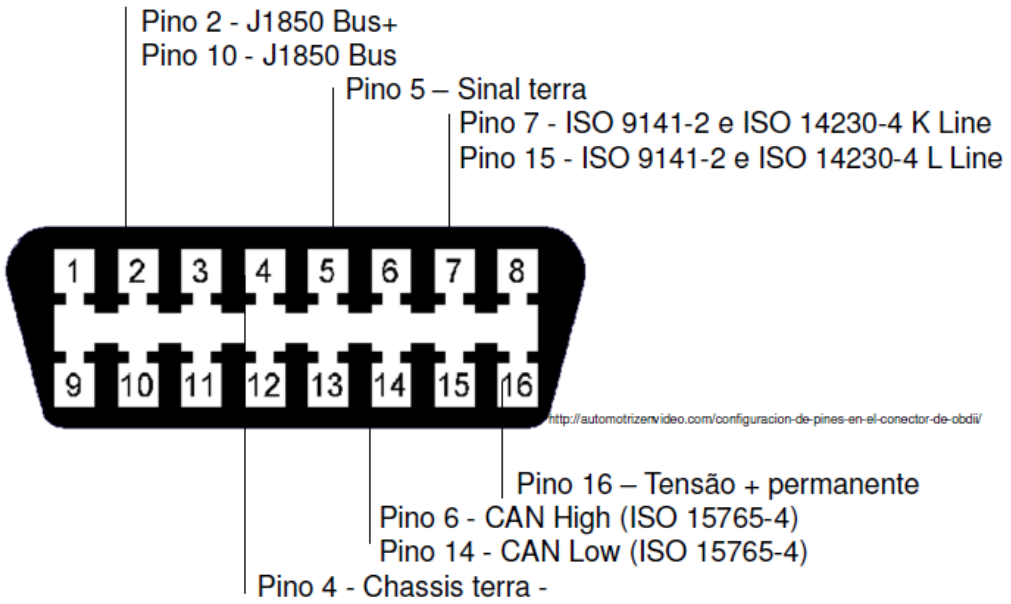


Figura 4 - Pinagem conector OBDII - Fonte: http://www.connector.pinoutsguide.com/ (29/04/2015)

2.2.1. Protocolo de comunicação

Protocolo é o conjunto de informações utilizadas para comunicação de dois dispositivos, cada protocolo se difere eletricamente e por formato de comunicação. A ECU utilizada no projeto utiliza o protocolo J1850 PWM, figura abaixo mostra uma estrutura de mensagem OBDII típica que são utilizados pelos padrões de protocolos SAE J1850, ISO 9141-2 e ISO 14230-4. Ela usa três bytes de cabeçalho como mostrado, para fornecer informações sobre a prioridade, receptor e o transmissor da mensagem.

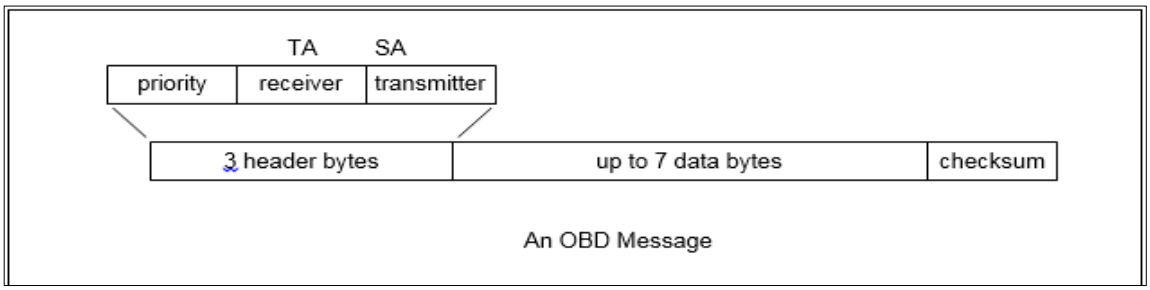


Figura 5 - Formato mensagem OBDII - Fonte: Datasheet – ELM327

O protocolo J1850 possui dois tipos de comunicação o VPW (*variable pulse width)* que utiliza largura de pulso variável para reconhecimento do bit dominante e recessivo e o PWM (*pulse width modulation*) que utiliza modulação de largura de pulso, conforme figura abaixo.

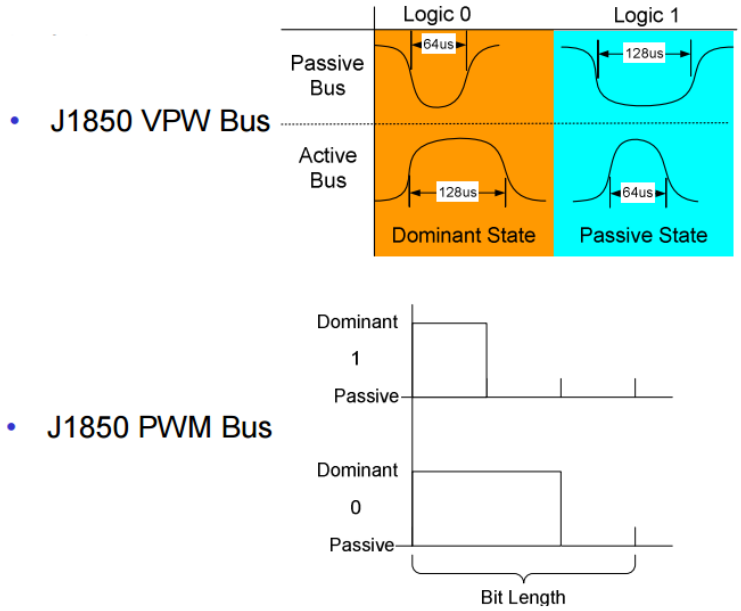


Figura 6 - Sinais do barramento J1850 - Fonte: In-Vehicle Networking - Marvin Stone

O J1850 PWM utilizado no projeto possui como características a utilização dos pinos 2, 4, 5, 10, 16 no conector OBDII, a velocidade de transferência é de 41,6Kbps, o comprimento da mensagem é restrita a 11 bytes incluindo o *checksum*, a tensão no barramento é de 5V e o comprimento máximo da arquitetura é de 40 metros contendo até 32 nós.

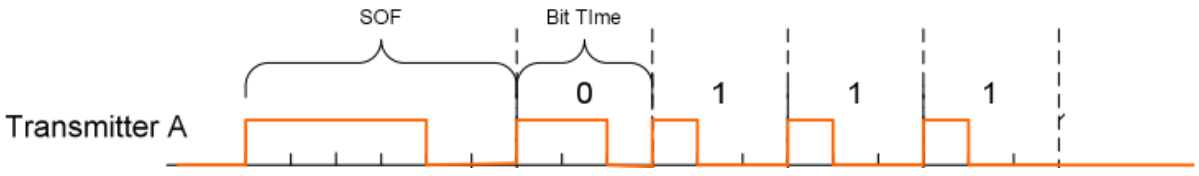


Figura 7 - Exemplo de transmissão de mensagem J1850 PWM - Fonte: In-Vehicle Networking - Marvin Stone

2.2.2. Interface de aquisição de dados OBDII

Foi utilizado no projeto o ODBII Scan conforme figura abaixo (leitor genérico), foi escolhido para o projeto devido ao custo baixo. Este conector utiliza um PIC18F2460 que simula o circuito integrado ELM327, este CI é projetado para traduzir os protocolos do OBDII realizando a leitura dos dados originados no veículo.

Figura 8- Scanner OBDII genérico utilizado

O scanner OBD *Scan* foi projetado para funcionar com comunicação via Bluetooth e para conseguirmos retirar os dados e utilizar no hardware do projeto foi implementado três cabos: RX / TX / GND conforme figura abaixo, com os cabos ligados a um MAX232 foi possível através de um *data logger serial* instalado no computador verificar todos os dados utilizados pelo OBD2 na comunicação via Bluetooth.

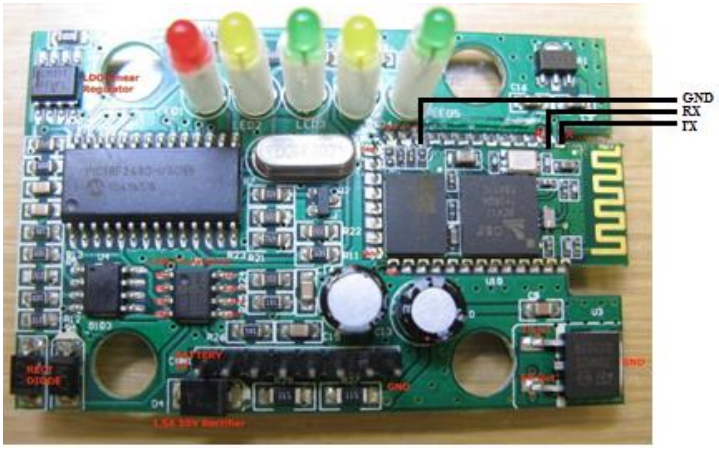


Figura 9 - Circuito interno do leitor genérico OBDII.

Para a leitura do conector ODBII *Scan* foi utilizado comando AT, é uma forma padrão na qual os modems estão configurados internamente e se comunicam com o PC. Com o terminal aberto, para enviar um comando para o OBDII o usuário precisa digitar o comando "AT+" seguido do parâmetro desejado, como exemplo "AT+SP0", com esse comando o usuário está definindo que o protocolo utilizado será reconhecido automaticamente pelo ODBII.

O projeto realizara a leitura dos seguintes dados:

- Velocidade: A leitura da velocidade é realizada no endereço 010D, esse dado será utilizado para verificar se o condutor está excedendo os limites de velocidade da legislação.
- Rotação: A leitura da rotação é realizada no endereço 010C, esse dado é utilizado para verificar se o condutor está utilizando o veículo de forma agressiva.
- Temperatura: A leitura da temperatura do liquido de arrefecimento do veículo é realizado no endereço 0105, com esse valor podemos verificar se a condição de trabalho do motor é a ideal.
- TPS: a leitura da posição angular da válvula borboleta é realizada no endereço 0111.

2.2.3. Características:

Modelo	ELM327
Alcance Bluetooth	10 m
Protocolo de envio de dados	RS232
Velocidade comunicação PC – OBD (kbps)	9600 / 38400
Protocolos leitura do veiculo	ISO15765-4(CAN) IS014230-4(KWP2000) ISO9141-2 J1850 VPW J1850 PWM
Comunicação	Bluetooth

Tabela 1- Características do leitor genérico OBDII - Fonte: http://www.southtrader.co.za/shop/bluetooth-obd-ii-scan-tool-for-motor-vehicles/ (01/05/2015)

2.3. Kit de desenvolvimento EVK2

O projeto utiliza o kit de desenvolvimento EVK2 que foi fornecido pela empresa Telit, esse kit possui inúmeras funções, tal como a transferência de dados via GSM e o sistema de localização GPS, essas são as tecnologias do modem que serão utilizadas no projeto. A empresa Telit forneceu o kit para utilização do projeto no concurso realizado por eles.

2.3.1. Conteúdo do Kit

- Placa Mãe - CS1139B:

A placa mãe inclui as interfaces básicas, tais como entrada de alimentação, suporte de cartão SIM, saído de monitor de áudio, RS-232 e USB; bem como um botão de Reset e interruptor de alimentação. O circuito implementado na placa mãe EVK2 é baseado no design de referência recomendados para componentes periféricos do módulo e conexões I / O (Telit.com).

- Placa adaptadora GSM/GPS – HE910

O HE910 é o um módulo compacto *multi-band* da série HSPA + LGA, com GPS e GSM / EGPRS e mede 28x28mm. A série HE910 (http://www.telit.com/products/cellular/3g/he910-series) é compatível com LTE, HSPA, GSM / GPRS em contrapartida da família CDMA / EVDO xE910. Apresentando HSPA + taxas de dados de sete bandas de até 21 Mbps *downlink* e 5,7 Mbps de *Upload*, ideal para *roaming* global de alto desempenho (Telit.com).

- Antena GSM
- *Sim Card*
- Fonte
- Cabo USB

2.3.2. Funcionalidades do Kit

Figura 10 - Modem Telit Evk2 - Fonte: Telit EVK2 UserGuide

O modem utilizado pelo grupo foi o EVK2, fornecido pela empresa Telit, seu ambiente robusto e flexível viabiliza o rápido desenvolvimento de aplicações para uma gama completa de família de módulos GSM/GPRS, UMTS/HSDPA e CDMA, possui recursos e periféricos básicos, tais como entrada de alimentação, SIM *card*, saídas de áudio, RS-232 e USB 1.1, botão de reset, chave liga/desliga, entre outros. Enfim, ela reuniu os recursos referenciais que uma solução deve possuir. Outros recursos específicos como, antena, entradas/saídas de uso geral (GPIO), ADC / DAC, UART, etc. São encontrados nas Interfaces que, através de 2 conectores machos, permitem a conexão com a aplicação do usuário, placas de extensão, ou outras ferramentas de desenvolvimento e equipamentos de medição. É possível controlar o módulo através da interface de Terminal que pode ser conectado via USB ou RS-232 (Telit).

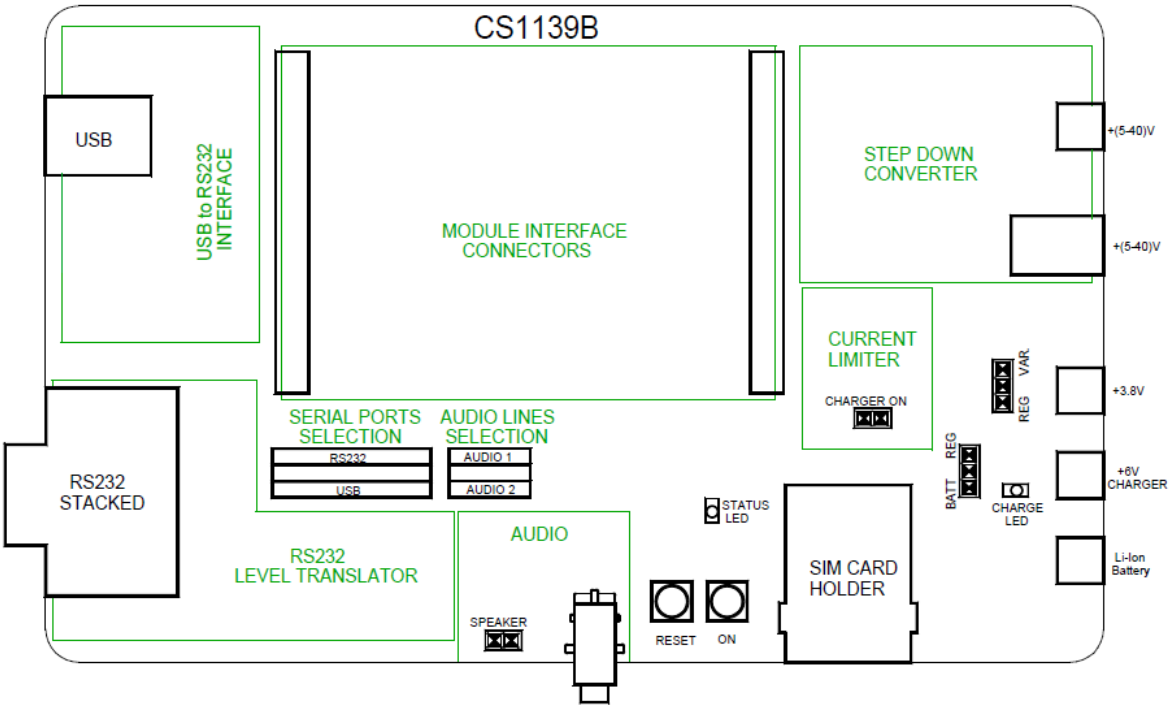


Figura 11 - Posicionamento dos circuitos Fonte: Telit EVK2 User Guide.

A conexão com o hardware de interface é feita através da conexão RS232 localizado na placa mãe CS1139B e nela está localizada a entrada da alimentação, conexão USB, SIM *card holder* onde é inserido o SIM *card* fornecido pela Telit e com acesso a rede de dados GSM e os pinos de entrada para placa auxiliar, além de outras funcionalidades que não serão comentas, pois não entra no escopo do projeto.

Figura 12 - Placa adaptadora GSM / GPS – HE910 – CS1467C

A comunicação via GSM será realizado através modem HE910 localizado na placa adaptadora CS1467C, os dados serão enviados através de conexão via socket para um servidor e este terá a função de exibir esses dados em um *web browser*. O modem GPS também está localizado na placa acima descrita e terá a função de enviar as coordenadas da latitude, longitude e horário mundial, com esses dados será possível estimar a localização do veículo em estudo. A comunicação entre esses módulos e o hardware do projeto será feita através de Terminal uma linguagem bastante utilizada na configuração de modens.

2.3.3. Especificações Técnicas:

- **Modem GSM:**

Taxa Transferência	21 Mbps *download* / 5,76 Mbps *upload*
Banda: Frequência	GSM/GPRS/EDGE: 850/900/1700/1800/1900 MHz / UMTS/HSPA: 800/850/900/AWS1700/1900/2100 MHz
Controle	Terminal
Porta	USB
Alimentação	3,1 V a 4,5 V
Temperatura de operação	-40°C a +85°C
Consumo	0,4W
Interface	10 I/O, USB 2.0, 2 UART, SPI, I2C, SIM
Conexão	UDP/TCP/FTP/SMTP e envio SMS
Protocolo	TCP e UDP

Tabela 2 - Especificação técnica modem GSM – HE910 Fonte: Datasheet – HE 910 Series

- **Modem GPS:**

Sensibilidade	Aquisição	- 147 dBm
	Rastreamento	- 163 dBm
	Navegação	- 160 dBm
Tempo inicialização	*Cold*	42 s
	Warm	30 s
	Hot	1.8 s
Exatidão	3 m	
Antena	Interna e externa	
Canais	48 Canais de recepção GPS	
Interface	UART, I²C, SPI	
Memoria	Flash, EEPROM	
Protocolo	NMEA	
Alimentação	1,75 V a 1,90 V	
Temperatura de operação	-40°C a +85°C	
TTFF	90% @ -180 dBm	
Hot Start	1s	
Cold Start	< 35 s	
Consumo	0,4W	
Controle	Terminal	

Tabela 3- Especificação técnica modem GPS – HE910

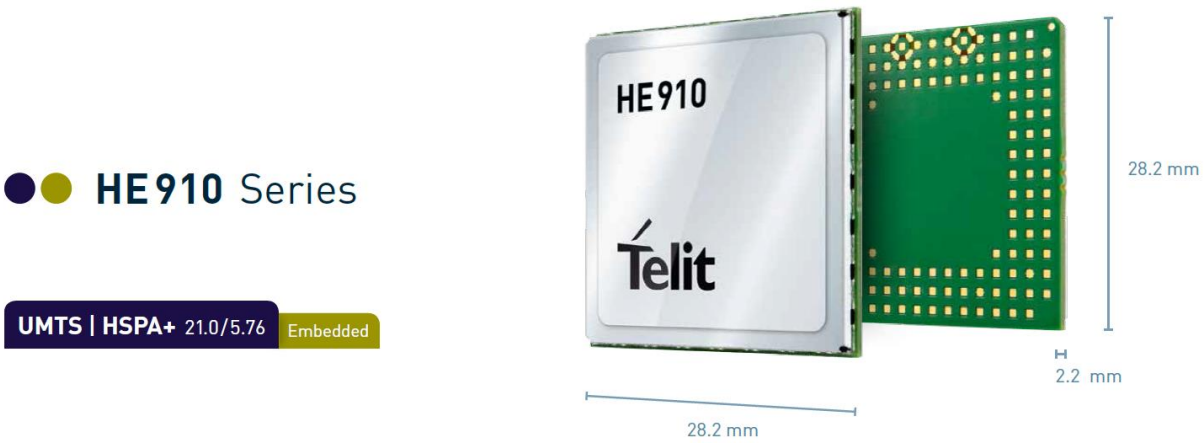


Figura 13 - Modem Telit HE910 Fonte: Datasheet - HE 910 Series

2.4. ECU Ford EEC–V

Desde que as novas normas ambientais (OBDII) entraram em vigor nos Estados Unidos, em meados dos anos 90, a *Ford Motor Company* necessitou desenvolver um novo sistema de controle eletrônico para os motores de seus carros. Ela reaproveitou a plataforma do já conhecido e amplamente utilizado EEC-IV que ainda participava do sistema OBDI Para desenvolver o seu sucessor, a EEC-V.

2.4.1. Novos componentes.

Além dos componentes já utilizados no controle do motor, foram acrescentados ao sistema sinais adicionais de entrada para o PCM com o intuito de bater a nova regulamentação ambiental, no caso da EEC-V foram estes:

- Monitor do Eletro ventilador do radiador;
- Solicitação de comunicação serial;
- Reprogramação da EEPROM;
- *Sonda Lambda* pós-catalisador.

Figura 14 – EEC-V

3. Metodologia

Para o desenvolvimento do projeto em questão o conteúdo foi dividido nas seguintes etapas:

- Analise do Scanner OBD;
- Comunicação scanner OBD2;
- Comunicação modem EVK2;
- Criação hardware;
- Criação Software.
- Criação do servidor;

3.1. Analise do Scanner ODBII – ELM 327

O scanner OBD genérico utiliza comunicação via Bluetooth e para sua configuração foi selecionado a comunicação RS232 através de um terminal, via computador assim conseguimos enviar os comandos.

3.1.1. Seleção de Protocolo:

- 1° comando - AT Z (reset total)
- 2° comando - AT SP __ (Seleciona protocolo)

Protocolo	Descrição
0	Automático
1	SAE J1850 PWM (41.6 *kbaud*)
2	SAE J1850 VPW (10.4 *kbaud*)
3	ISO 9141-2 (5 *baudinit*)
4	ISO 14230-4 KWP (5-*baud init*)
5	ISO 14230-4 KWP (*fast init*)
6	ISO 15765-4 CAN (11 bit ID, 500 *kbaud*)
7	ISO 15765-4 CAN (29 bit ID, 500 *kbaud*)
8	ISO 15765-4 CAN (11 bit ID, 250 *kbaud*)
9	ISO 15765-4 CAN (29 bit ID, 250 *kbaud*)
A	SAE J1939 CAN (29 bit ID, 250* *kbaud*)
B	*User1* CAN (11* bit ID, 125* *kbaud)*

Tabela 4- Seleção de protocolo OBD2 Fonte: Datasheet – ELM327

3.1.2. Formato das mensagens:

Para iniciar a comunicação com o veículo e o OBDII primeiro inserimos o comando "ATZ" para reset das operações e após isso selecionamos o protocolo utilizado a tabela acima e digitando os comandos no terminal "AT SP x" (onde o x representa o protocolo), no projeto como esse envio é feito pelo PIC então usamos a seleção automática "AT SP 0" assim o OBD *Scan* seleciona automaticamente o protocolo do veículo.

Para requisitar a leitura de um endereço correspondente a um parâmetro do veículo basta digitar o endereço e a resposta será dada referenciando o primeiro o endereço e depois o valor que lá contem. A tabela abaixo contém alguns comandos utilizados nos testes com o ELM327 com a resposta obtida e a explicação resumida do comando solicitado.

Comando	Resposta	Comentário
ATZ	ELM237 V1.5	Reset - versão do software
AT SP 0	OK	Escolha do protocolo utilizado - automático
AT I	ELM237 V1.5	Mostra a versão do Software
AT DP	AUTO, ISO 15765 (CAN 11/500)	Mostra o protocolo utilizado
01 04	41 04 00	Carga do motor
01 05	41 05 00	Temperatura do líquido de arrefecimento
01 0B	41 0B 00	Pressão no coletor de admissão
01 0C	41 0C 00	Rotação atual do motor
01 0D	41 0D 00	Velocidade instantânea do veiculo
01 0E	41 0E 00	Avanço do ângulo de ignição
01 0F	41 0F 00	Temperatura do Ar
01 11	41 11 32	Posição da válvula borboleta - TPS
03	43 00	DTC – Sem DTC (erros armazenados)
AT BD	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Mostra o valor presente no buffer de memoria

Tabela 5 - Comandos utilizados nos testes OBD Scan – ELM327

Com os valores obtidos em hexadecimal é necessária a conversão para um valor real a ser entendido e exibido.

- Temperatura do liquido de arrefecimento (°C):

Solicitar temperatura do liquido de arrefecimento – PID 05 do modulo – 01: (>01 05) caso tenhamos uma resposta do tipo: (>41 05 7B) significa:

41 → 40 (resposta) + 1 (modulo 1);

05 → PID 05;

7B → 7B hexa = 123 decimais. Representa a temperatura em graus Celsius, mas com o deslocamento para permitir temperaturas abaixo de zero:

$$\mathbf{123 - 40 = 83}℃$$

Equação 2.1 – Calculo Liquido de Arrefecimento

- Rotação do motor (RPM):

Solicitar a rotação instantânea do motor – PID 0C do modulo – 01: (>01 0C) caso tenhamos uma resposta do tipo: (>41 0C 1A F8) significa:

41 → 40 (resposta) + 1 (modulo 1);

0C → PID 0C;

1A F8 → 1A F8 hexa = 6904 decimais.

Representa a rotação do motor multiplicada por quatro e para visualizar a rotação em RPM (rotação por minuto):

$$\mathbf{6904 \div 4 = 1726}\ \boldsymbol{RPM}$$

Equação 2.2 – Calculo rotação do motor

- Posição da válvula borboleta - TPS (%):

Solicitar a carga instantânea do motor – PID 11 do modulo – 01: (>01 11) caso tenhamos uma resposta do tipo: (>41 11 3C) significa:

41 → 40 (resposta) + 1 (modulo 1);

04 → PID 04;

1E → 3D hexa = 61 decimais.

Representa o valor do TPS do veículo dividida em oito bytes e para obter o valor em porcentagem:

$$(\mathbf{61} * \mathbf{100}) \div \mathbf{255} = \mathbf{23}\ \%$$

Equação 2.3 – Calculo carga do motor

- Velocidade do veículo (km/h):

Solicitar a carga instantânea do motor – PID 0D do modulo – 01: (>01 0D) caso tenhamos uma resposta do tipo: (>41 0D48) significa:

41 → 40 (resposta) + 1 (modulo 1);

0D→ PID 0D;

48→48 hexa = 72 decimais.

Representa o valor da velocidade do veículo onde pode variar de 0 a 255:

$$72\ \boldsymbol{Decimal} = \mathbf{72}\ \boldsymbol{km/h}$$

Equação 2.4 – Calculo velocidade do veiculo

- Temperatura do ar de admissão (°C):

Processo idêntico ao cálculo de Temperatura do liquido de arrefecimento (°C).

Com os dados obtidos com a leitura dos códigos vindo do ODBII *Scan* o próximo passo é analise do Kit de desenvolvimento EVK2 fornecido pela Telit a fim de enviar as mensagens obtidas para o servidor de internet.

3.2. Analise do Kit de desenvolvimento EVK2

O kit de desenvolvimento fornecido pela Telit o EVK2 possui inúmeras funções desde a envio de SMS via GSM a pequenas configurações em linguagem Python (Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte), mas nesta parte abordaremos apenas os tópicos utilizados no projeto.

3.2.1. Instalação:

Para iniciarmos os testes primeiramente obtemos o registro do SIM *card* presente no kit, assim o trafego de dados na rede GSM foi liberado, com restrição no limite de trafego de dados. Para instalação do software no computados foi necessário a instalação dos drivers de comando do kit, material fornecido pela Telit. Com os drivers de acesso instalados é preciso apenas conectar o modulo EVK2 com auxílio de um cabo USB ao computador e as portas são instaladas automaticamente com isso o acesso aos recursos que o kit de desenvolvimento estão liberados.

As configurações são feitas utilizando comandos AT com a utilização de qualquer terminal de visualização de porta serial no computador. A Telit oferece junto com o kit um programa que possui as funcionalidades de um terminal com uma janela de comandos previamente configurados de fácil acesso ao usuário.

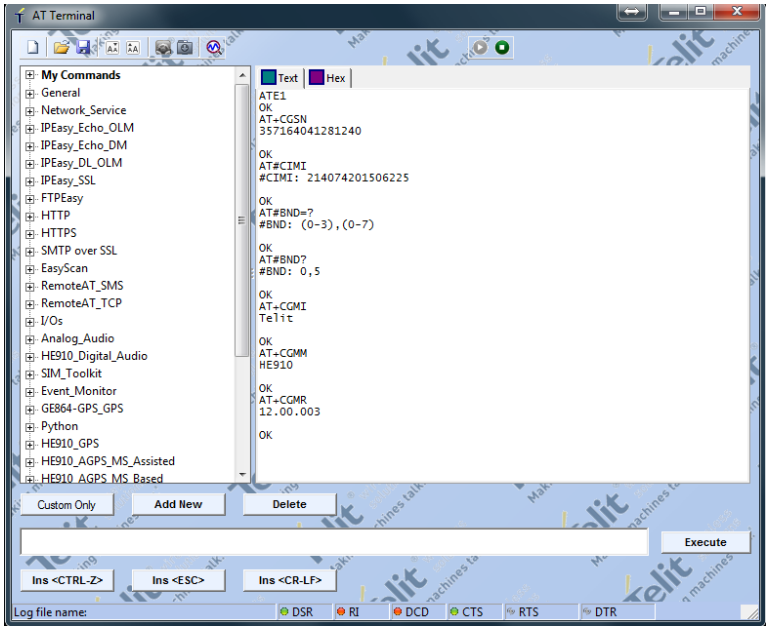


Figura 15 - Programa Telit – Terminal – Via serial USB

O projeto tem objetivo o envio de pacote de dados via GSM para internet, o modo de envio escolhido através do modem EVK2 foi a transferência via socket PDP, esse tipo de comunicação cria um túnel entre servidor e cliente onde é transferido o pacote de dados, após a transferência é necessário fechar esse túnel. Para que essa transferência de dados aconteça é necessário registrar o modem na rede e com isso obter um número de IP, configurar o socket de acordo com as suas necessidades, abrir o socket, enviar os dados e fechar o socket. Abaixo serão descritos os comandos utilizados neste projeto.

3.2.2. Comandos

AT+CREG	Relatório de Registro na Rede. Comando utilizando para verificar se o modulo está registrado na rede de dados	
	0	Não registrado – ME não valido;
	1	Registrado;
	2	Não registrado – ME valido;
	3	Registro negado;
	4	Erro desconhecido;
	5	Registrado – *roamming*;
	AT+CREG? - Mostra o parâmetro atual; AT+CREG=? - Mostra as configurações possíveis;	

Tabela 6 - Comando +CREG – Modem EVK2

AT+CSQ	Qualidade do Sinal. Verifica a qualidade do sinal GSM recebido;	
	0	(-113) dBm ou menor;
	1	1 - (-111) dBm;
	2 – 30	(-109) dBm. (-53) dBm / 2 dBm por passo;
	31	(-51) dBm ou maior;
	99	Não reconhecido ou não detectado;
	0	(-113) dBm ou menor;
	AT+CSQ?- Mostra o parâmetro atual; AT+CSQ=?- Mostra as configurações possíveis;	

Tabela 7 - Comando +CSQ – Modem EVK2

AT+IPR	Velocidade de comunicação. Define a velocidade de comunicação durante o modo de operação.	
	0	9600
	300	19200
	1200	38400
	2400	57600
	4800	115200
	0	9600
	AT+IPR? - Mostra o parâmetro atual; AT+IPR=? - Mostra as configurações possíveis;	

Tabela 8 - Comando +IPR – Modem EVK2

AT&F	Padrão de fábrica. Retorna as configurações de fábrica.	
	0	Padrão de fábrica com algumas ressalvas;
	1	Padrão de fabrica total;

Tabela 9 - Comando &F – Modem EVK2

AT&W	Salva configuração Salva modificação nas configurações para possível reset no modem.	
	0	Salva configuração atual.

Tabela 10 - Comando &W – Modem EVK2

AT+CGCONT <x>,<y>,<z>	Definição do contexto PDP. Comando set especifica valores de parâmetro contexto PDP para um contexto PDP identificado pelo parâmetro de identificação de contexto (local).		
	<x>	1	Parâmetro numérico que especifica uma definição determinado contexto PDP.
	<y>	"IP"	Especifica o tipo de protocolo de dados;
	<z>	"Internet m2m.air.com"	Selecionar o GGSN ou a rede de dados por pacotes externo a ser utilizado;
	AT+CGDCONT? - Mostra o parâmetro atual; AT+CGDCONT =? - Mostra as configurações possíveis;		

Tabela 11 - Comando +CGDCONT – Modem EVK2

AT+ SGACT <x>,<y>	Ativação de contexto PDP. : Comando de execução é usado para ativar ou desativar qualquer contexto GSM ou o contexto PDP especificado; Obs.: Após a execução deste comando o programa retorna o valor do IP usado pelo modem na rede		
	<x>	1	Especifica que o contexto é PDP
	<y>	1	Ativa o contexto;
	AT+ SGACT? - Mostra o parâmetro atual; AT+ SGACT=? - Mostra as configurações possíveis;		

Tabela 12 - Comando +CGACT – Modem EVK2

AT#SCFG <x>,<y>,<z> <w>,<q>,<k>	Configura socket. Configuração dos parâmetros dos sockets.		
	<x>	1	Identifica o socket a ser configurado
	<y>	0	Conexão GSM
	<z>	30	Tamanho do pacote enviado (bytes)
	<w>	90	Tempo de inatividade (s)
	<q>	600	Tempo de espera de conexão (s)
	<k>	50	Tempo de espera por byte (s)
	AT#SCFG? - Mostra o parâmetro atual; AT#SCFG=? - Mostra as configurações possíveis;		

Tabela 13 - Comando #SCFG – Modem EVK2

AT#SD <x>,<y>,<z> <w>	Conexão via socket. Execução de comando abre uma conexão remota via socket.		
	<x>	1	Especifica o socket a ser conectado
	<y>	0	Padrão TCP;
	<z>	10510	Especifica a porta a ser conectada
	<w>	177.102.220.XXX	Endereço de IP a ser conectado
	AT#SD?- Mostra o parâmetro atual; AT#SD=? - Mostra as configurações possíveis;		

Tabela 14 - Comando #SD – Modem EVK2

AT#SSEND	. Envio mensagem via socket. Execução de comando envia mensagens via socket. Após execução deste comando todos os caracteres digitados serão enviados ao servidor, essa conexão se encerra se for pressionado <ctrl+z> ou enviado os caracteres <+++>.		
	<x>	1	Abre túnel para cone-xão
	AT#SSEND=?- Mostra as configurações possíveis;		

Tabela 15 - Comando #SSEND– Modem EVK2

AT#SRECV <x>,<y>	. Ler mensagens enviadas. Execução de comando permite que o usuário leia os dados enviados através de um socket conectado.		
	<x>	1	Especifica o socket
	<y>	20	Especifica o tamanho da mensagem a ser lida (bytes);
	AT# SRECV=?- Mostra as configurações possíveis;		

Tabela 16 - Comando #SRECV – Modem EVK2

AT#SCFG <x>,<y>,<z> <w>,<q>,<k>	. Configura GPS. Configuração dos parâmetros dos sockets.		
	<x>	1	Ativa protocolo NMEA
	<y>	0	*Global Positioning System Fix Data*
	<z>	0	Posição global - Latitude/Longitude
	<w>	0	GPS/GLONASS DOP e ativa satélites
	<q>	0	Tempo de espera de conexão (s)
	<k>	0	*recommended Minimum Specific GNSS Data*
	<j>	0	*Course Over Ground and Ground Speed*
	AT$GPSNMUN?- Mostra o parâmetro atual; AT$GPSNMUN=? - Mostra as configurações possíveis;		

Tabela 17 - Comando #SCFG – Modem EVK2

AT$GPSACP <x>,<y>,<z>	. Configura GPS. Configuração dos parâmetros dos sockets.		
	<x>	hhmmss.sss	Hora mundial
	<y>	ddmm.mmmm N/S	Latitude (Norte/Sul)
	<z>	dddmm.mmmm E/W	Longitude (Leste/Oeste)
	AT+ SCFG? - Mostra o parâmetro atual; AT+ SCFG=? - Mostra as configurações possíveis;		

Tabela 18 - Comando $GPSACP – Modem EVK2

AT$GPSR	. Configura GPS. Configuração dos parâmetros dos sockets.	
	0	Volta aos padrões iniciais.
	1	*Coldstart*
	2	*Warmstart*
	3	*Hotstart*
	AT+ GPSR? - Mostra o parâmetro atual; AT+ GPSR =? - Mostra as configurações possíveis;	

Tabela 19 - Comando $GPSR – Modem EVK2

Com os valores definidos o próximo passo foi criar a sequência de programação a ser executada pelo hardware do projeto, a fim de receber os valores do OBDII, armazená-los em vetores e enviar via socket PDP para o servidor.

AT+CGDCONT=1,”IP”,INTERNETM2M.AIR.COM

AT#SGACT=1,1

AT$GPSACP

AT#SD=1,0,10510,177.102.220.XXX

AT#SSEND=1

ENVIA PACOTES,

+++ (FEXA SOCKET)

3.3. Giga de testes ECU Fiesta.

3.3.1. Velocidade

O sensor de velocidade do veiculo ou VSS (*Vehicle Speed Sensor)* é um sensor de ação pôr efeito HALL, o qual fornece um sinal de onda quadrada para o PCM, cuja frequência será proporcional à velocidade do veículo. Localiza-se na saída de velocidade da caixa de câmbio. É composto de um imã permanente, circuito integrado HALL e um rotor metálico, fixado a um eixo. Quando este eixo gira, movimentando o rotor, provoca uma variação de fluxo de corrente no circuito HALL, o qual emitirá um sinal de massa para o PCM. O mesmo, a partir da frequência de recepção deste sinal de massa, consegue determinar a velocidade do veiculo. O sensor VSS é energizado diretamente pelo PCM, gerando um sinal de 12,00 volts

DC, toda vez que o mesmo receber estes impulsos negativos. (Sistema EEC-V ZETEC RO-CAM - Flavio Xavier - Elói Training).

Sensor VSS	*PCM 60 pinos*	*Função*
1	20	Linha de massa
2	03	Linha de sinal
3	37/57	12,00 volts DC

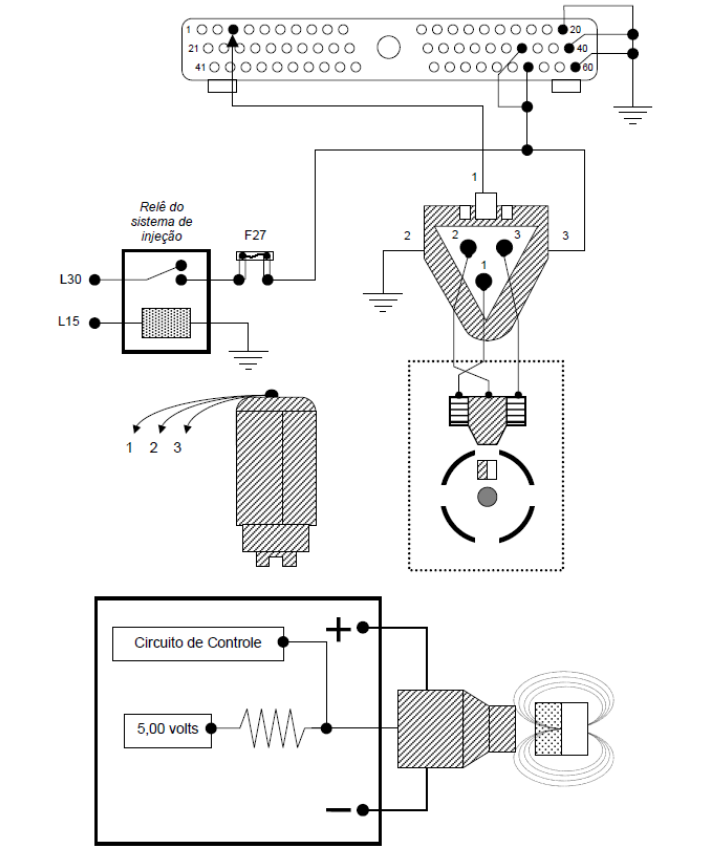


Figura 16- Sistema Sensor Velocidade ECU Fiesta - Fonte: Sistema EEC-V ZETEC ROCAM -Flavio Xavier

Para simular a frequência do veiculo foi utilizado um circuito com o CI555, que foi ajustado para uma frequência de 15 à 700 Hz onde podemos observar a variação de velocidade aferida pelo Scanner OBDII. Quanto maior a frequência maior a velocidade.

3.3.2. Rotação

A rotação utilizada na ECU é aferida a partir de uma roda fônica localizada no eixo de manivelas do motor de 36 dentes sendo 2 ausentes representando a falha, com esse sistema a ECU sabe exatamente em qual posição estão os cilindros do motor e assim executa a função de ignição nas velas no momento exato de máxima compressão e injeção de combustíveis nos bicos injetores que está na fase de admissão. Para simular esse sinal foi utilizado um micro-controlador (PIC18F2220) onde a variação é simulada a partir da leitura na entrada analógica

de um potenciômetro quanto maior o valor lido menor a frequência simulada, representa pela tabela 6.

RPM	PERIODO		FALHA	RPM	PERIODO		FALHA
	ALTA	BAIXA			ALTA	BAIXA	
800	250	1000	2500	3400	60	240	600
1000	200	800	2000	3600	56	224	560
1200	166	664	1660	3800	53	212	530
1400	142	568	1420	4000	50	200	500
1600	125	500	1250	4200	48	192	480
1800	111	444	1110	4400	45	180	450
2000	100	400	1000	4600	43	172	430
2200	90	360	900	4800	42	168	420
2400	83	332	830	5000	40	160	400
2600	77	308	770	5200	38	152	380
2800	71	284	710	5400	37	148	370
3000	67	268	670	5600	36	144	360
3200	63	252	630	5800	34	136	340
				6000	33	132	330

Tabela 20 - Gerador de falha da roda fônica

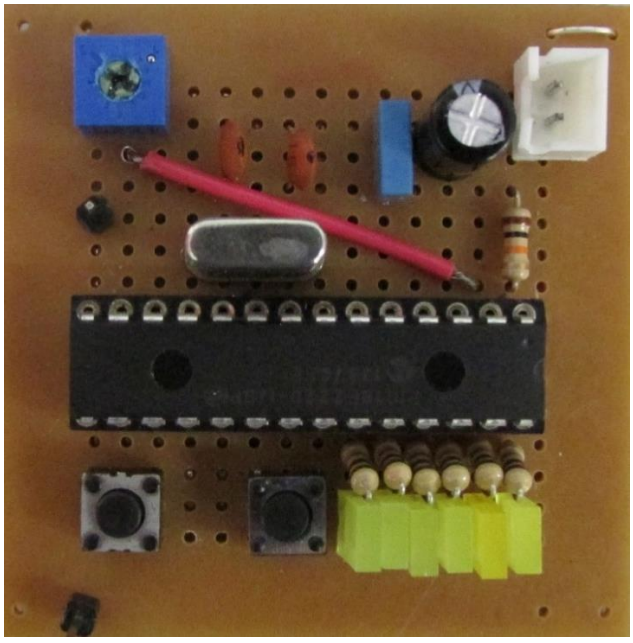


Figura 17 - Circuito finalizado Roda fônica

Sensor CKP	PCM 60 pinos	Função
1	56	Sinal do sensor
2	55	Linha de massa

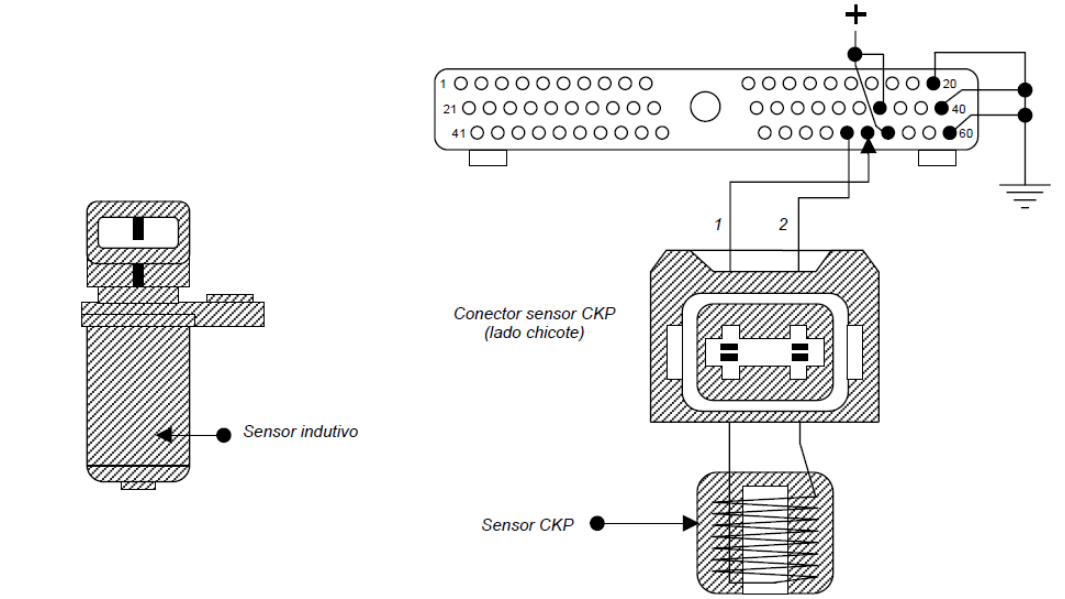


Figura 18-Sistema Sensor Rotação ECU Fiesta - Fonte: Sistema EEC-V ZETEC ROCAM -Flavio Xavier.

3.3.3. Temperatura do liquido de arrefecimento

O sensor ECT é constituído internamente pôr um resistor térmico de tipo NTC (*Negative Temperature Coeficient*) onde a forma de leitura da temperatura do motor é inversamente proporcional à resistência do sensor, ou melhor, dizendo, quando aumenta a temperatura do sensor, a resistência diminui, alterando o valor de tensão de retorno ao PCM. Quando o motor está frio, a resistência é alta, portanto ao PCM irá aumentar a voltagem do circuito. Quando o motor está quente, a resistência é baixa, o PCM irá diminuir a voltagem do circuito. O PCM, adquirindo a tensão de trabalho do circuito, consegue determinar a temperatura do líquido de arrefecimento, efetuando uma correção do tempo de injeção, com a lógica de aumentá-lo com motor frio (mistura rica) e diminuí-lo com motor quente (mistura pobre). O sinal do ECT influencia cálculo do avanço de ignição, controle de ar em marcha lenta, sistema de controle de emissões evaporativas e controle do ar condicionado. A variação de tensão do circuito varia de 0,00 a 5,00 volts DC. A desconexão do sensor simula motor frio (aumenta a resistência/aumenta a voltagem) e o curto-circuito simula motor quente (diminui a resistência/diminui a voltagem). (Sistema EEC-V ZETEC ROCAM - Flavio Xavier - Elói Training).

Sensor ECT	PCM 60 pinos	Função
1	7	Sinal do sensor
2	46	Linha de massa

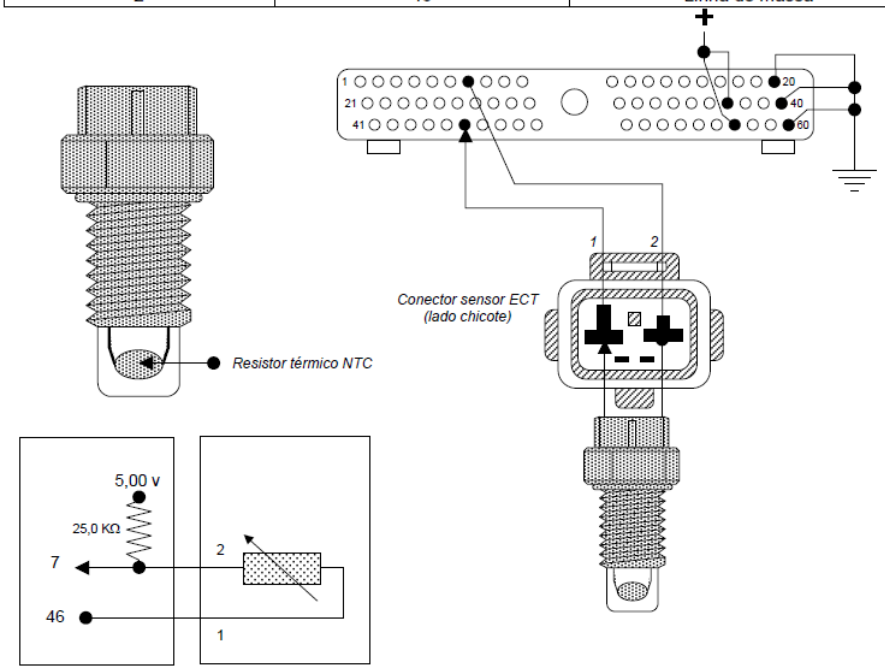


Figura 19-Sistema Sensor Temperatura ECU Fiesta - Fonte: Sistema EEC-V ZETEC ROCAM -Flavio Xavier

Para simular a temperatura do veículo foi utilizado um potenciômetro de 100K e conforme sua variação a temperatura aferida se alterava.

3.3.4. Posição da válvula borboleta

O sensor de posição de borboleta TPS (*Throttle Position System*) é composto de um potenciômetro cuja parte móvel é comandada pelo eixo de borboleta, a partir do pedal do acelerador. Uma tomada com três terminais (1, 2 e 3) situada na peça efetua a ligação com o PCM. A mesma alimenta o sensor, durante o seu funcionamento, com uma tensão de 5,00 volts DC. O sinal medido é a posição da borboleta, da mínima a máxima abertura, para o controle de injeção de combustível. Com a borboleta fechada um sinal elétrico é enviado O PCM, a qual realizará o reconhecimento de marcha lenta. A medida que acelera-se o motor, altera-se a posição do potenciômetro, alterando o valor da resistência do circuito, até a máxima abertura. O PCM, com base neste sinal, controla a quantidade de combustível a ser injetado. Algumas estratégias de funcionamento baseiam-se neste sinal, entre elas a condição CUT-OFF (corte de combustível em desaceleração), com base no número de rotações do motor e borboleta em posição fechada. Não é necessário efetuar nenhum tipo de regulagem na sua posição angular, já que o próprio PCM, que através de adequadas lógicas de auto adaptação, reconhece as condições de borboleta fechada (IAC) ou completamente aberta (WOT). (Sistema EEC-V ZETEC ROCAM - Flavio Xavier - Elói Training).

Sensor TP	*PCM 60 pinos*	*Função*
1	26	5,00 volts DC
2	47	Linha de sinal
3	46	Linha de massa

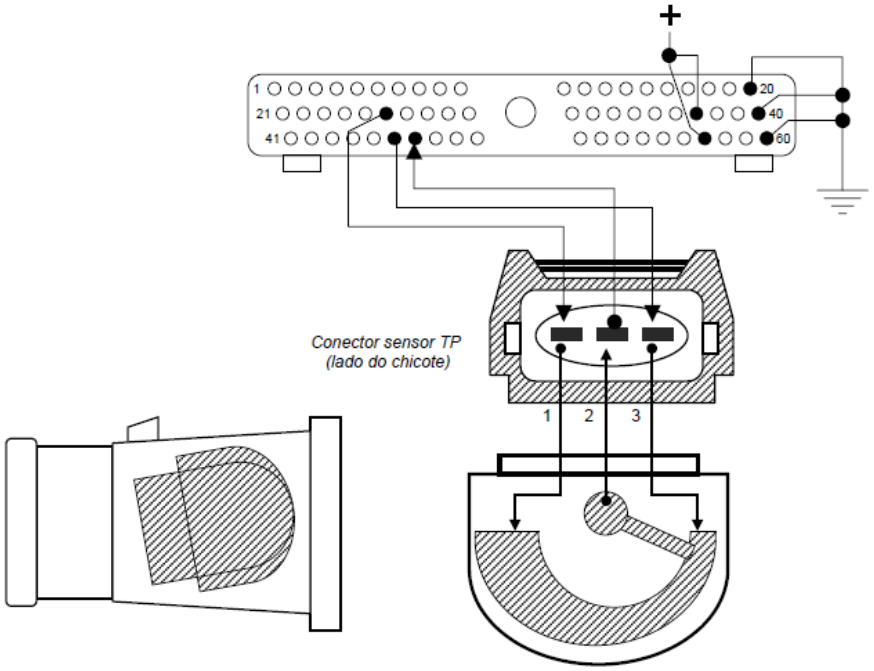


Figura 20-Sistema Sensor posição válvula borboleta ECU Fiesta - Fonte: Sistema EEC-V ZETEC ROCAM -Flavio Xavier

Para simular a posição da válvula borboleta do veiculo foi utilizado um potenciômetro de 1K e conforme a variação da tensão no pino de entrada a porcentagem de abertura aferida se altera.

3.4. Hardware do projeto

Para que o hardware do projeto fosse robusto para suportar o ambiente hostil de um veículo e diminuir as interferências externas colocamos os seguintes materiais:

Quantidade	Componente	Referencia
2	Capacitor - 15 pF	C8/C12
4	Capacitor - 1 µF	C7/C9/C10/C11
3	Capacitor - 100 nF	C2/C3/C6
2	Capacitor Eletrolítico - 100 µF	C4/C5
1	Capacitor Eletrolítico - 1000 µF - 16 v	C1
1	Conector KK 2,5 - 3 vias	CN1
2	Conector KK 2,5 - 4 vias	CN3/CN4
1	Conector KK 2,5 - 6 vias	CN2
2	Diodo - ln4007	D1/D2

10	Led - 3mm	LD1 a LD10
1	LM7805	CI1
1	MAX232	CI3
1	PIC 18F4620	CI2
2	*Push Button*	S1/S2
1	Resistor - 10 k	R1
2	Resistor - 270r	R3/R4
1	Resistor - 330r	R2
1	Cristal - 20 Mhz	XTAL

Tabela 21 - Lista de Materiais

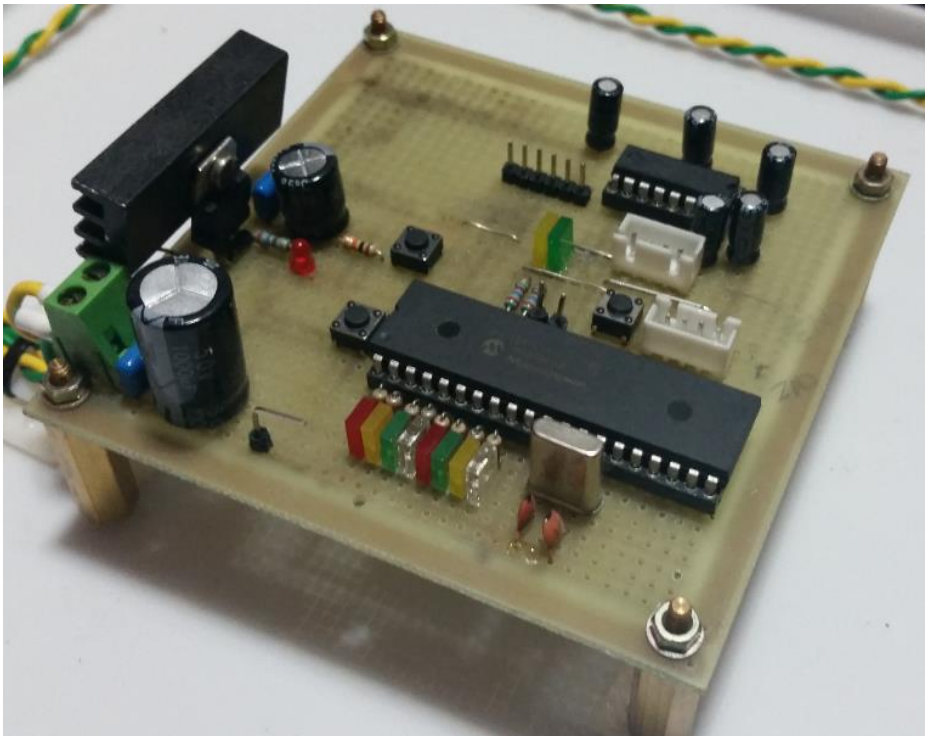


Figura 21 - Hardware finalizado

3.5. Software do projeto

O software do projeto foi produzido com o compilador CCS, que possui várias bibliotecas de programas de exemplo para muitas aplicações comuns. A seguir será explicado o passo a passo na execução do programa.

Para execução dos cálculos dos valores aferidos foi feita a função abaixo, pois as variáveis estão em forma de “string” e com sua execução o valor da variável de retorno é a quantidade correspondente.

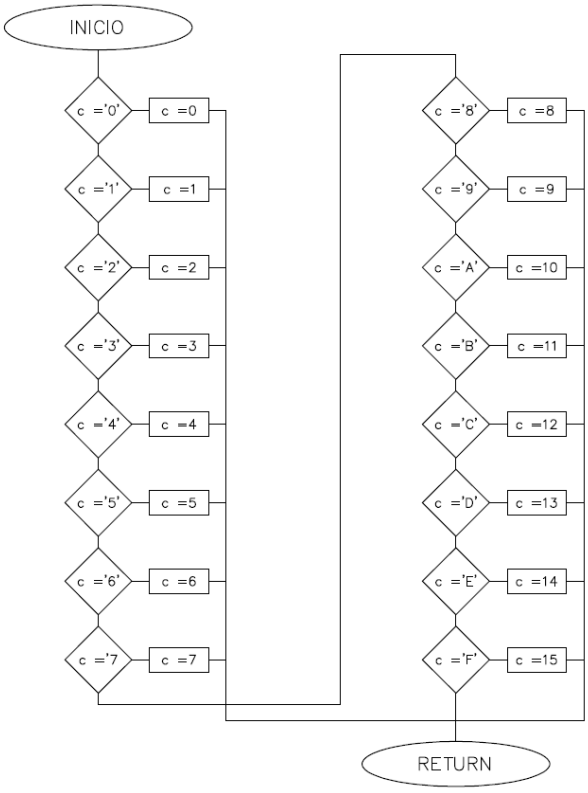


Figura 22 - Transforma String em numero

Para imprimi-lo os caracteres é preciso transformar os números obtidos em string, assim realizados na função abaixo.

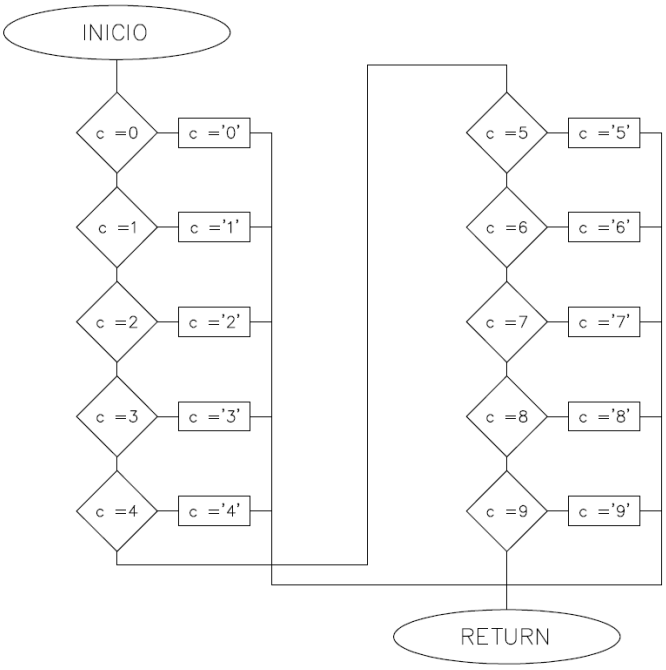


Figura 23 - Transforma número em String

Primeiramente o valor do indexador de caractere é colocado em zero para garantir que o valor recebido seja gravado no primeiro caractere do buffer de entrada, pois os dados recebidos são tratados na interrupção externa do microcontrolador, após essa execução os dados do buffer de entrada são zerados e enviados o endereço correspondente ao comando indicando o valor correspondente ao valor a ser lido, após enviar essa informação o scanner responde e o valor é gravado no buffer de entrada, para verificação do dado recebido o programa procura os caracteres “41” (corresponde a resposta do valor perguntado ao scanner OBDII) no buffer, caso o valor recebido não seja o correto um LED de erro será aceso e só irá se apagar caso o valor recebido esteja correto. O valor obtido é transformado de string para número para ser realizado as devidas contas e envio para o servidor

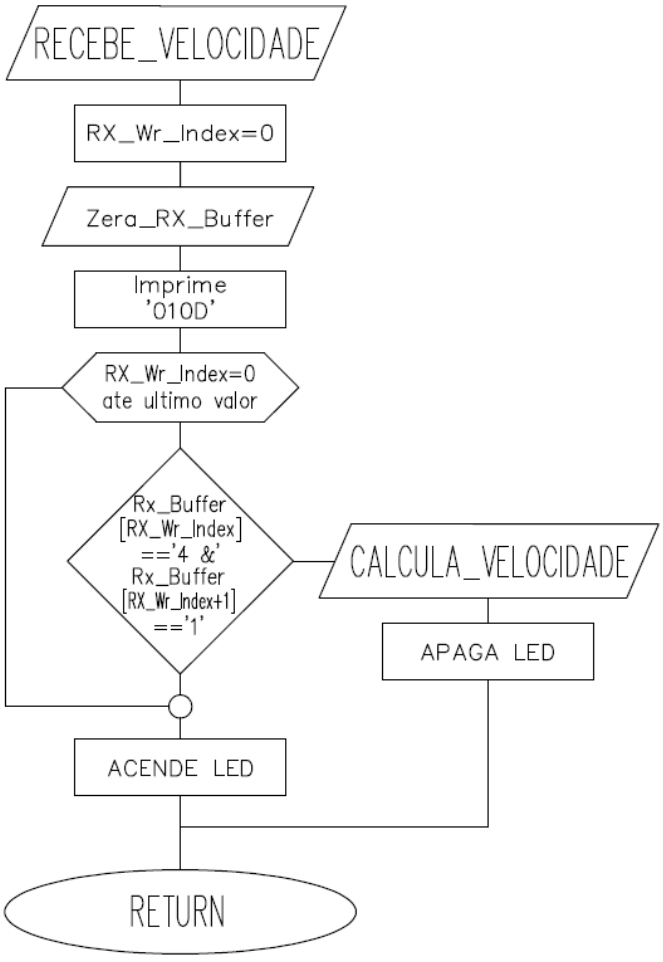


Figura 24 - Envio de dados e tratamento.

O cálculo das operações é descrito tomando como exemplo a velocidade do veículo (figura abaixo). Para realizar a captura do valor da informação a ser convertida o programa toma como base o valor do último index (Rx_Buffer[RX_Wr_Index]=='4') realizado que do

caso foi a resposta do valor a ser lido “41” assim o caractere de reposta está no vetor “+4” e “+5”, com isso o valor é convertido de string para número, o valor recebido é tratado e transformado na velocidade atual do veículo, para que esse valor seja enviado para o modem EVK2 ele é transformado para string novamente e enviado para a mensagem.

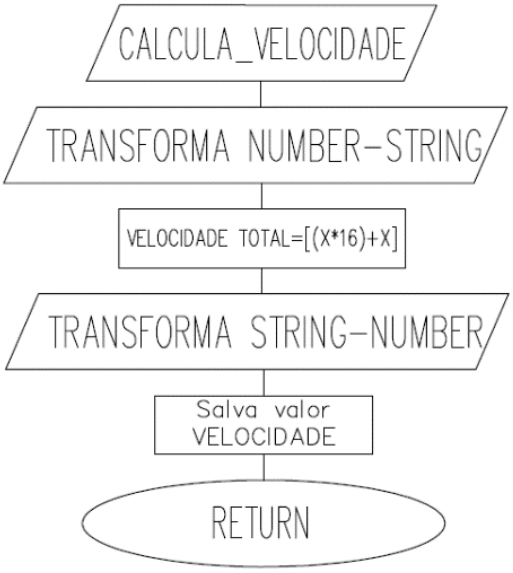


Figura 25- Cálculo dos valores obtidos

Para amostragem dos dados obtidos e calculados no modem EVK2 os caracteres devem ser enviados um a um e para isso foi feita a função descrita na figura abaixo, tomando como exemplo a velocidade e executada da mesma forma para todos os valores aferidos.

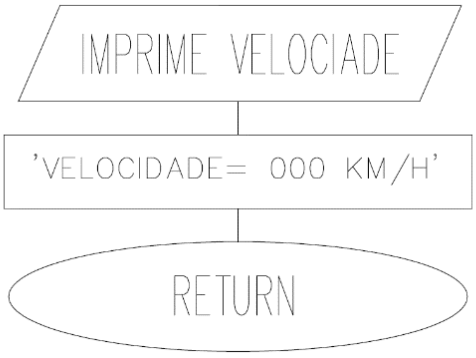


Figura 26 – Amostragem dos dados recebidos

O envio dos comandos de inicialização e configuração do modulo EVK2 do pic para o modem (AT+CGDCONT=1,”IP”,INTERNETM2M.AIR.COM), (AT#SGACT=1,1), (AT$GPSACP) e (AT#SD=1,0,10510, 177.102.220.XXX) segue o padrão de envio de men-

sagem conforme figura abaixo, bem como a inicialização do modulo OBD2 mudando apenas o *stream*.

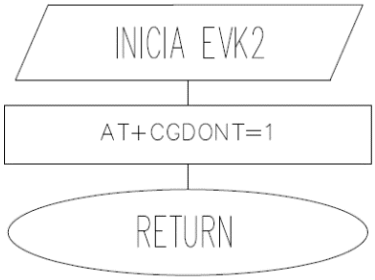


Figura 27 - Inicialização do modem EVK2

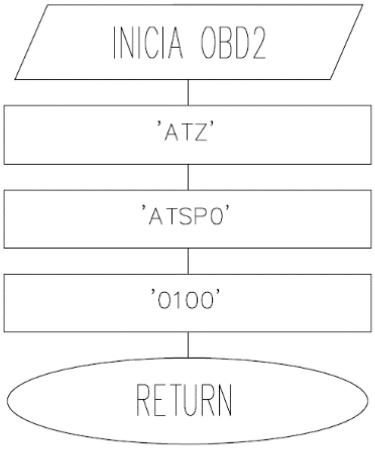


Figura 28 - Inicialização do scanner OBDII

A posição global utilizada no projeto é lida a partir kit de desenvolvimento através da função representada abaixo, logo após o envio dos dados (AT$GPSACP) requisitando o valor dos dados do GPS o modem retorna, por exemplo: ($GPSACP:080220.479, 4542.82691N,01344.26820E,259.07,3,2.1,0.1,0.0,0.0,270705,09), esse dado fica gravado no buffer de entrada e caso o programa não encontre o caractere “ : ” representa que o ocorreu um erro e um led acenderá, para leitura do dado o programa conta os caracteres a partir do “:” encontrado conforme Figura 30 para o horário e figura 31 para latitude e longitude.

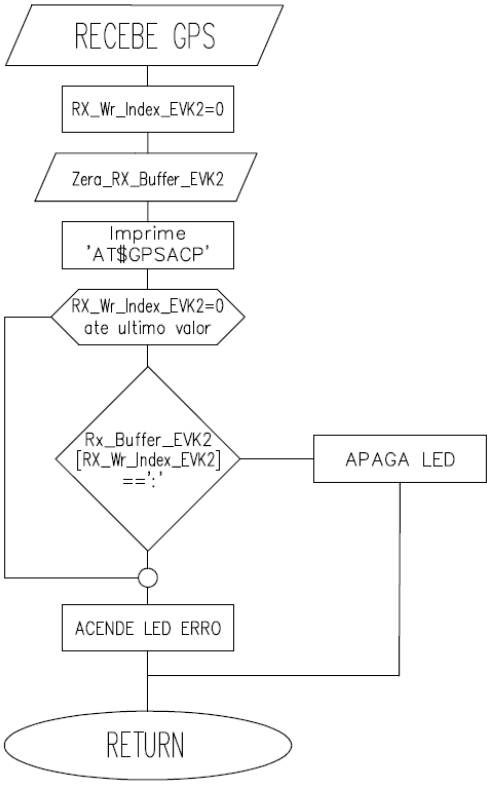


Figura 29 - Função, recebe dados do GPS

Figura 30 - Imprime o horário mundial

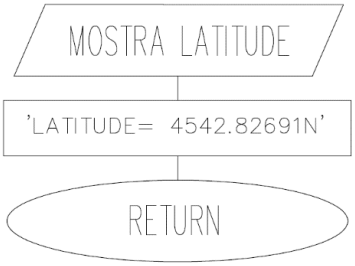


Figura 31 - Imprime latitude e longitude

O software primeiramente inicia os periféricos (*Scan* OBD2 e Modem EVK2), após a confirmação do registro o programa entra no loop infinito onde o programa requisita as informações de velocidade, rotação, ignição, pressão e GPS, essas informações são processadas

e transformadas em texto contendo as informações em tempo real do veículo, após isso o programa requisita a abertura de um socket PDP para o modem EVK2, com o socket aberto é enviado às informações do veículo, terminado esse processo o socket é fechado e as informações podem ser vistas no servidor web de internet e o programa recomeça seu loop.

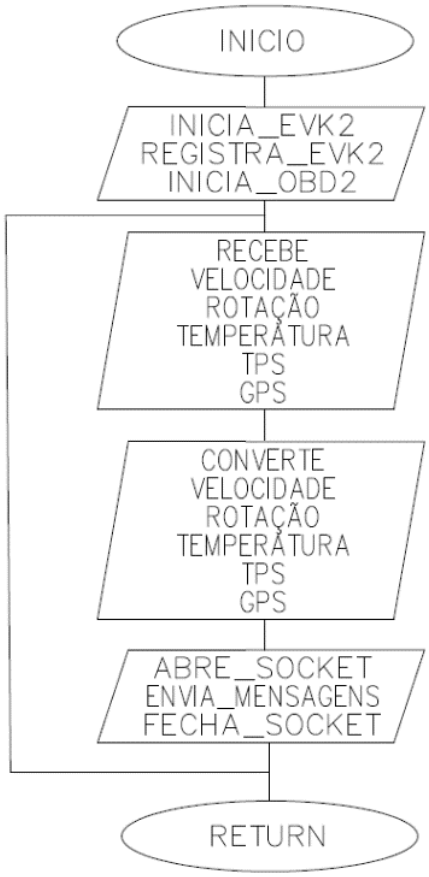


Figura 32 - Fluxograma do software

3.6. Servidor do projeto

Para criação do servidor do projeto foi utilizado o software Eclipse (O Eclipse é uma plataforma de desenvolvimento de software livre extensível, baseada em Java. *http://www.ibm.com/developerworks/br/library/os-eclipse-platform/* – 25/05/2015) utilizando linguagem JAVA (linguagem de programação orientada ao objeto - www.java.com – 25/05/2015) e para operação e desenvolvimento deste servidor foi utilizado o Glassfish 4.1, com isso é possível enviar sockets do kit de desenvolvimento EVK2 ao servidor e ser visualizado em um web browser, também conhecidos como páginas na web.

As figuras representam o tratamento dos dados em linguagem JAVA utilizados no Eclipse, o grupo se limitou apenas a utilizar o programa e não aprofundar o estudo nele, pois o foco de estudo da faculdade é a parte automotiva.

*ChatServerEndpoint.java

```java
package com.za.tutorial.websocket;
import java.io.IOException;

@ServerEndpoint(value = "/chatServerEndpoint" , encoders = { ChatMessageEncoder.class }, decoders = { ChatMessageDecoder.class})
public class ChatServerEndpoint {
    static Set<Session> chatroomUsers = Collections.synchronizedSet(new HashSet<Session>());
    @OnOpen
    public void handleOpen(Session userSession) {
        chatroomUsers.add(userSession);
    }
    @OnMessage
    public void handleMessage(ChatMessage incomingChatMessage, Session userSession) throws IOException, EncodeException {
        String username = (String) userSession.getUserProperties().get("username");
        ChatMessage outgoingChatMessage = new ChatMessage();
        if (username==null) {
            userSession.getUserProperties().put("username" , incomingChatMessage.getMessage());
            outgoingChatMessage.setName("System");
            outgoingChatMessage.setMessage("you are now connected as " + incomingChatMessage.getMessage());
            userSession.getBasicRemote().sendObject(outgoingChatMessage);
        } else {
            outgoingChatMessage.setName(username);
            outgoingChatMessage.setMessage(incomingChatMessage.getMessage());
            Iterator<Session> iterator = chatroomUsers.iterator();
            while (iterator.hasNext()) iterator.next().getBasicRemote().sendObject(outgoingChatMessage);
        }
    }

    @OnClose
    public void handleClose(Session userSession) {
        chatroomUsers.remove(userSession);
    }
}
```

Figura 33 - ChatServerEndPoint.java

*ChatMessageEncoder.java

```java
package com.za.tutorial.websocket;

import javax.json.Json;

public class ChatMessageEncoder implements Encoder.Text<ChatMessage> {

    @Override
    public void destroy() {
    }
    public void init(EndpointConfig arg0) {
    }
    public String encode(ChatMessage message) throws EncodeException {
        return Json.createObjectBuilder().add("name", message.getName())
                                         .add("message", message.getMessage())
                                         .build().toString();
    }
}
```

Figura 34 - ChatMessageEncoder.java

ChatMessage.java

```java
package com.za.tutorial.websocket;

public class ChatMessage {
    private String name;
    private String message;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }


}
```

Figura 35 - ChatMessage.java

*ChatMessageDecoder.java

```java
package com.za.tutorial.websocket;

import java.io.StringReader;
public class ChatMessageDecoder implements Decoder.Text<ChatMessage> {
    public void destroy() {}
    public void init(EndpointConfig arg0) {}
    public ChatMessage decode(String message) throws DecodeException {
        ChatMessage chatMessage = new ChatMessage();
        chatMessage.setMessage((Json.createReader(new StringReader(message)).readObject()).getString("message"));
        return chatMessage;
    }
    public boolean willDecode(String message) {
        boolean flag = true;
        try {Json.createReader(new StringReader(message)).readObject();}
        catch (Exception e) {flag = false;}
        return flag;
    }

}
```

Figura 36 - ChatMessageDecoder.java

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>WebSocket Tutorial 03</title>
    </head>
    <body>
        <textarea id="messages" readonly="readonly" rows="10" cols="60"></textarea><br/>
        <script type="text/javascript">
            var webSocket = new WebSocket('ws://localhost:8080/WebSocketPrj03/chatServerEndpoint');
            webSocket.onmessage = function processMessage(chatMessage) {
                var json = JSON.parse(chatMessage.data);
                document.getElementById('messages').value += json.name +': '+json.message + '\n';
            }
            function send() {
                var message = document.getElementById('message');
                webSocket.send(JSON.stringify({'message' : message.value}));
                message.value = "";
            }
            window.onbeforeunload = function() {
                websocket.onclose = function() {};
                websocket.close()
            };
        </script>
    </body>
</html>
```

Figura 37 - Formatando o estilo da pagina

A figura acima é a parte da aplicação em JAVA no eclipse que formata o estilo da página utilizada e as informações a ser exibida. Para demonstração foi executado a abertura de uma caixa de texto onde serão exibidos os dados via socket recebidos pelo veículo e enviados via GSM pelo kit de desenvolvimento EVK2 conforme figura abaixo.

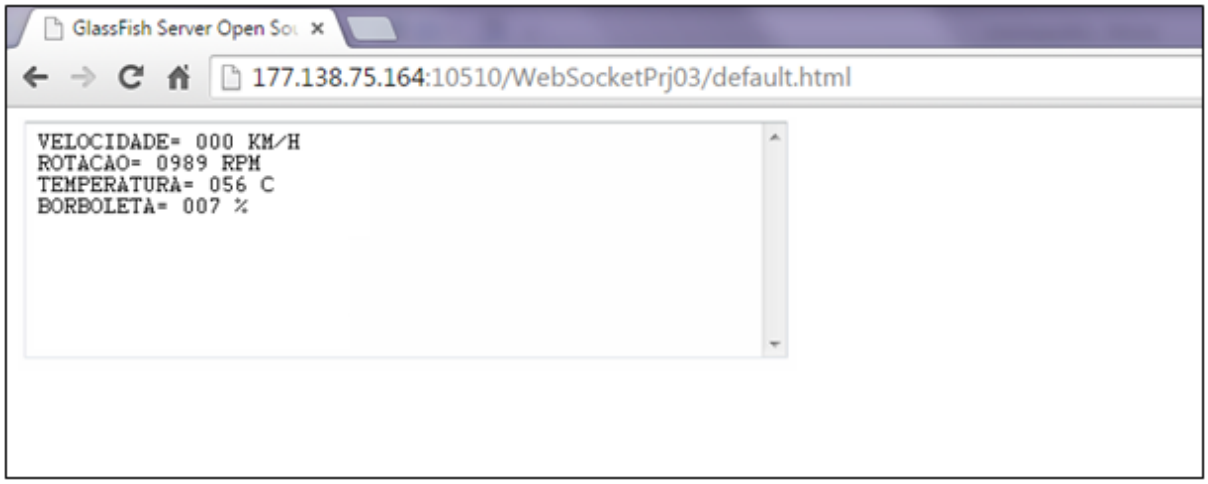


Figura 38 - Simulação do web browser

4. Analise dos resultados

O projeto consiste em basicamente três etapas pré-definidas simulação, tratativa e telemetria de dados vindos de uma ECU de um veículo. Analisando os resultados obtidos podemos concluir que em alguns pontos obtivemos sucesso e em outros encontramos algumas dificuldades devido à falta de conhecimento técnico sobre o assunto, um exemplo disso foi a tentativa de criar um servidor de internet para receber as informações, mas para realizar é necessário um grande conhecimento em JAVA e o foco do nosso curso não é este, com isso dedicamos a maior parte do tempo em realizar as comunicações com o veículo. Abaixo explicaremos mais detalhadamente cada etapa do projeto e tentativa de solução dos problemas encontrados.

4.1. Simulação

O projeto teve como objetivo simular cinco dados provenientes da ECU do veículo: velocidade, rotação, temperatura do liquido de arrefecimento, posição da válvula borboleta (TPS) do veículo.

A velocidade do veículo funcionou conforme o esperado, através da simulação da frequência utilizando o CI 555, conseguimos obter a leitura dos dados via OBD. O sinal será proporcional à velocidade do veiculo, maior frequência, maior velocidade.

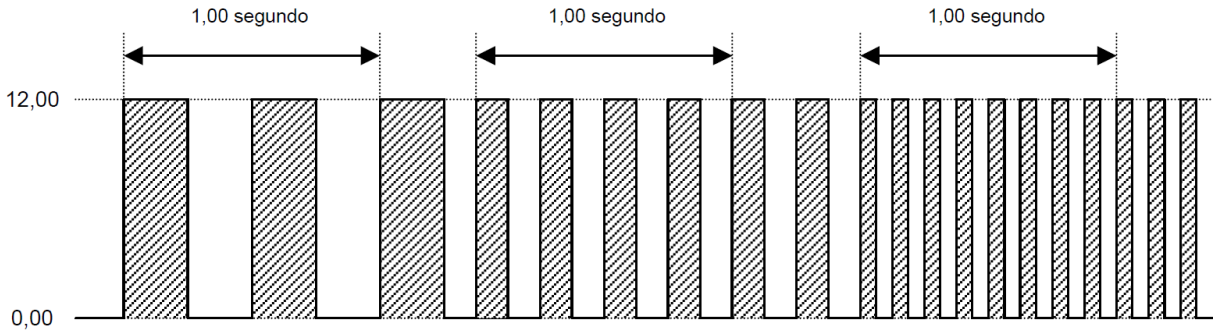


Figura 39-Velocidade em função da Frequência – Fonte: Sistema EEC-V Flavio Xavier

A simulação da rotação do veículo foi um dos pontos que apresentaram problemas, pois inicialmente projetamos nossa placa para enviar um sinal de onda quadrada de 0 a 5v, mas após alguns testes e falhas descobrimos que o sinal enviado pela roda fônica do veículo oscila de positivo a negativo, conforme figura 40, com isso nos limitamos as usar um simulador de roda fônica externo para concluir o projeto e demonstrar que a leitura do sinal funciona perfeitamente.

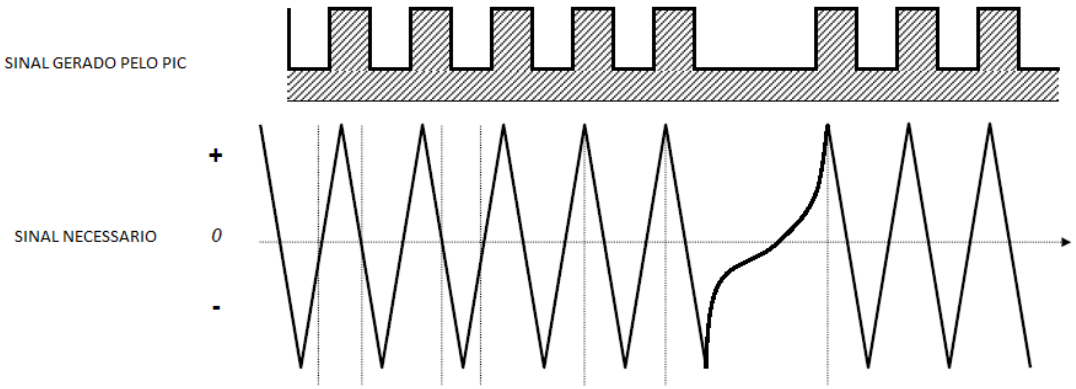


Figura 40 - Sinal Rotação – Fonte: Sistema EEC-V Flavio Xavier

O sensor da temperatura do liquido de arrefecimento do veículo é realizada por um PTC, que no projeto foi substituído por um potenciômetro de 100kΩ e com a sua variação conseguimos simular a mudança da temperatura conforme tabela abaixo.

Temperatura (°C)	Resistência elétrica (KΩ)	Volts DC
-10	78,2	4,40
0	65,9	4,00
10	56,0	3,45
20	36,0	3,00
30	24,0	2,60
40	16,2	2,10
50	11,1	1,70
60	7,50	1,30
70	5,35	1,00
80	4,00	0,80
90	2,90	0,60
100	2,15	0,45
110	1,60	0,35
120	1,25	0,30
130	1,00	0,22
191	0,00	0,00

Tabela 22 - Temperatura em função da resistência –– Fonte: Sistema EEC-V Flavio Xavier

A simulação do sensor de posição da válvula borboleta funcionou conforme esperado, com a variação do potenciômetro a tensão lida pela ECU varia de 0 a 5v, assim conseguimos variar o valor em porcentagem lido pelo OBD. Observando o gráfico da figura abaixo pode-mos visualizar o valor de tensão lida e a relação em porcentagem que representa a abertura da válvula borboleta do veículo.

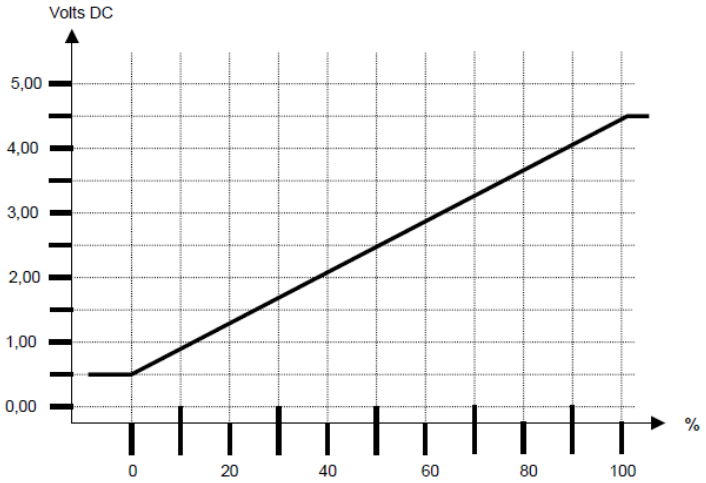


Figura 41 - Posição da válvula borboleta em função da tensão - – Fonte: Sistema EEC-V Flavio Xavier

4.2. Leitura e tratativa

A leitura dos dados vindas da ECU realizadas pelo ODBII funcionou conforme o esperado, os dados são obtidos via comunicação serial pelo microcontrolador da placa interface e são tratados e convertidos para visualização e envio. Conforme figura abaixo podemos verificar um teste realizado, onde mostra o valor da velocidade de 65 Km/h, uma rotação de 4163 RPM, 65°C a temperatura do liquido de arrefecimento, abertura da válvula borboleta de 45%.

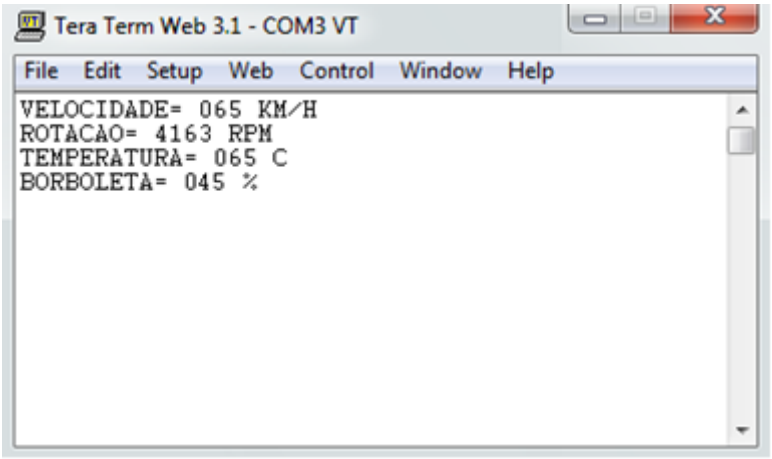


Figura 42 - Teste de leitura dos dados ECU - OBD2 via PC

A leitura dos dados de posicionamento global feita pelo modem EVK2 e realizada pelo microcontrolador do projeto não funcionou conforme o esperado, o PIC escolhido para o projeto possui apenas uma porta serial via hardware, mas é possível emular portas seriais via software, o problema encontrado pelo grupo foi na recepção dos dados via interrupção externa através da porta serial emulada, os dados recebidos via hardware funcionam perfeitamente enquanto os dados via software não são identificados de forma correta pelo microcontrolador,

tentamos executar essa leitura diretamente no programa excluindo a leitura via interrupção externa, o problema encontrado nesta solução foi o travamento do programa, pois seu ocorresse qualquer anomalia no programa ele ficaria travado. Uma solução proposta pelo grupo foi a utilização de um microcontrolador com duas portas seriais via hardware, com isso o programa se torna muito mais confiável. O envio dos dados via porta serial emulada do microcontrolador para o modem funcionou corretamente e assim foi possível enviar os dados via GSM.

4.3. Telemetria

Com os dados obtidos e armazenados no microcontrolador a última parte do projeto é o envio destes dados para a internet, executamos a criação do servidor utilizando o software eclipse e o servidor *Glassfish*, seguindo as instruções no site "www.zaneacademy.com" onde eles ensinam a criar seu servidor. Conseguimos criar e conectar, o problema foi na troca de dados que não aconteceu e não conseguimos localizar o defeito devido a complexidade da programação em JAVA. Como o foco do curso não é programação em JAVA decidimos deixar esta parte como propostas futuras e para realizar a telemetria dos dados usamos a comunicação via SMS que é possível através do modem EVK2, utilizado pelo grupo. Com isso realizamos a leitura dos dados e conseguimos ler estes a partir do celular escolhido.

Para envio dos dados via SMS temos que configurar o modem para modo texto (AT+CMGF=1) e para enviarmos a mensagem precisamos digitar o código (AT+CMGS=”+n°Celular”) assim será aberto um túnel de comunicação. Enviamos todas as mensagens e finalizamos com o caractere 0x1A (CTRL+Z).

Figura 43 - Telemetria dos dados via SMS

5. Conclusão

O principal objetivo deste projeto foi o desenvolvimento de uma interface que fizesse a leitura dos valores dos sensores de um veículo utilizando o padrão de comunicação OBD II e que esses valores lidos pudessem ser transmitidos sem a utilização de fios através de algum formato de transmissão que também tivesse um formato padrão e que fosse largamente utilizado, por isso a escolha da transmissão via GSM. Podemos concluir que obtivemos êxito, pois foi possível coletar, tratar, exibir e enviar estes dados de forma fácil e coerente.

No entanto, uma das maiores dificuldades enfrentadas no nosso projeto foi a de efetuar uma comunicação segura e estável utilizando a comunicação RS 232, um protocolo antigo e que é facilmente corrompível sob a influência de alguma interferência eletromagnética externa. Também enfrentamos dificuldades na transmissão de dados utilizando transmissão via GPRS, devido ao baixo conhecimento na aplicação desta tecnologia não foi possível exibir as informações enviadas de uma maneira global e por isso optamos pela transmissão via SMS, onde também obtivemos sucesso.

.

5.1. Projetos futuros

Uma das dificuldades encontradas, por falta de conhecimento técnico foi a criação de um site para exibir os dados tratados, portanto temos como proposta futura a exibição destes dados via um servidor remoto apresentando o conteúdo em uma *webpage* onde será possível a visualização em um site podendo ser exibido em qualquer dispositivo com um navegador web.

Outra proposta para um trabalho futuro é a simulação dos demais sensores presentes na EEC – V, tais como o sensor MAF, rotação e o sensor de oxigênio.

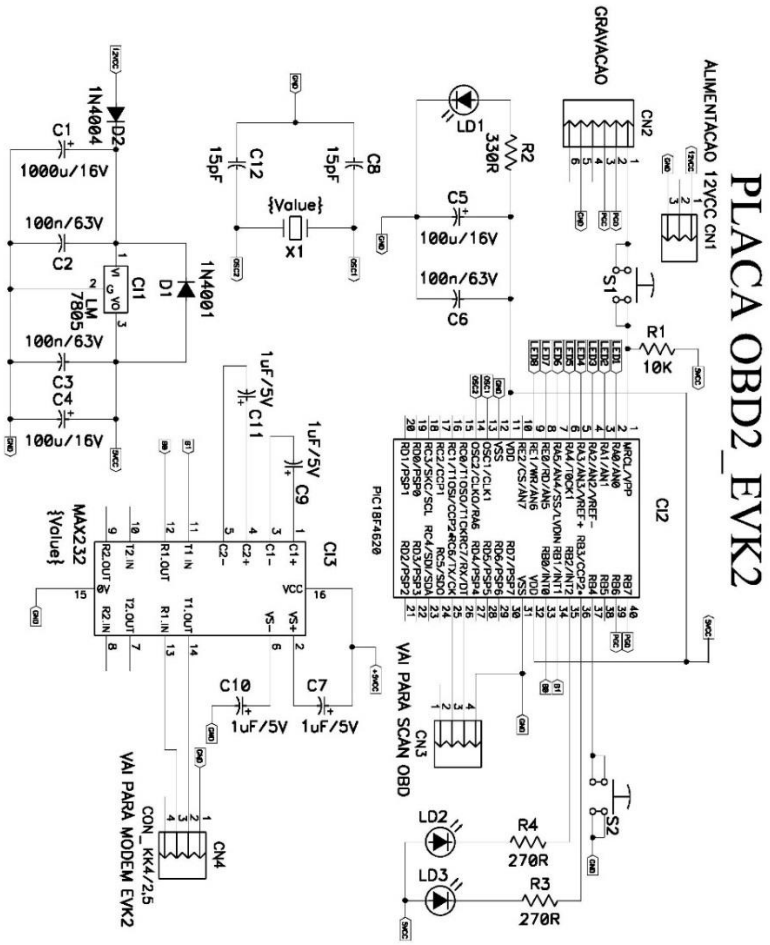
6. Referências

- Estudo das diferenças dos requerimentos das principais legislações de onboard diagnostics para padronização de testes de desenvolvimento e validação de transmissão automática de automóveis– Eduardo Bastos – Centro universitário do instituto Mauá de tecnologia;
- SISTEMA MULTI-PROTOCOLO PARA TRANSFERÊNCIA DE DADOS - RAQUEL LAMPAÇA VIEIRA RADOMAN - INSTITUTO MILITAR DE ENGENHARIA;
- Sistema de Monitoramento e Rastreamento por GPS e GSM – Albano Rocha da Costa – Universidade Federal do Rio Grande do Norte;
- Telemetria Automotiva via Internet Móvel - Alcides Carlos de Moraes Neto - Universidade de Brasília;
- EASY EMBEDDED CONNECTION - DANILO NEGOZZECK - PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ;
- SISTEMA MULTI-PROTOCOLO PARA TRANSFERÊNCIA DE DADOS - RAQUEL LAMPAÇA VIEIRA RADOMAN - INSTITUTO MILITAR DE ENGENHARIA
- FERRAMENTA DE DIAGNÓSTICO AUTOMOTIVO OBD II - LUIZ RICARDO TRAJANO DA SILVA - FATEC SANTO ANDRÉ
- K-line Communication Description – Volkswagen – Group of America;
- SISTEMA DE DIAGNÓSTICO PARA VEÍCULOS QUE UTILIZAM OS PROTOCOLOS ISO 9141 E ISO 14230 ATRAVÉS DE UMA PLATAFORMA EM LabView - ADEMAR DULTRA CERQUEIRA - FATEC SANTO ANDRÉ;
- INITIALIZING COMMUNICATION TO VEHICLE OBDII SYSTEM - Peter Dzhelekarski, Dimiter Alexiev - Technical University of Sofia – Bulgaria;
- SISTEMA DE DIAGNOSE VEICULAR ON-BOARD EM UMA PLATAFORMA DIDÁTICA DE GERENCIAMENTO ELETRÔNICO - BRUNO SILVA PEREIRA - FATEC SANTO ANDRÉ;
- Creating A Wireless OBDII Scanner - John Keenan - WORCESTER POLYTECHNIC INSTITUTE
- Monitoramento de Dados via Internet baseado em Telefonia Celular - Getúlio Teruo Tateoki - UNIVERSIDADE ESTADUAL PAULISTA (UNESP)
- Sistema para Diagnostico Automático de Falhas em Veículos Automotores OBD-2 - Valdeci Pereira Belo – Universidade Federal de Minas Gerais;

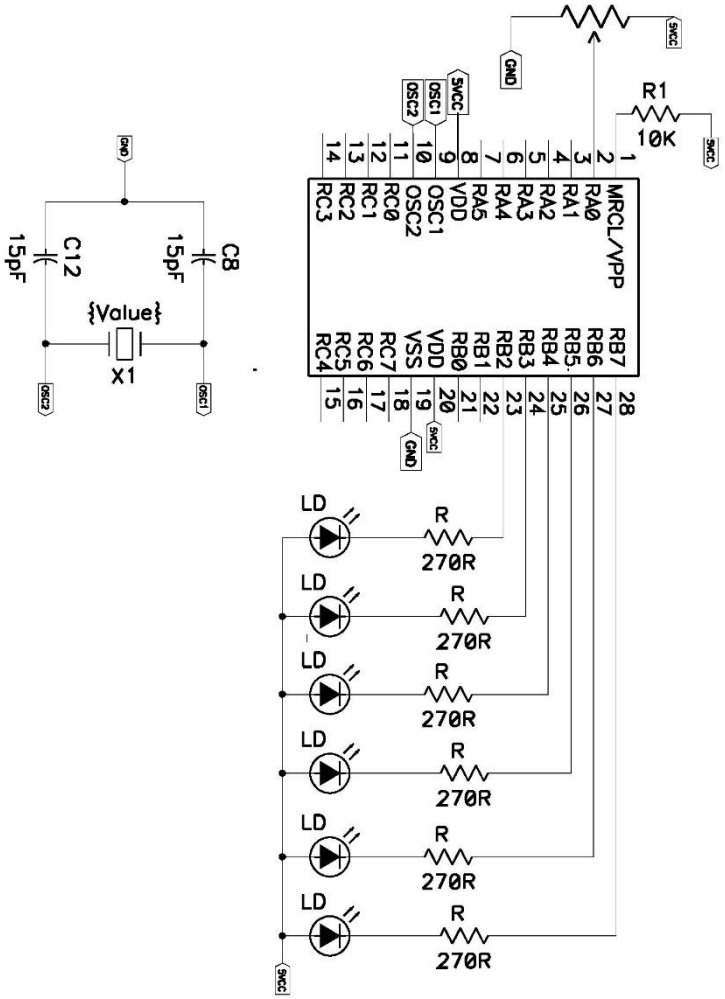
- AT Commands Reference Guide – Telit – r21;
- Telit Evaluation Kit EVK2 Datasheet;
- EVK2 Drivers Installation Guide – Telit;
- Telit EVK2 User Guide – Telit;
- HE910/UE910/UL865 At Commands;
- CCS C Compiler Manual - PCB / PCM / PCH
- O Sistema OBD (On-Board Diagnosis) - António Sérgio Leite Machado - Instituto Superior de Engenharia do Porto
- In-Vehicle Networking - Lecture 4 Introduction to SAE J1850 - BAE 5030 – 003 - Instructor: Marvin Stone - Biosystems and Agricultural Engineering - Oklahoma State
- http://www.forumhd.com.br/index.php?topic=13120.0;wap2 – 22/01/2015;
- http://www.devmedia.com.br/introducao-a-websockets-com-tomcat-7/26932 - 13/02/15;
- http://g8gt.blogspot.com.br/2012/04/adventures-in-sw-cangmlan-land-part-ii.html - 12/01/15;
- http://aniki.daionet.gr.jp/~iwata/blog/ - 24/04/15
- http://konchatech.blogspot.com.br/2014/02/obd-ii-arduino-car-information-display.html - 16/01/15
- http://www.zaneacademy.com/ - 04/2015;
- http://juliobattisti.com.br/artigos/windows/tcpip_p1.asp - 17/12/14;
- http://www.elmelectronics.com/obdic.html - 15/01/15
- http://www.obdexperts.co.uk/faq.html - 18/01/15
- https://blog.idrsolutions.com/2013/12/websockets-an-introduction/ - 30/04/15
- https://netbeans.org/kb/docs/javaee/maven-websocketapi.html -25/01/15
- https://blog.openshift.com/how-to-build-java-websocket-applications-using-the-jsr-356-api/ - 17-03/15
- Datasheet ELM327 – OBD to RS232 Interpreter;
- Datasheet Pic18F2220;
- Datasheet Pic18F4620;
- Datasheet 555;
- Datasheet Max232 – Texas Instruments;

7. Apêndice

APÊNDICE A – Circuito placa OBD 2 EVK2

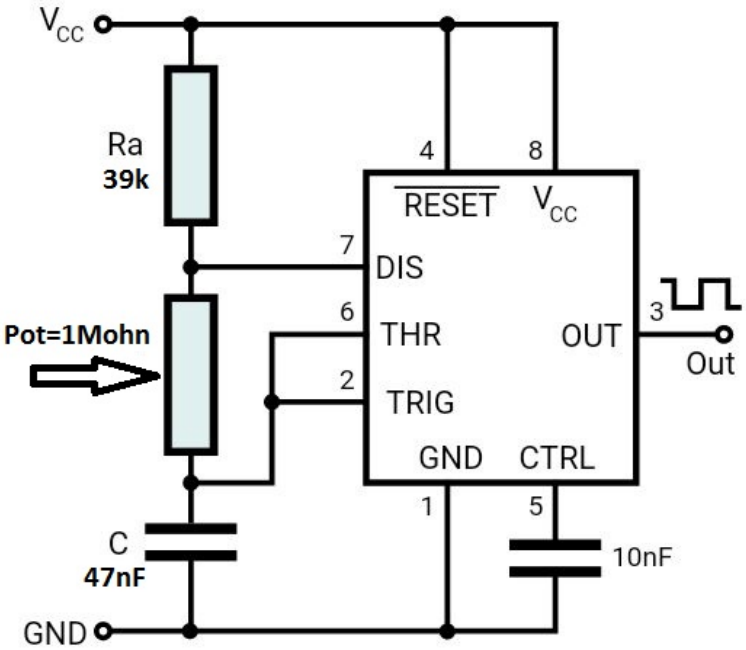


APÊNDICE B – Esquema elétrico Placa Rotação



APÊNDICE C – Esquema elétrico placa do sensor de velocidade

CIRCUITO VELOCIDADE



$$Freq = \frac{1.44}{(Ra + 2 * \mathbf{Pot}) * C}$$

APÊNDICE D – Código do programa

```
#include <18F4620.h>
#fuses HS,NOWDT
#use delay(clock=20000000)

#use rs232(baud=38400,parity=N,xmit=PIN_C6,rcv=PIN_C7,stream=OBD2,bits=8)
#use rs232(baud=9600,parity=N,xmit=PIN_B1,rcv=PIN_B0,stream=EVK2,bits=8)

//Area dos Defines
#define LED1OFF output_high(pin_B3);
#define LED1ON output_low(pin_B3);
#define LED2OFF output_high(pin_B2);
#define LED2ON output_low(pin_B2);
#define LED_ERRO_VELOCIDADE_OFF output_high(pin_A0)
#define LED_ERRO_VELOCIDADE_ON output_low(pin_A0)
#define LED_ERRO_ROTACAO_OFF output_high(pin_A1)
#define LED_ERRO_ROTACAO_ON output_low(pin_A1)
#define LED_ERRO_TEMPERATURA_OFF output_high(pin_A2)
#define LED_ERRO_TEMPERATURA_ON output_low(pin_A2)
#define LED_ERRO_BORBOLETA_OFF output_high(pin_A3)
#define LED_ERRO_BORBOLETA_ON output_low(pin_A3)
#define LED_ERRO_SMS_OFF output_high(pin_E1)
#define LED_ERRO_SMS_ON output_low(pin_E1)
#define LED_ERRO_GPS_OFF output_high(pin_A5)
#define LED_ERRO_GPS_ON output_low(pin_A5)
#define LED_ERRO_E0_OFF output_high(pin_E0)
#define LED_ERRO_E0_ON output_low(pin_E0)
#define LED_ERRO_E1_OFF output_high(pin_A4)
#define LED_ERRO_E1_ON output_low(pin_A4)

#define  RX_BUFFER_SIZE        30
char Rx_Buffer[RX_BUFFER_SIZE+1];
char RX_Wr_Index = 0;
char RX_Rd_Index = 0;
char RX_Counter = 0;
```

```
#define  RX_BUFFER_SIZE_EVK2        50
char Rx_Buffer_EVK2[RX_BUFFER_SIZE_EVK2+1];
char RX_Wr_Index_EVK2 = 0;
char RX_Counter_EVK2 = 0;

//Declaraçao da variaveis utilizadas no programa
char velocidade_nova[2];
unsigned long int velocidade_total;
char velocidade_imprime0;
char velocidade_imprime1;
char velocidade_imprime[3];
char rotacao_nova[4];
unsigned long int rotacao_total;
char rotacao_imprime0;
char rotacao_imprime1;
char rotacao_imprime2;
char rotacao_imprime[4];
char temperatura_nova[2];
unsigned long int temperatura_total;
char temperatura_imprime0;
char temperatura_imprime1;
char temperatura_imprime[3];
char borboleta_nova[2];
int32 borboleta_total;
char borboleta_imprime[3];
char gps[50];


// interrupcao externa via harware

#int_rda
void serial_rx_isr()
{
   Rx_Buffer[RX_Wr_Index] = fgetc(OBD2);
   if(++RX_Wr_Index > RX_BUFFER_SIZE) RX_Wr_Index = 0;
```

```
   if(++RX_Counter > RX_BUFFER_SIZE)
   {
     RX_Counter = RX_BUFFER_SIZE;
   }
}

// interrupcao externa via software

#int_ext
void serial_rx_ext()
{
   Rx_Buffer_EVK2[RX_Wr_Index_EVK2] = fgetc(EVK2);
   if(++RX_Wr_Index_EVK2 > RX_BUFFER_SIZE_EVK2) RX_Wr_Index_EVK2 = 0;
   if(++RX_Counter_EVK2 > RX_BUFFER_SIZE_EVK2)
   {
     RX_Counter = RX_BUFFER_SIZE;
   }
}

void zera_Rx_Buffer()
{
  for(RX_Wr_Index=0;RX_Wr_Index <= RX_BUFFER_SIZE;RX_Wr_Index++)
  {
    Rx_Buffer[RX_Wr_Index]=0;
  }
}

void zera_Rx_Buffer_EVK2()
{
  for(RX_Wr_Index2=0;RX_Wr_Index2 <= RX_BUFFER_SIZE2;RX_Wr_Index2++)
  {
    Rx_Buffer2[RX_Wr_Index2]=0;
  }
}

char String_to_number(unsigned int c)
```

```
{
  switch (c)
       {
            case '0': c=0; return c; break;
            case '1': c=1; return c; break;
            case '2': c=2; return c; break;
            case '3': c=3; return c; break;
            case '4': c=4; return c; break;
            case '5': c=5; return c; break;
            case '6': c=6; return c; break;
            case '7': c=7; return c; break;
            case '8': c=8; return c; break;
            case '9': c=9; return c; break;
            case 'A': c=10; return c; break;
            case 'B': c=11; return c; break;
            case 'C': c=12; return c; break;
            case 'D': c=13; return c; break;
            case 'E': c=14; return c; break;
            case 'F': c=15; return c; break;
       }
}

char number_to_string(unsigned int b)
{
  switch (b)
  {
    case 0: b='0'; return b; break;
    case 1: b='1'; return b; break;
    case 2: b='2'; return b; break;
    case 3: b='3'; return b; break;
    case 4: b='4'; return b; break;
    case 5: b='5'; return b; break;
    case 6: b='6'; return b; break;
    case 7: b='7'; return b; break;
    case 8: b='8'; return b; break;
    case 9: b='9'; return b; break;
```

```
  }
}

char imprimi_porcento(unsigned int total)
{
  switch (total)
  {
    case 0: d='0'; e='0'; f='0'; return d,e,f;
    case 1: d='0'; e='0'; f='2'; return d,e,f;
    case 2: d='0'; e='0'; f='4'; return d,e,f;
    case 3: d='0'; e='0'; f='6'; return d,e,f;
    case 4: d='0'; e='0'; f='8'; return d,e,f;
    case 5: d='0'; e='1'; f='0'; return d,e,f;
    case 6: d='0'; e='1'; f='2'; return d,e,f;
    case 7: d='0'; e='1'; f='4'; return d,e,f;
    case 8: d='0'; e='1'; f='6'; return d,e,f;
    case 9: d='0'; e='1'; f='8'; return d,e,f;
    case 10: d='0'; e='2'; f='0'; return d,e,f;
    case 11: d='0'; e='2'; f='2'; return d,e,f;
    case 12: d='0'; e='2'; f='4'; return d,e,f;
    case 13: d='0'; e='2'; f='6'; return d,e,f;
    case 14: d='0'; e='2'; f='8'; return d,e,f;
    case 15: d='0'; e='3'; f='0'; return d,e,f;
    case 16: d='0'; e='3'; f='2'; return d,e,f;
    case 17: d='0'; e='3'; f='4'; return d,e,f;
    case 18: d='0'; e='3'; f='6'; return d,e,f;
    case 19: d='0'; e='3'; f='8'; return d,e,f;
    case 20: d='0'; e='4'; f='0'; return d,e,f;
    case 21: d='0'; e='4'; f='2'; return d,e,f;
    case 22: d='0'; e='4'; f='4'; return d,e,f;
    case 23: d='0'; e='4'; f='6'; return d,e,f;
    case 24: d='0'; e='4'; f='8'; return d,e,f;
    case 25: d='0'; e='5'; f='0'; return d,e,f;
    case 26: d='0'; e='5'; f='2'; return d,e,f;
    case 27: d='0'; e='5'; f='4'; return d,e,f;
    case 28: d='0'; e='5'; f='6'; return d,e,f;
```

```
    case 29: d='0'; e='5'; f='8'; return d,e,f;
    case 30: d='0'; e='6'; f='0'; return d,e,f;
    case 31: d='0'; e='6'; f='2'; return d,e,f;
    case 32: d='0'; e='6'; f='4'; return d,e,f;
    case 33: d='0'; e='6'; f='6'; return d,e,f;
    case 34: d='0'; e='6'; f='8'; return d,e,f;
    case 35: d='0'; e='7'; f='0'; return d,e,f;
    case 36: d='0'; e='7'; f='2'; return d,e,f;
    case 37: d='0'; e='7'; f='4'; return d,e,f;
    case 38: d='0'; e='7'; f='6'; return d,e,f;
    case 39: d='0'; e='7'; f='8'; return d,e,f;
    case 40: d='0'; e='8'; f='0'; return d,e,f;
    case 41: d='0'; e='8'; f='2'; return d,e,f;
    case 42: d='0'; e='8'; f='4'; return d,e,f;
    case 43: d='0'; e='8'; f='6'; return d,e,f;
    case 44: d='0'; e='8'; f='8'; return d,e,f;
    case 45: d='0'; e='9'; f='0'; return d,e,f;
    case 46: d='0'; e='9'; f='2'; return d,e,f;
    case 47: d='0'; e='9'; f='4'; return d,e,f;
    case 48: d='0'; e='9'; f='6'; return d,e,f;
    case 49: d='0'; e='9'; f='8'; return d,e,f;
    case 50: d='1'; e='0'; f='0'; return d,e,f;
    case 51: d='1'; e='0'; f='0'; return d,e,f;
  }
}

void calcula_velocidade()
{
  velocidade_nova[0]=String_to_number(rx_buffer[RX_Wr_Index+8]);
  velocidade_nova[1]=String_to_number(rx_buffer[RX_Wr_Index+9]);
  velocidade_total=((velocidade_nova[0]*16)+(velocidade_nova[1]));
  velocidade_imprime0=velocidade_total%10;
  velocidade_total=velocidade_total/10;
  velocidade_imprime1=velocidade_total%10;
  velocidade_total=velocidade_total/10;
  velocidade_imprime[0]=number_to_string(velocidade_imprime0);
```

```
  velocidade_imprime[1]=number_to_string(velocidade_imprime1);
  velocidade_imprime[2]=number_to_string(velocidade_total);
}

void recebe_velocidade()
{
  zera_Rx_Buffer();
  RX_Wr_Index=0;
  bputc(0x30);//"0"
  bputc(0x31);//"1"
  bputc(0x30);//"0"
  bputc(0x44);//"D"
  bputc(0x0D);//ENTER
  bputc(0x0A);
  delay_ms(1000);
  for(RX_Wr_Index=0;RX_Wr_Index <= RX_BUFFER_SIZE;RX_Wr_Index++)
  {
    if((Rx_Buffer[RX_Wr_Index]=='4') & (Rx_Buffer[RX_Wr_Index+6]=='1'))
    {
      calcula_velocidade();
      LED_ERRO_VELOCIDADE_OFF;
      return;
    }
  }
  LED_ERRO_VELOCIDADE_ON;
}

void mostra_velocidade() //VELOCIDADE= 000 KM/H
{
  fputc(0x56,EVK2);
  fputc(0x45,EVK2);
  fputc(0x4C,EVK2);
  fputc(0x4F,EVK2);
  fputc(0x43,EVK2);
  fputc(0x49,EVK2);
  fputc(0x44,EVK2);
```

```
  fputc(0x41,EVK2);
  fputc(0x44,EVK2);
  fputc(0x45,EVK2);
  fputc(0x3D,EVK2);
  ESPACOe fputc(0x20,EVK2);
  fputc(velocidade_imprime[2],EVK2);
  fputc(velocidade_imprime[1],EVK2);
  fputc(velocidade_imprime[0],EVK2);
  fputc(0x20,EVK2);
  fputc(0x4B,EVK2);
  fputc(0x4D,EVK2);
  fputc(0x2F,EVK2);
  fputc(0x48,EVK2);
  fputc(0x0D,EVK2);
  fputc(0x0A,EVK2);
}

void calcula_rotacao()//rotacao[8] - 41 0C 00
{
  rotacao_nova[0]=String_to_number(rx_buffer[RX_Wr_Index+8]);
  rotacao_nova[1]=String_to_number(rx_buffer[RX_Wr_Index+9]);
  rotacao_nova[2]=String_to_number(rx_buffer[RX_Wr_Index+11]);
  rotacao_nova[3]=String_to_number(rx_buffer[RX_Wr_Index+12]);
  rota-
cao_total=((rotacao_nova[0]*4096)+(rotacao_nova[1]*256)+(rotacao_nova[2]*16)+(rotacao_
nova[3]))/4;
  rotacao_imprime0=rotacao_total%10;
  rotacao_total=rotacao_total/10;
  rotacao_imprime1=rotacao_total%10;
  rotacao_total=rotacao_total/10;
  rotacao_imprime2=rotacao_total%10;
  rotacao_total=rotacao_total/10;
  rotacao_imprime[0]=number_to_string(rotacao_imprime0);
  rotacao_imprime[1]=number_to_string(rotacao_imprime1);
  rotacao_imprime[2]=number_to_string(rotacao_imprime2);
  rotacao_imprime[3]=number_to_string(rotacao_total);
```

```
}

void recebe_rotacao()
{
  zera_Rx_Buffer();
  RX_Wr_Index=0;
  bputc(0x30);//"0"
  bputc(0x31);//"1"
  bputc(0x30);//"0"
  bputc(0x43);//"C"
  bputc(0x0D);//ENTER
  delay_ms(1000);
  for(RX_Wr_Index=0;RX_Wr_Index <= RX_BUFFER_SIZE;RX_Wr_Index++)
  {
    if((Rx_Buffer[RX_Wr_Index]=='4') & (Rx_Buffer[RX_Wr_Index+6]=='1'))
    {
      calcula_rotacao();
      LED_ERRO_ROTACAO_OFF;
      return;
    }
    LED_ERRO_ROTACAO_ON;
  }
}

void mostra_rotacao()// Rotação 000 RPM
{
  fputc(0x52,EVK2);
  fputc(0x4F,EVK2);
  fputc(0x54,EVK2);
  fputc(0x41,EVK2);
  fputc(0x43,EVK2);
  fputc(0x41,EVK2);
  fputc(0x4F,EVK2);
  fputc(0x3D,EVK2);
  fputc(0x20,EVK2);
  fputc(rotacao_imprime[3],EVK2);
```

```
  fputc(rotacao_imprime[2],EVK2);
  fputc(rotacao_imprime[1],EVK2);
  fputc(rotacao_imprime[0],EVK2);
  fputc(0x20,EVK2);
  fputc(0x52,EVK2);
  fputc(0x50,EVK2);
  fputc(0x4D,EVK2);
  fputc(0x20,EVK2);
  fputc(0x0D,EVK2);
  fputc(0x0A,EVK2);
}

void calcula_temperatura()
{
  temperatura_nova[0]=String_to_number(rx_buffer[RX_Wr_Index+8]);
  temperatura_nova[1]=String_to_number(rx_buffer[RX_Wr_Index+9]);
  temperatura_total=((temperatura_nova[0]*16)+(temperatura_nova[1]))-40;
  temperatura_imprime0=temperatura_total%10;
  temperatura_total=temperatura_total/10;
  temperatura_imprime1=temperatura_total%10;
  temperatura_total=temperatura_total/10;
  temperatura_imprime[0]=number_to_string(temperatura_imprime0);
  temperatura_imprime[1]=number_to_string(temperatura_imprime1);
  temperatura_imprime[2]=number_to_string(temperatura_total);
}

void recebe_temperatura()
{
  zera_Rx_Buffer();
  RX_Wr_Index=0;
  bputc(0x30);//"0"
  bputc(0x31);//"1"
  bputc(0x30);//"0"
  bputc(0x35);//"5"
  bputc(0x0D);//ENTER
  delay_ms(1000);
```

```
  for(RX_Wr_Index=0;RX_Wr_Index <= RX_BUFFER_SIZE;RX_Wr_Index++)
  {
    if((Rx_Buffer[RX_Wr_Index]=='5') & (Rx_Buffer[RX_Wr_Index+6]=='5'))
    {
      calcula_temperatura();
      LED_ERRO_TEMPERATURA_OFF;
      return;
    }
    LED_ERRO_TEMPERATURA_ON;    // caso tenha falha acende led de erro
  }
}

void mostra_temperatura()// temperatura= 000 C
{
  fputc(0x54,EVK2);
  fputc(0x45,EVK2);
  fputc(0x4D,EVK2);
  fputc(0x50,EVK2);
  fputc(0x45,EVK2);
  fputc(0x52,EVK2);
  fputc(0x41,EVK2);
  fputc(0x54,EVK2);
  fputc(0x55,EVK2);
  fputc(0x52,EVK2);
  fputc(0x41,EVK2);
  fputc(0x3D,EVK2);
  fputc(0x20,EVK2);
  fputc(temperatura_imprime[2],EVK2);
  fputc(temperatura_imprime[1],EVK2);
  fputc(temperatura_imprime[0],EVK2);
  fputc(0x20,EVK2);
  fputc(0x43,EVK2);
  fputc(0x0D,EVK2);
  fputc(0x0A,EVK2);
}
```

```
void calcula_borboleta()
{
  borboleta_nova[0]=String_to_number(rx_buffer[RX_Wr_Index+6]);
  borboleta_nova[1]=String_to_number(rx_buffer[RX_Wr_Index+7]);
  borboleta_total=(borboleta_nova[0]*16)+(borboleta_nova[1]);
  borboleta_total=borboleta_total/5;
  imprimi_porcento(borboleta_total);
  borboleta_imprime[0]=d;
  borboleta_imprime[1]=e;
  borboleta_imprime[2]=f;
}

void recebe_borboleta()
{
  zera_Rx_Buffer();
  RX_Wr_Index=0;
  bputc(0x30);//"0"
  bputc(0x31);//"1"
  bputc(0x31);//"1"
  bputc(0x31);//"1"
  bputc(0x0D);//ENTER
  delay_ms(1000);
  for(RX_Wr_Index=0;RX_Wr_Index <= RX_BUFFER_SIZE;RX_Wr_Index++)
  {
    if((Rx_Buffer[RX_Wr_Index]=='4') & (Rx_Buffer[RX_Wr_Index+1]=='1'))
    {
      calcula_borboleta();
      LED_ERRO_BORBOLETA_OFF;
      return;
    }
    LED_ERRO_BORBOLETA_ON;
  }
}

void mostra_borboleta()// borboleta= 000 %
{
```

```
  fputc(0x42,EVK2);
  fputc(0x4F,EVK2);
  fputc(0x52,EVK2);
  fputc(0x42,EVK2);
  fputc(0x4F,EVK2);
  fputc(0x4C,EVK2);
  fputc(0x45,EVK2);
  fputc(0x54,EVK2);
  fputc(0x41,EVK2);
  fputc(0x3D,EVK2);
  fputc(0x20,EVK2);
  fputc(borboleta_imprime[0],EVK2);
  fputc(borboleta_imprime[1],EVK2);
  fputc(borboleta_imprime[2],EVK2);
  fputc(0x20,EVK2);
  fputc(0x25,EVK2);
  fputc(0x0D,EVK2);
  fputc(0x0A,EVK2);
}

void recebe_gps()
{
  zera_Rx_Buffer_EVK2();
  RX_Wr_Index_EVK2=0;
  fputc(0x41,EVK2);//A
  fputc(0x54,EVK2);//T
  fputc(0x24,EVK2);//$
  fputc(0x47,EVK2);//G
  fputc(0x50,EVK2);//P
  fputc(0x53,EVK2);//S
  fputc(0x41,EVK2);//A
  fputc(0x43,EVK2);//C
  fputc(0x50,EVK2);//P
  delay_ms(1000);
  for(RX_Wr_Index_EVK2=0;RX_Wr_Index_EVK2<=
RX_BUFFER_SIZE_EVK2;RX_Wr_Index++_EVK2)
```

```
  {
     if(Rx_Buffer_EVK2[RX_Wr_Index_EVK2]==':')
    {
      LED_ERRO_GPS_OFF;
      return;
     }
   LED_ERRO_GPS_ON;
  }
}


void mostra_horario( )
{
  fputc(0x48,EVK2);//H
  fputc(0x4F,EVK2);//O
  fputc(0x52,EVK2);//R
  fputc(0x41,EVK2);//A
  fputc(0x3D,EVK2);//=
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+1],EVK2);//HORA
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+1],EVK2);//HORA
  fputc(0x3A,EVK2);//:
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+1],EVK2);//MINUTO
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+1],EVK2);//MINUTO
  fputc(0x3A,EVK2);//:
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+1],EVK2);//SEGUNDO
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+1],EVK2);//SEGUNDO
}


void mostra_latitude()
{
  fputc(0x4C,EVK2);//L
  fputc(0x41,EVK2);//A
  fputc(0x54,EVK2);//T
  fputc(0x49,EVK2);//I
  fputc(0x54,EVK2);//T
```

```
  fputc(0x55,EVK2);//U
  fputc(0x44,EVK2);//D
  fputc(0x45,EVK2);//E
  fputc(0x3D,EVK2);//=
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+12],EVK2);//GRAUS
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+13],EVK2);//GRAUS
  fputc(0xF8,EVK2);//°
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+14],EVK2);//MINUTOS
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+15],EVK2);//MINUTOS
  fputc(0x6E,EVK2);//m
}


void mostra_lONGITUDE()
{
  fputc(0x4C,EVK2);//L
  fputc(0x4F,EVK2);//O
  fputc(0x4E,EVK2);//N
  fputc(0x47,EVK2);//G
  fputc(0x49,EVK2);//I
  fputc(0x54,EVK2);//T
  fputc(0x55,EVK2);//U
  fputc(0x44,EVK2);//D
  fputc(0x45,EVK2);//E
  fputc(0x3D,EVK2);//=
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+24],EVK2);//GRAUS
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+25],EVK2);//GRAUS
  fputc(0xF8,EVK2);//°
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+26],EVK2);//MINUTOS
  fputc(Rx_Buffer_EVK2[RX_Wr_Index_EVK2+27],EVK2);//MINUTOS
  fputc(0x6E,EVK2);//m
}

void inicia_OBD2()
{
  bputc(0x41);//A
```

```
  bputc(0x54);//T
  bputc(0x5A);//Z
  bputc(0x0D);//ENTER
  delay_ms(1000);
  bputc(0x41);//A
  bputc(0x54);//T
  bputc(0x53);//S
  bputc(0x50);//P
  bputc(0x30);//0
  bputc(0x0D);//ENTER
  delay_ms(5000);
  bputc(0x30);
  bputc(0x31);
  bputc(0x30);
  bputc(0x30);
  bputc(0x0D);
  delay_ms(1000);
  bputc(0x30);
  bputc(0x31);
  bputc(0x30);
  bputc(0x30);
  bputc(0x0D);
  delay_ms(1000);
}

VOID inicia_EVK2()
{
      zera_Rx_Buffer();
      fputc(0x41,EVK2); // "A"
      fputc(0x54,EVK2); // "T"
      fputc(0x2B,EVK2); // "+"
      fputc(0x43,EVK2); // "C"
      fputc(0x4D,EVK2); // "M"
      fputc(0x47,EVK2); // "G"
      fputc(0x46,EVK2); // "F"
      fputc(0x3D,EVK2); // "="
```

```
        fputc(0x31,EVK2); // "1"
        fputc(0x0A,EVK2); // "\n"
        fputc(0x0D,EVK2); // "\R"
        delay_ms(500);
}

void envia_SMS() // AT+CMGS
{
        zera_Rx_Buffer();
        fputc(0x41,EVK2); // "A"
        fputc(0x54,EVK2); // "T"
        fputc(0x2B,EVK2); // "+"
        fputc(0x43,EVK2); // "C"
        fputc(0x4D,EVK2); // "M"
        fputc(0x47,EVK2); // "G"
        fputc(0x53,EVK2); // "S"
        fputc(0x3D,EVK2);//=
        fputc(0x22,EVK2);//"
        fputc(0x2B,EVK2);//+
        fputc(0x35,EVK2);
        fputc(0x35,EVK2);
        fputc(0x31,EVK2);
        fputc(0x31,EVK2)
        fputc(0x39,EVK2);
        fputc(0x38,EVK2);
        fputc(0x36,EVK2);
        fputc(0x32,EVK2);
        fputc(0x36,EVK2);
        fputc(0x37,EVK2);
        fputc(0x31,EVK2);
        fputc(0x38,EVK2);
        fputc(0x32,EVK2);
        fputc(0x22,EVK2);
        fputc(0x0D,EVK2);
        fputc(0x0A,EVK2);
        delay_ms(500);
```

```
      for(RX_Wr_Index2=0;RX_Wr_Index2 <= RX_BUFFER_SIZE2;RX_Wr_Index2++)
      {
        if(Rx_Buffer2[RX_Wr_Index2]!=0)
        {
          LED_ERRO_SMS_OFF;
          return;
        }
      }
      LED_ERRO_SMS_ON;
}

void main(void)
{
  set_tris_a(0x00);  // DEFINIÇAO DAS PORTAS A
  set_tris_b(0b00001000);  // DEFINIÇAO DAS PORTAS B
  set_tris_c(0x00);  // DEFINIÇAO DAS PORTAS C
  set_tris_d(0x00);  // DEFINIÇAO DAS PORTAS D
  set_tris_e(0b00);  // DEFINIÇAO DAS PORTAS E
  LED1OFF;
  LED2OFF;
  LED_ERRO_VELOCIDADE_OFF;
  LED_ERRO_ROTACAO_OFF ;
  LED_ERRO_TEMPERATURA_OFF ;
  LED_ERRO_BORBOLETA_OFF ;
  LED_ERRO_SMS_OFF;
  LED_ERRO_GPS_OFF;
  LED_ERRO_E0_OFF ;
  LED_ERRO_E1_OFF;
  enable_interrupts(global);
  enable_interrupts(int_rda);
  enable_interrupts(int_ext);

  inicia_OBD2();
  inicia_EVK2();

  while(1)
```

```
  {
    LED1ON;
    LED2ON;
    delay_ms(1000);
    LED1OFF;
    LED2OFF;
    delay_ms(1000);

    recebe_velocidade();
    recebe_rotacao();
    recebe_temperatura();
    recebe_borboleta();
    recebe_gps();
    delay_ms(1000);

    envia_SMS();
    delay_ms(1000);

    mostra_velocidade();
    mostra_rotacao();
    mostra_temperatura();
    mostra_borboleta();
    mostra_horario();
    mostra_latitude();
    mostra_longitude();

    fputc(0x1A,EVK2); // "ctrl+z" ENCERRA ENVIO SMS
    delay_ms(60000);  // espera de 1min
  }
}
```

8. Anexos

ANEXO A – DIAGRAMA ELETRICO EEC-V

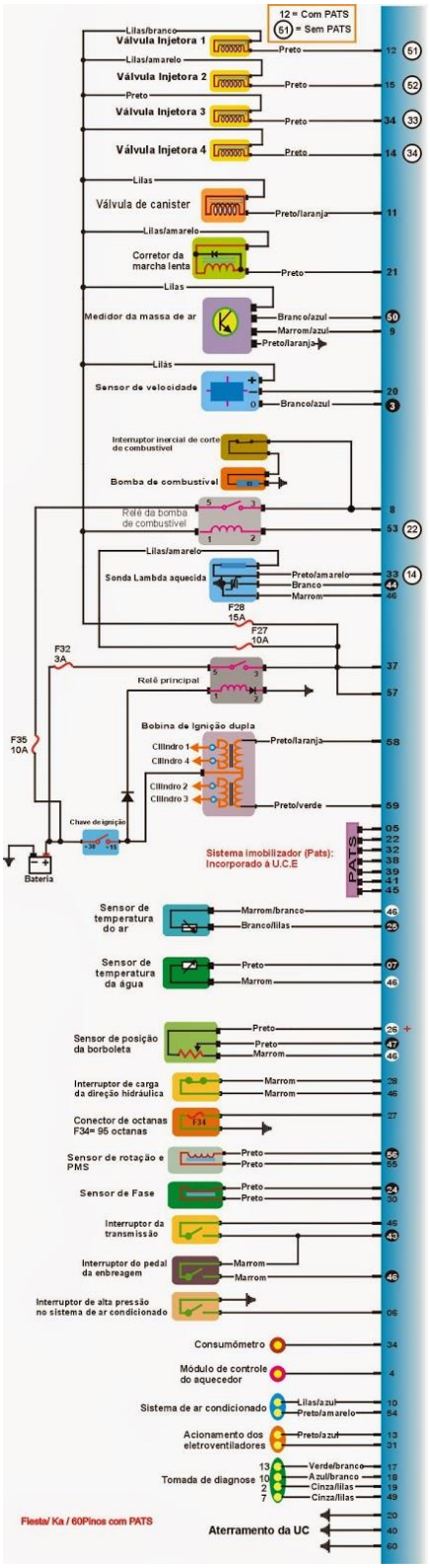


Figura 44 - Esquema elétrico EEC-V - http://falhasdeinjecaoeletronicaautos.blogspot.com.br/2015/01/eec-v-fiesta-10l-e-13l-mondeo-97-ranger.html - 15/02/2015

ANEXO B – COMANDO AT – ELM327

ELM327 AT Commands

Version in which the command first appeared…

version	Command	Description	Group
1.0	@1	display the device description	General
1.3	@2	display the device identifier	General
1.3	@3 ccccccccccc	store the device identifier	General
1.0	<CR>	repeat the last command	General
1.0	AL	Allow Long (>7 byte) messages	OBD
1.2	AR	Automatic Receive	OBD
1.2	AT0	Adaptive Timing Off	OBD
1.2	AT1	Adaptive Timing Auto1	OBD
1.2	AT2	Adaptive Timing Auto2	OBD
1.0	BD	perform a Buffer Dump	OBD
1.0	BI	Bypass the Initialization sequence	OBD
1.2	BRD hh	try Baud rate Divisor hh	General
1.2	BRT hh	set Baud Rate handshake Timeout	General
1.0	CAF0	CAN Automatic Formatting Off	CAN
1.0	CAF1	CAN Automatic Formatting On	CAN
1.4	CEA	turn off CAN Extended Addressing	CAN
1.4	CEA hh	use CAN Extended Address hh	CAN
1.0	CF hh hh hh hh	set the ID Filter to hhhhhhhh	CAN
1.0	CF hhh	set the ID Filter to hhh	CAN
1.0	CFC0	CAN Flow Control Off	CAN
1.0	CFC1	CAN Flow Control On	CAN
1.0	CM hh hh hh hh	set the ID Mask to hhhhhhhh	CAN
1.0	CM hhh	set the ID Mask to hhh	CAN
1.0	CP hh	set CAN Priority (only for 29 bit)	CAN
1.4b	CRA	reset CAN Receive Address filters	CAN
1.3	CRA hhh	set CAN Receive Address to hhh	CAN
1.3	CRA hhhhhhhh	set CAN Receive Address to hhhhhhhh	CAN
1.0	CS	show the CAN Status	CAN
1.4b	CSM0	CAN Silent Mode Off	CAN
1.4b	CSM1	CAN Silent Mode On	CAN
1.0	CV dddd	Calibrate the Voltage to dd.dd volts	Volts
1.4	CV 0000	Restore CV value to factory setting	Volts
1.0	D	set all to Defaults	General
1.3	D0	display of the DLC Off	CAN
1.3	D1	display of the DLC On	CAN

Version in which the command first appeared…

version	Command	Description	Group
1.2	DM1	(J1939) Monitor for DM1 messages	J1939
1.0	DP	Describe the current Protocol	OBD
1.0	DPN	Describe the Protocol by Number	OBD
1.0	E0	Echo Off	General
1.0	E1	Echo On	General
1.1	FC SD [1-5 bytes]	Flow Control Set Data to [...]	CAN
1.1	FC SH hh hh hh hh	Flow Control Set the Header to hhhhhhhh	CAN
1.1	FC SH hhh	Flow Control Set the Header to hhh	CAN
1.1	FC SM h	Flow Control Set the Mode to h	CAN
1.3a	FE	Forget Events	General
1.4	FI	perform a Fast Initiation	ISO
1.0	H0	Headers Off	OBD
1.0	H1	Headers On	OBD
1.0	I	Print the ID	General
1.0	IB 10	set the ISO Baud rate to 10400	ISO
1.4	IB 48	set the ISO Baud rate to 4800	ISO
1.0	IB 96	set the ISO Baud rate to 9600	ISO
1.2	IFR H	IFR value from Header	J1850
1.2	IFR S	IFR value from Source	J1850
1.2	IFR0	IFRs Off	J1850
1.2	IFR1	IFRs Auto	J1850
1.2	IFR2	IFRs On	J1850
1.4	IGN	read the IgnMon input level	Other
1.2	IIA hh	set the ISO (slow) Init Address to hh	ISO
1.3	JE	use J1939 Elm data format	J1939
1.4b	JHF0	J1939 Header Formatting Off	J1939
1.4b	JHF1	J1939 Header Formatting On	J1939
1.3	JS	use J1939 SAE data format	J1939
1.4b	JTM1	set the J1939 Timer Multiplier to 1x	J1939
1.4b	JTM5	set the J1939 Timer Multiplier to 5x	J1939
1.3	KW	display the Key Words	ISO
1.2	KW0	Key Word checking Off	ISO
1.2	KW1	Key Word checking On	ISO
1.0	L0	Linefeeds Off	General
1.0	L1	Linefeeds On	General

Version in which the command first appeared…

version	Command	Description	Group
1.4	LP	go to Low Power mode	General
1.0	M0	Memory Off	General
1.0	M1	Memory On	General
1.0	MA	Monitor All	OBD
1.2	MP hhhh	(J1939) Monitor for PGN hhhh	J1939
1.4b	MP hhhh n	(J1939) Monitor for PGN hhhh, get n messages	J1939
1.3	MP hhhhhh	(J1939) Monitor for PGN hhhhhh	J1939
1.4b	MP hhhhhh n	(J1939) Monitor for PGN hhhhhh, get n messages	J1939
1.0	MR hh	Monitor for Receiver = hh	OBD
1.0	MT hh	Monitor for Transmitter = hh	OBD
1.0	NL	Normal Length (7 byte) messages	OBD
1.4	PB xx yy	set Protocol B options and baud rate	CAN
1.0	PC	Protocol Close	OBD
1.1	PP FF OFF	all Prog Parameters Off	PPs
1.1	PP FF ON	all Prog Parameters On	PPs
1.1	PP xx OFF	disable Prog Parameter xx	PPs
1.1	PP xx ON	enable Prog Parameter xx	PPs
1.1	PP xx SV yy	for PP xx, Set the Value to yy	PPs
1.1	PPS	print a PP Summary	PPs
1.0	R0	Responses Off	OBD
1.0	R1	Responses On	OBD
1.3	RA hh	set the Receive Address to hh	OBD
1.4	RD	Read the stored Data	General
1.3	RTR	send an RTR message	CAN
1.0	RV	Read the Voltage	Volts
1.3	S0	printing of Spaces Off	OBD
1.3	S1	printing of Spaces On	OBD
1.4	SD hh	Store Data byte hh	General
1.0	SH xx yy zz	Set Header	OBD
1.0	SH yzz	Set Header	OBD
1.4	SI	perform a Slow Initiation	ISO
1.0	SP Ah	Set Protocol to Auto, h and save it	OBD
1.0	SP h	Set Protocol to h and save it	OBD
1.3	SP 00	Set Protocol to Auto and save it	OBD
1.2	SR hh	Set the Receive address to hh	OBD

Version in which the command first appeared…

version	Command	Description	Group
1.4	SS	set Standard Search order (J1978)	OBD
1.0	ST hh	Set Timeout to hh x 4 msec	OBD
1.0	SW hh	Set Wakeup interval to hh x 20 msec	ISO
1.4	TA hh	set Tester Address to hh	OBD
1.0	TP Ah	Try Protocol h with Auto search	OBD
1.0	TP h	Try Protocol h	OBD
1.3	V0	use of Variable DLC Off	CAN
1.3	V1	use of Variable DLC On	CAN
1.2	WM [1-6 bytes]	Set the Wakeup Message	ISO
1.0	WM xxyyzzaa	set the Wakeup Message to xxyyzzaa	ISO
1.0	WM xxyyzzaabb	set the Wakeup Message to xxyyzzaabb	ISO
1.0	WM xxyyzzaabbcc	set the Wakeup Message to xxyyzzaabbcc	ISO
1.0	WS	Warm Start	General
1.0	Z	reset all	General