Universidade de Aveiro
Ano 2011
Departamento de Electrónica Telecomunicações e Informática

HUGO RICARDO DA CONCEIÇÃO CUNHA

SISTEMA DE DIAGNÓSTICO OBD2/EOBD PARA VIATURAS AUTOMÓVEIS

Universidade de Aveiro
Ano 2011
Departamento de Electrónica Telecomunicações e Informática

HUGO RICARDO DA CONCEIÇÃO CUNHA

SISTEMA DE DIAGNÓSTICO OBD2/EOBD PARA VIATURAS AUTOMÓVEIS

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor António Ferreira Pereira de Melo, Professor Catedrático do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho aos meus pais por todo o apoio incondicional que me deram desde sempre. À minha esposa, Gisela, pela força, motivação e estabilidade que o seu amor trouxe à minha vida. A todos os meus Amigos, e em especial ao Ludimar Guenda, ao Luís Ferreira, ao Tiago Pires e ao Filipe Abrantes por todo o suporte e ajuda que me deram durante o percurso académico e especialmente durante o desenvolvimento deste projecto. Ao meu primo Fábio e a toda a minha família pelo importante papel que representam na minha vida.

Agradeço também ao Professor Doutor António Ferreira Pereira de Melo por todos os conhecimentos transmitidos, pela oportunidade de desenvolver um projecto que envolve as duas áreas que mais gosto, a electrónica e os automóveis, e por todo o suporte durante o desenvolvimento do mesmo.

Agradeço ainda ao Paulo Martins do DETI pelo seu esforço no fabrico das PCBs deste dispositivo, mesmo com recursos de produção muito limitados.

O júri

Presidente	Prof. Doutor Paulo Bacelar Reis Pedreiras Professor Auxiliar da Universidade de Aveiro
Arguente	Prof. Doutor Luís Miguel Pinho de Almeida Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto
Orientador	Prof. Doutor António Ferreira Pereira de Melo Professor Catedrático do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro

palavras-chave obd, obd2, obdii, eobd, diagnóstico, automóvel, carro, can

resumo Este trabalho apresenta o desenvolvimento de um sistema de diagnóstico automóvel via interface OBD2/EOBD com possibilidade de monitorização de sensores em tempo real, e leitura e eliminação de erros.

keywords obd, obd2, obdii, eobd, diagnostics, road vehicle, car, can

abstract This work presents the development of a road vehicle diagnostics system via OBD2/EOBD interface with real-time sensors monitoring, and errors reading and clearing features.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

CAN – Controller Area Network

ECAN – Enhanced Controller Area Network

CPU – Central Processing Unit

OBD – On-Board Diagnostics

EOBD – European On-Board Diagnostics

JOBD – Japanese On-Board Diagnostics

DTC – Diagnostic Trouble Codes

ISO – International Organization of Standards

SAE – Society of Automobile Engineers

ECU – Engine Control Unit

SID – Service Identification

PID – Parameter Identification

ALU – Arithmetic Logic Unit

SRAM – Static Random Access Memory

ROM – Read-Only Memory

EEPROM – Electrically Erasable Programmable Read-Only Memory

I/O – Input/Output

TX – Transmission

RX – Reception

USART – Universal Synchronous Asynchronous Receiver/Transmitter

SPI – Serial Peripheral Interface

SDO – Serial Data Out

SDI – Serial Data In

SCK – Serial Clock

SS – Slave Select

MOSI – Master Out Slave In

MISO – Master In Slave Out

I2C – Inter-Integrated Circuit

ICSP – In-Circuit Serial Programming

IDE – Integrated Development Environment

ACK – Acknowledge

CRC– Cyclic Redundancy Check

NBT – Nominal Bit Time

NBR – Nominal Bit Rate

TQ – Time Quanta

IPT – Information Processing Time

SJW – Synchronization Jump Width

LCD – Liquid Crystal Display

PLL – Phase Lock Loop

DPLL – Digital Phase Lock Loop

NRZ – Non-Return-to-Zero

PCB – Printed Circuit Board

PDU – Protocol Data Unit

PCI – Protocol Control Information

1| Introdução

1.1| Motivação

A chegada da electrónica à indústria automóvel é a grande responsável pela sua rápida evolução ao longo das últimas décadas.

No entanto, esta evolução e a popularização do automóvel trouxeram alguns problemas, entre eles a poluição.

A Agência para a Protecção Ambiental – ou EPA, Environmental Protection Agency nos Estados Unidos da América – pressionou os fabricantes de automóveis para que se iniciasse um processo de controlo de emissões poluentes.

Entre o final dos anos 70 e o início dos anos 80, com a utilização de sensores e actuadores controlados electronicamente, os construtores puderam estabelecer esse processo de controlo de emissões poluentes, monitorizando os diferentes componentes dos automóveis, permitindo-lhes assim optimizar cada vez mais todos os subsistemas que constituem um automóvel. Esta optimização foi notória com o aumento dos níveis de performance e a diminuição dos níveis de poluição, no caso do controlo electrónico dos componentes do motor.

Criou-se então o conceito de “Diagnóstico a Bordo”, derivado do inglês *On Board Diagnostic*, cujo acrónimo deu o nome a este tipo de sistemas: OBD.

Os primeiros sistemas OBD eram destinados ao uso pelos fabricantes, e cada marca tinha o seu próprio sistema, com diferentes protocolos. Para além da monitorização dos sistemas do motor, no controlo das emissões poluentes, tinham ainda um papel importante na detecção de avarias no sistema.

Em 1988, a SAE – *Society of Automotive Engineers* – especificou um conector standard para os sistemas OBD e um conjunto de sinais de teste.

A partir de Janeiro de 1996, entrou em vigor a especificação OBD2, uma extensão do sistema original, com novos requisitos.

A especificação OBD2 tem utilizado 5 protocolos de comunicação distintos ao longo dos anos:

- ISO 9141-2
- KWP 2000-4
- SAE J1850 PWM
- SAE J1850 VPW
- ISO 15765-4 (CAN 11-bit e 29-bit)

Em 2008 foi definido como protocolo *standard* o último, o protocolo ISO 15765-4, que especifica as condições os requisitos para a utilização do protocolo CAN em sistemas OBD. É sobre este protocolo que o sistema aqui documentado funciona.

Na União Europeia foi definida a especificação EOBD – *European On Board Diagnostics* – que não é mais do que a especificação OBD2 aplicada a veículos com o primeiro registo num país-membro. As características de ambas são idênticas.

De igual forma, o Japão definiu a JOBD como a especificação equivalente a OBD2 para veículos comercializados nesse país.

Com o estabelecimento de todas estas especificações tornou-se possível criar ferramentas quase universais, de utilização mais ou menos simples, para obter informações a partir deste interface.

1.2| Estado da Arte

Existem actualmente diversos dispositivos que permitem monitorizar sensores em tempo real ou ler erros armazenados na unidade de comando do veículo.

A grande maioria destes sistemas são orientados para os profissionais do ramo automóvel e pouco direccionados para um utilizador comum.

Estes dispositivos podem ser baseados num interface de comunicação com um computador, ou podem assumir um formato *standalone*, permitindo maior portabilidade.

Um exemplo desses dispositivos é o VCDS da Ross-Tech (interface para utilização com o computador pessoal) especificamente para veículos do grupo VAG. Quanto aos últimos, os preços podem variar entre 700$USD e os 1300$USD, para as versões profissionais, e entre 250$USD e 350$USD para as versões particulares.

Outro sistema bastante popular é o BOSCH ESItronic para multimarcas, ou o Star Diagnosis System da Mercedes-Benz. O preço destes sistemas pode variar entre centenas de Euros até dezenas de milhares de Euros. Os custos podem aumentar com as actualizações de *software*.

Figura 1 - Equipamento de Diagnóstico Standalone Profissional - Bosch ESItronic KTS 670

A empresa Scantool também oferece algumas soluções. Entre elas está um interface para utilização com um computador pessoal, o OBDLink SX Scan Tool, que custa cerca de 75$USD sem qualquer licença de *software* incluída.

Figura 2 - Interface OBD2 (USB) - OBDLink SX

Existem algumas opções para utilizadores comuns, no formato de interface de ligação com o computador, que não oferecem portabilidade, a menos que o utilizador possua um computador portátil.

Outra desvantagem desta opção é que também necessária a aquisição de um *software* para completar o sistema. Apesar de existirem opções gratuitas, estas são escassas e não têm qualidade.

Actualmente existem também alguns interfaces com ligação *Bluetooth* e *WiFi* que permitem a utilização destes com aplicações desenvolvidas para *smartphones*. Para estes casos a empresa Scantool oferece o OBDLink MX e OBDLink Bluetooth Scan Tool para Bluetooth e o OBDLink Wifi Scan Tool para WiFi, cujos valores se encontram entre os 180$USD e os 200$USD.

Figura 3 - Interface OBD2 (Bluetooth) – OBDLink MX

1.3| Objectivos

O principal objectivo deste projecto é desenvolver um dispositivo de diagnóstico automóvel com possibilidade monitorização de dados em tempo real e de leitura de erros armazenados na unidade de comando de um veículo automóvel.

O dispositivo deve ser compatível com os protocolos *standard* actuais da especificação OBDII/EOBD, os protocolos ISO 11898, ISO 15765 e ISO 15031.

Deverá ainda ser portátil e autónomo, sem necessidade de utilização de um computador ou de outro dispositivo externo.

Este sistema é orientado para o utilizador comum, e como tal deverá ser de fácil utilização e bastante intuitivo.

Com este dispositivo pretende-se que o utilizador possa detectar avarias comuns a partir da leitura de valores erróneos nos sensores ou determinar a existência de erros na unidade de comando. O utilizador pode ainda ler *freeze frame data* – valores dos sensores que são guardados durante a ocorrência de um erro e que são armazenados na memória da ECU (*Engine Control Unit*) – e ainda outros dados que não estão habitualmente disponíveis no quadrante do veículo.

2| Base teórica das tecnologias utilizadas

2.1| Microcontrolador

Actualmente, qualquer sistema embebido, por mais simples que seja, é provavelmente controlado por um microcontrolador.

Este é um circuito integrado que contém um processador – ou CPU (*Central Processing Unit*) – e um conjunto de periféricos – como portas de entrada/saída ou periféricos de comunicação com os mais variados protocolos – que na realidade pode ser visto como um microcomputador programável.

Existem actualmente microcontroladores que podem funcionar controlados por osciladores funcionando até centenas de MHz de velocidade.

Os modelos de microcontroladores variam em função do tipo e do número de periféricos integrados, da memória e das velocidades de oscilador suportadas, entre outros aspectos. Obviamente todas estas variáveis influenciam também o custo de cada microcontrolador. No entanto, regra geral, estes são dispositivos de baixo custo.

Outra característica de grande importância anos neste tipo de dispositivo, é o consumo de energia, uma vez que cada vez mais estão presentes nos sistemas alimentados por bateria. Existem actualmente microcontroladores a consumirem dezenas de µA/MHz em funcionamento normal e dezenas de nA em modo de suspensão (*sleep*). Alguns modelos funcionam inclusivamente com tensões de 1.8V.

Os microcontroladores são muito versáteis já que são dispositivos programáveis – tipicamente em linguagem *Assembly* ou C – e graças à vasta gama de periféricos que oferecem, permitindo assim criar desde simples sistemas de monitorização de sensores até sistemas interactivos de controlo de elevada complexidade.

Exemplo disso são por exemplo os microcontroladores da Atmel baseados nos famosos processadores ARM, que equipam vários *smartphones*.

A Figura 4 apresenta o diagrama de blocos de um exemplo de um microcontrolador.

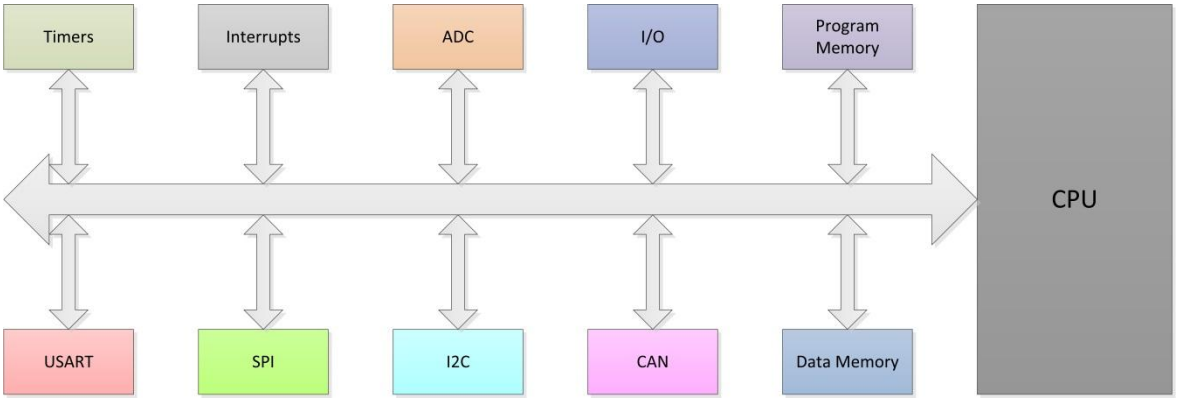

Figura 4 - Diagrama de blocos de um microcontrolador genérico

2.1.1| Funcionamento – Linha Gerais

Para funcionar devidamente, um microcontrolador necessita de um oscilador que forneça um sinal de relógio para sincronizar devidamente todas as operações internas, isto é, a execução das instruções do código do programa. O oscilador pode ser interno ou externo, sendo que o segundo é tipicamente um cristal de quartzo. Um oscilador externo oferece velocidades de relógio mais rápidas, mais precisas e mais estáveis. No entanto alguns microcontroladores de alta performance já integrarem osciladores internos de 64MHz e possibilitam a reconfiguração da velocidade de relógio interno no próprio código de execução e durante o tempo de execução, permitindo assim optimizar o consumo de energia.

Um microcontrolador tem também um temporizador conhecido por *Watchdog Timer* (WDT) que tem como função reiniciar o microcontrolador caso este bloqueie durante um determinado intervalo de tempo programável.

O processador é responsável por executar as instruções programadas e armazenadas na memória. Já as operações lógicas e aritméticas são da responsabilidade da ALU (*Arithmetic Logic Unit*), que é parte integrante do CPU.

2.1.2| Organização de Memória

A memória de um microcontrolador pode normalmente ser dividida em 2 ou 3 tipos, consoante o modelo.

Tipicamente, 2 tipos estão sempre presentes:

- *Program Memory*

- *Data SRAM*

A primeira pode ser do tipo ROM (*Read-Only Memory*) – que só pode ser escrita uma vez – ou do tipo Flash que permite reprogramação, e serve para armazenar o código de execução do programa. Esta é uma memória não-volátil, isto é, não perde informação quando não está alimentada.

Já a memória SRAM (*Static Random Access Memory*) – ou memória estática de acesso aleatório – tem como função armazenar variáveis – inicializadas ou não inicializadas – do código, utilizadas durante a execução do programa. Esta é uma memória volátil, ou seja, quando a alimentação é desligada perde a informação.

Note-se que as variáveis do código estão também armazenadas na *Program Memory*, caso contrário o programa não funcionaria já que a memória SRAM é não-volátil. As variáveis são carregadas da *Program Memory* para a SRAM por operações internas do microcontrolador na sua inicialização.

Também é possível armazenar variáveis na *Program Memory*, no entanto poderá ser necessário efectuar uma operação de cópia para a memória SRAM para que possam ser utilizadas.

Ambas as memórias anteriores estão mapeadas no espaço de memória do microcontrolador.

O terceiro tipo de memória normalmente existente em alguns microcontroladores é a memória EEPROM (*Electrically Erasable Programmable Read-Only Memory*). É uma memória não-volátil que tem como objectivo guardar de forma permanente dados obtidos durante a execução do programa, até que seja efectuada uma operação de eliminação ou sobrescrita na mesma posição de memória. Tipicamente a operação de sobrescrita envolve uma operação de eliminação prévia. Ao contrário das memórias anteriormente referidas, esta não está directamente mapeada no espaço de memória, portanto o acesso a este tipo de memória requer normalmente operações de leitura e escrita como se de um periférico se tratasse.

2.1.3| Entrada/Saída e Periféricos

Os pinos de Entrada/Saída (I/O) têm um papel fundamental nos microcontroladores. Estes permitem a comunicação com qualquer periférico ou

dispositivo, desde ler informação de um simples sensor ou activar um LED para sinalizar um evento até enviar informações para um ecrã (LCD) ou para outros dispositivos mais complexos. Alguns pinos I/O podem ainda ser utilizados como entradas analógicas, já que estes estão normalmente multiplexados com os canais de um módulo ADC (*Analog-to-Digital Converter*).

No entanto, actualmente não existe necessidade de emular alguns protocolos de comunicação nas portas I/O genéricas. Os microprocessadores actuais oferecem uma vasta gama de periféricos dedicados que para além de tornarem as comunicações mais simples e transparentes para o programador, libertam o processador de operações e instruções de emulação de protocolos. Alguns dos periféricos de comunicação mais comuns num microcontrolador são:

- USART (*Universal Synchronous Asynchronous Receiver/Transmitter*)
- SPI (*Serial Peripheral Interface*)
- I2C (*Inter-Integrated Circuit*)

Podendo existir mais do que uma porta de comunicação do mesmo protocolo no mesmo microcontrolador.

2.1.4| Programação

A programação destes dispositivos é tipicamente feita linguagem *Assembly* ou C, havendo no entanto alguns que já suportam linguagens de mais alto nível. A linguagem de programação depende do compilador utilizado.

2.1.4.1| Edição e Compilação

Compilação é a conversão do código editado pelo programador em código máquina, para ser posteriormente carregado para a memória do microcontrolador.

Alguns fabricantes de microcontroladores disponibilizam os seus próprios compiladores específicos – como por exemplo a Microchip™ – outros disponibilizam bibliotecas para utilização com compiladores genéricos como o GCC (*GNU C Compiler*).

Para além dos compiladores, alguns fabricantes disponibilizam ainda *software* IDE (*Integrated Development Environment*) – ou ambiente integrado de desenvolvimento – que engloba edição e compilação num só ambiente gráfico, e algumas funcionalidades

extra como ferramentas de *debug* e de monitorização de registos, ou configurações de compilação.

2.1.4.2| ICSP

ICSP (*In-Circuit Serial Programming*) é um método de carregamento do código máquina para o microcontrolador com o próprio já integrado na placa do sistema.

Este método de carregamento permite que sejam feitas correcções ou actualizações ao *firmware* do sistema sem que seja necessário dessoldar o microcontrolador para que seja reprogramado num programador externo.

O carregamento do *firmware* por ICSP é feito a partir do próprio IDE ou com um *software* dedicado para o efeito, que controla todo o processo de transferência de dados.

Este método permite ainda ler o *firmware* já existente no microcontrolador.

Tipicamente é utilizado um pequeno programador ICSP com um interface físico de 5 pinos.

A Figura 5 apresenta um exemplo desse interface.

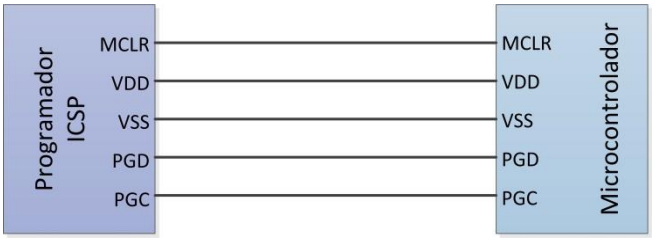

Figura 5 - Esquema de ligação ICSP

Da imagem podem ser identificados os seguintes pinos:

- MCLR – *Master Clear* – Tipicamente utilizado para reinicializar o microcontrolador
- VDD – Tensão de alimentação
- VSS – Pino negativo de alimentação – Tipicamente 0v (massa, GND)
- PGD – *Programmer Data* – Linha de transferência de dados
- PGC – *Programmer Clock* – Linha de sinal de relógio de sincronismo

2.2| Modelo OSI

Ao longo dos anos surgiu a necessidade de criação de standards de intercomunicação entre sistemas heterogéneos.

A ISO (*International Organization for Standardization*) – a maior organização mundial de desenvolvimento de *standards* internacionais – criou o Sub-Comité 16 para "*Open Systems Interconnection*" – abreviado por OSI que deu o nome ao modelo de referência – para desenvolver uma solução.

Foi desenvolvida uma arquitectura por camadas para que fosse possível o desenvolvimento de diferentes protocolos e a intercomunicação entre cada protocolo de diferentes camadas, mantendo o máximo nível de abstracção.

A Tabela 1 representa o modelo de referência OSI – constituído por 7 camadas (*layers*) – os respectivos tipos de dados associados e um pequeno resumo das funções de cada camada.

Tabela 1 - Modelo de referência OSI

	Tipo de Dados	*Layer* / Camada	Função
Host Layers	Dados	*Application* Aplicação	Interface entre o utilizador e a máquina
	Dados	*Presentation* Apresentação	Conversão e apresentação de dados Encriptação/Desencriptação
	Dados	*Session* Sessão	Estabelecimento de sessões entre aplicações
	Segmentos	*Transport* Transporte	Segmentação/União de pacotes entre as camadas adjacentes
Media Layers	Pacotes	*Network* Rede	Endereçamento Lógico Fragmentação/Assemblagem
	Frames	*Data Link* Ligação de Dados	Endereçamento Físico
	Bits	*Physical* Física	Media, Sinal e Transmissão Binária

2.3| ISO 11898 – Protocolo CAN

Apresentado em 1986 por Robert Bosch GmbH, o protocolo CAN (*Controller Area Network*) é um protocolo de comunicação série muito popular na implementação de sistemas de tempo real distribuídos, especialmente na indústria automóvel e na automação industrial.

A sua popularidade surge de um dos seus pontos fortes, a flexibilidade, que permite a sua utilização quer em sistemas que exigem largura de banda considerável, quer em sistemas em que as baixas latências são um elemento fulcral. Outro ponto forte é a simplicidade do barramento e das ligações físicas, que a juntar ao seu baixo custo de implementação, o tornam um protocolo muito atraente do ponto de vista custo-benefício.

2.3.1| Introdução

Actualmente na indústria automóvel, o protocolo CAN é utilizado para interligar sensores, actuadores, módulos de comando, etc. que anteriormente exigiam cablagens mais complexas, e que podem agora tirar partido da simplicidade deste tipo de comunicação, e podendo atingir velocidades até 1Mbit/s, na especificação actual (2.0). Normas europeias exigem a sua implementação nos automóveis fabricados a partir de 2008, sendo no entanto já implementado por algumas marcas desde 1992 (Mercedes-Benz).

A especificação 2.0 deste protocolo divide-se em 2 partes: 2.0A e 2.0B.

A primeira envolve todas as definições da versão 1.2 do protocolo CAN. O desenvolvimento deste trabalho foca-se essencialmente sobre a versão B do protocolo, que veio introduzir um novo formato de mensagens em relação à versão anterior da especificação, mantendo no entanto total retro-compatibilidade.

O protocolo CAN é caracterizado pelas seguintes propriedades:

- Priorização de mensagens
- Latências garantidas
- Flexibilidade de configuração
- Recepção *multicast* com sincronização temporal
- *Multi-master*

- Detecção e sinalização de erros
- Retransmissão automática de mensagens corrompidas assim que o barramento esteja livre
- Distinção entre erros temporários e falhas permanentes nos nós e desactivação automática dos respectivos nós

Este protocolo funciona num modelo Dominante-Recessivo, em que o bit ‘0’ é dominante e o bit ‘1’ é recessivo.

2.3.2| Baud Rate

Relativamente à velocidade de comunicação, esta norma estabelece duas:

- 500 Kbit/s
- 250 Kbit/s

A escolha da velocidade utilizada no sistema de diagnóstico OBD2 de cada veículo é da responsabilidade do fabricante, sendo depois necessário que os dispositivos externos de diagnóstico façam a devida detecção da velocidade utilizada aquando da inicialização do sistema.

2.3.3| Camadas

Esta especificação cobre as duas camadas inferiores do modelo OSI, a camada física (*Physical Layer*) e a camada de ligação de dados (*Data Link Layer*).

As restantes camadas acima não estão definidas neste protocolo, permitindo assim um elevado grau de liberdade e flexibilidade no *design* e implementação dos sistemas.

A Tabela 2 representa as camadas do modelo de referência OSI definidas por este protocolo.

Tabela 2 - Camadas do modelo de referência OSI definidas pela norma ISO 11898

Layer / Camada		Função
Data Link Ligação de Dados	*Logical Link Control* Controlo Lógico da Ligação	Filtros de aceitação de mensagens Notificação de sobrecarga Gestão de recuperação
	Medium Access Control Controlo de Acesso ao Meio	Encapsulamento/Desencapsulamento de mensagens Codificação das frames Detecção e sinalização de erros Serialização/de-serialização
Physical Física		Codificação/descodificação dos bits Bit timing e sincronização

2.3.3.1| Camada Física (Physical Layer)

A camada física (*Physical Layer*) define como é feita a transmissão e recepção dos dados para e do barramento, isto é, lida nomeadamente com o *Bit Timing*, a codificação e a sincronização. Nesta camada, o protocolo já permite uma certa flexibilidade uma vez que não define características fixas para a linha.

O interface físico que cobre esta camada consiste num barramento bifilar (CAN-H e CAN-L), com tensões diferenciais, terminado com uma impedância de 120Ω em cada extremidade. Cada nó é ligado a essas mesmas linhas.

2.3.3.2| Camada de ligação de dados (*Data Link Layer*)

A camada de ligação de dados (*Data* Link Layer) está dividida em duas subcamadas:

MAC (*Medium Access Control* / Controlo de Acesso ao Meio)

LLC (*Logical Link Control* / Controlo Lógico da Ligação)

Estas são responsáveis pelos parâmetros do protocolo propriamente dito, isto é, por todo o controlo de mensagens, enviadas e recebidas, e por todos os processos inerentes ao tratamento das mesmas.

2.3.3.2.1| Subcamada MAC (*Medium Access Control*)

A subcamada MAC (*Medium Access Control*) pode ser vista como o *kernel* do protocolo, ou seja, estabelece um interface entre a camada física e as camadas superiores, estando responsável por diversas tarefas:

- Encapsulamento/Desencapsulamento de dados
- Codificação de *frames*
- Gestão do acesso ao meio
- Detecção e sinalização de erros
- Notificações
- Serialização/De-serialização

2.3.3.2.2| Subcamada LLC (*Logical Link Control*)

A subcamada LLC (*Logical Link Control*) está acima da subcamada anterior e é responsável pela filtragem de mensagens, pelas notificações de sobrecarga e pela gestão de recuperação.

2.3.4| Arbitragem

No protocolo CAN, as questões de arbitragem são resolvidas por comparação directa de bits. Os bits dominantes (‘0’) têm sempre prioridade perante os bits recessivos (‘1’).

Tomemos o exemplo da Figura 6 em que 4 nós iniciam transmissão em simultâneo com as seguintes sequências de bits.

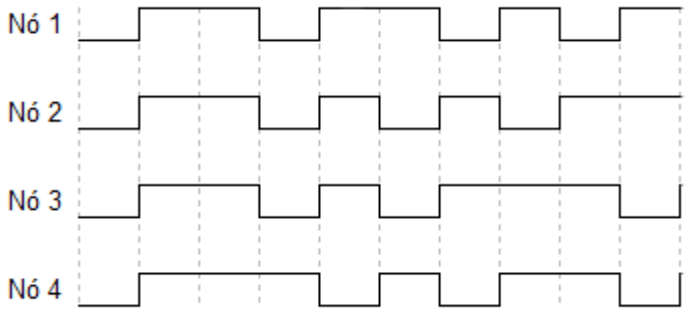

Figura 6 - Diagrama temporal de exemplo de 4 nós CAN a transmitir em simultâneo

Podemos concluir que pela ordem de maior para menor prioridade, a arbitragem é ganha pelos nós:

1. Nó 2
2. Nó 3
3. Nó 1
4. Nó 4

2.3.5| Mensagens

O protocolo CAN é baseado no envio de mensagens, e como tal existem vários tipos e formatos de mensagens para cada fim.

Quando uma mensagem é enviada no barramento, todos os nós a recebem. Cabe ao mecanismo de filtragem de cada nó determinar se a mensagem é utilizada ou ignorada.

2.3.5.1| Formato

As mensagens CAN podem assumir dois formatos: *Standard Frames* e *Extended Frames*.

A principal diferença entre estes reside essencialmente na dimensão do campo *Identifier*, 11 bits e 29 bits respectivamente. No entanto as diferenças serão explicadas com maior detalhe mais adiante neste capítulo.

Cada nó determina se a mensagem é utilizada ou ignorada comparando o campo *Identifier* com os filtros estabelecidos. Se uma mensagem é utilizada por um determinado nó por ter ocorrido uma correspondência com os filtros deste, nada impede outros nós de utilizarem a mesma mensagem. Posto isto, o campo *Identifier* não deve ser visto como um endereço, mas sim como um identificador do tipo de mensagem que é enviada nos campos de dados, que permite a cada nó determinar se a mensagem deve ser utilizada ou ignorada.

Quando o barramento está livre, todos os nós podem enviar mensagens. No entanto, o nó de prioridade inferior interromperá a sua transmissão assim que um dos bits da sua mensagem seja diferente do bit correspondente da mensagem do nó com maior prioridade. O nó cujo endereço tenha o menor valor tem maior prioridade, dado o facto de ser utilizado um modelo *Dominante-Recessivo*, conforme descrito anteriormente.

2.3.5.1.1| Standard Frame vs Extended Frame

Como já foi referido, as mensagens dividem-se em dois formatos, *Standard Frame* e *Extended Frame*.

Em relação à estrutura de uma mensagem CAN, esta pode ser subdividida em 4 campos: *Arbitration Field*, *Control Field*, Data Field e *CRC Field*. O primeiro engloba os parâmetros relacionados com a arbitragem, isto é, com as prioridades e identificação de mensagens, o segundo é constituído pelos parâmetros relacionados com o controlo, nomeadamente definir se a mensagem é *Standard* ou *Extended*, o terceiro campo corresponde à zona da *frame* que contém os dados transportados na mensagem (*payload*), que pode variar entre 0 e 8 *bytes*, e finalmente o último corresponde à zona de verificação de erros. Adicionalmente existem ainda alguns sinais de início e fim de trama e ainda sinais de *acknowledge*.

2.3.5.2| Tipo

As mensagens do protocolo CAN podem ainda ser divididas em quatro tipos:

- *Data Frame*
- *Remote Frame*
- *Error Frame*
- *Overload Frame*

2.3.5.2.1| Data Frame

Este tipo de mensagem transporta informação entre o transmissor e os receptores. Pode ter ambos os formatos, *Standard* e *Extended* e pode ter ou não campo de dados, isto é, pode transportar entre 0 a 8 bytes de informação. A dimensão do campo de dados é indicada pelo parâmetro DLC (*Data Length Code*), que tem o valor correspondente ao número de bytes de dados transportados.

Este tipo de *frame* é definido com o bit RTR (*Remote Transmission Request*) dominante, isto é, a '1'.

2.3.5.2.2| Remote Frame

Uma *Remote Frame* é transmitida no sentido de solicitar a transmissão de uma mensagem do tipo *Data Frame* com o mesmo valor do campo *Identifier*, por parte de outro(s) nó(s). Este tipo de mensagem pode igualmente ter ambos os formatos.

O bit RTR (*Remote Transmission Request*) recessivo identifica uma mensagem como *Remote Frame*.

Uma *Remote Frame* não tem campo de dados, independentemente de o valor do parâmetro DLC (*Data Length Code*), que indica a dimensão do campo de dados, poder indicar o contrário. Este indica a dimensão dos dados da respectiva *Data Frame* que será enviada como resposta.

2.3.5.2.3| Error Frame

Uma *Error Frame* é transmitida quando é detectado um erro no barramento. Este tipo de *frame* é composto por dois elementos: *Error Flag* e *Error Delimiter*.

O nó que detecta o erro da mensagem envia uma *Error Frame* com a respectiva *flag* de erro activo (*Active Error*) ou passivo (*Passive Error*), seja o nó transmissor ou o nó receptor.

Uma *Active Error Flag* – correspondente a um erro activo – é composta por 6 bits dominantes (‘0’) consecutivos. No caso de uma *Passive Error Flag* – erro passivo – é composta por 6 bit recessivos (‘1’) consecutivos.

O *Error Delimiter* é composto por 8 bits recessivos (‘1’) consecutivos.

A detecção e sinalização de erros encontram-se detalhadas mais adiante neste documento.

2.3.5.2.4| Overload frame

A *Overload Frame* é enviada quando se pretende solicitar um intervalo de tempo extra entre duas *frames* recebidas, quer *Data Frames*, quer *Remote Frames*.

À semelhança de uma *Error Frame*, este tipo de *frame* divide-se em duas partes: *Overload Flag* e *Overload Delimiter*.

2.3.5.2.5| Interframe Space

Este espaço é um intervalo que existe entre transmissões, em que apenas é possível enviar *Overload Frames*, não são permitidos envios de *Data Frames* ou *Remote Frames*.

É constituído por duas partes: *Intermission* e *Bus Idle*.

O primeiro consiste em 3 bits recessivos (‘1’) e o segundo tem tamanho arbitrário, que poderá ser nulo caso uma mensagem já esteja em espera para transmissão durante a transmissão da última mensagem, ou seja, é transmitida a partir do primeiro bit a seguir ao campo *Intermission*.

2.3.5.3| Estrutura

2.3.5.3.1| Standard Frame (Data Frame/Remote Frame)

A Tabela 3 explica em detalhe os campos e os parâmetros de uma *Standard Frame*, do tipo *Data Frame* ou *Remote Frame*.

Tabela 3 - Descrição dos parâmetros de uma *Standard Frame*

Campo	Parâmetro	Descrição
	SOF – Start of Frame (1 bit)	• Marca o início de uma *frame* (*Standard* ou *Extended*, sendo simplesmente um bit a '0' (dominante)
Arbitration Field	*Identifier (11 bits)*	• Identifica a mensagem, permitindo a cada nó determinar se a mensagem lhe é dirigida
	RTR – Remote Transmission Request (1 bit)	• Identifica se a mensagem é do tipo *Remote Frame* • Numa *Data Frame* está a '0' (dominante) • Numa *Remote Frame* está a '1' (recessivo)
Control Field	*IDE – Identifier Extension Bit (1 bit)*	• Identifica se a mensagem está no formato *Standard* ou *Extended* • Numa *Standard Frame* está a '0' (dominante)
	R0 – Reserved Bit (1 bit)	• Reservado • Deve ser transmitido a '0' (dominante), apesar ser aceite em ambos os estados
	DLC – Data Length Code (4 bits)	• Numa *Data* Frama indica a dimensão (em bytes) dos dados transportados no *Data* Field • Numa *Remote Frame* indica a dimensão (em bytes) dos dados a serem transportados na *Data Frame* de resposta a esta
Data Field	*Data* (0 a 8 bytes)	• Dados transportados na mensagem • Numa *Data Frame* pode ter entre 0 a 8 bytes • Numa *Remote Frame* não existe
CRC Field	*CRC Sequence – Cyclic Redundancy Check (15 bits)*	• Sequência de bits que permite detectar a existência de erros na recepção da mensagem
	CRC Delimiter (1 bit)	• Bit a '1' (recessivo) que delimita a CRC Sequence
ACK Field	*ACK Slot (1 bit)*	• Indica que uma mensagem foi correctamente recebida (verificando o campo CRC) • O transmissor envia este bit a '1' (recessivo) • O Receptor envia (em simultâneo) o bit a '0' (dominante) em caso de recepção com sucesso

	ACK Delimiter (1 bit)	• Bit a '1' (recessivo) que delimita o ACK Slot, fazendo com que este fique entre dois bits recessivos
	EOF – End of Frame (7 bits)	• Delimita a *frame*, colocando 7 bits recessivos no final da mesma • Não é detectado como erro passivo (6 bits recessivos consecutivos)

A Figura 7 representa a estrutura da trama de dados enviados para o barramento, referente a uma *Standard Frame*.

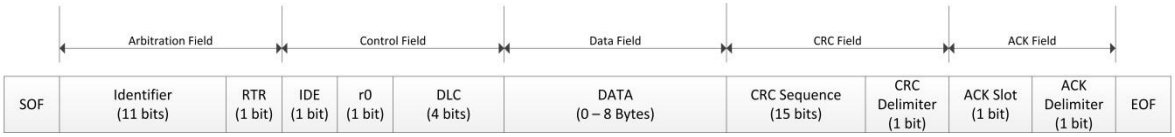

Figura 7 - Estrutura de uma *Standard Frame*

2.3.5.3.2| Extended Frame (Data Frame/Remote Frame)

No caso de uma mensagem no formato *Extended* existem algumas diferenças no que toca aos campos acima descritos, nomeadamente no *Arbitration Field* e em alguns outros bits da *frame*, conforme descrito na tabela seguinte onde se explicam em detalhe os campos e os parâmetros deste formato.

Tabela 4 - Descrição dos parâmetros de uma *Extended Frame*

Campo	Parâmetro	Descrição
	SOF – Start of Frame (1 bit)	• Marca o início de uma *frame* (*Standard* ou *Extended*, sendo simplesmente um bit a '0' (dominante)
Arbitration Field	*Identifier (Base ID) (11 bits)*	• Identifica a mensagem, permitindo a cada nó determinar se a mensagem lhe é dirigida • Não pode ser totalmente composto por bits recessivos ('1')
	SRR – Substitute Remote Request (1 bit)	• Substitui o bit RTR nesta posição da *frame* uma vez que este estará mais adiante na trama, neste formato de mensagem • Bit a '1' (recessivo)
	IDE – Identifier Extension Bit (1 bit)	• Identifica se a mensagem está no formato *Standard* ou *Extended* • Numa *Extended Frame* está a '1' (recessivo)
	Identifier (Extended ID) (18 bits)	• Parte adicional do parâmetro *Identifier*, que será concatenada com o primeiro formando assim os 29 bits *Identifier* de uma *Extended Frame*

	RTR – Remote Transmission Request (1 bit)	• Identifica se a mensagem é do tipo *Remote Frame* • Numa *Remote Frame* está a '1' (recessivo)
Control Field	*R1 – Reserved Bit (1 bit)*	• Reservado • Deve ser transmitido a '0' (dominante), apesar ser aceite em ambos os estados
	R0 – Reserved Bit (1 bit)	• Reservado • Deve ser transmitido a '0' (dominante), apesar ser aceite em ambos os estados
	DLC – Data Length Code (4 bits)	• Numa *Data* Frama indica a dimensão (em bytes) dos dados transportados no *Data* Field • Numa *Remote Frame* indica a dimensão (em bytes) dos dados a serem transportados na *Data Frame* de resposta a esta
Data Field	*Data* (0 a 8 bytes)	• Dados transportados na mensagem • Numa *Data Frame* pode ter entre 0 a 8 bytes • Numa *Remote Frame* não existe
CRC Field	*CRC Sequence – Cyclic Redundancy Check (15 bits)*	• Sequência de bits que permite detectar a existência de erros na recepção da mensagem
	CRC Delimiter (1 bit)	• Bit a '1' (recessivo) que delimita a CRC Sequence
ACK Field	*ACK Slot (1 bit)*	• Indica que uma mensagem foi correctamente recebida (verificando o campo CRC) • O transmissor envia este bit a '1' (recessivo) • O Receptor envia (em simultâneo) o bit a '0' (dominante) em caso de recepção com sucesso
	ACK Delimiter (1 bit)	• Bit a '1' (recessivo) que delimita o ACK Slot, fazendo com que este fique entre dois bits recessivos
	EOF – End of Frame (7 bits)	• Delimita a *frame*, colocando 7 bits recessivos no final da mesma • Não é detectado como erro passivo (6 bits recessivos consecutivos)

A Figura 8 representa a estrutura da trama de dados enviados para o barramento, referente a uma *Extended Frame*.

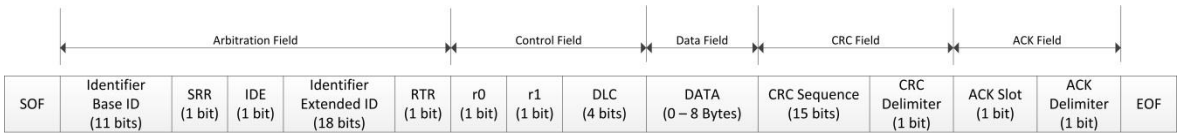

Figura 8 - Estrutura de uma *Extended Frame*

2.3.5.3.3| Arbitragem

De notar que pode ocorrer uma situação em que o Base ID de uma mensagem no formato *Extended* tem o mesmo valor que o *Identifier* de uma *Standard Frame*, e que estas seja transmitidas no barramento em simultâneo. Esta situação está salvaguardada pelos bits RTR, SRR e IDE, garantindo sempre prioridade para a *Standard Frame*. Senão vejamos a Figura 9 com exemplos de duas *Standard Frames*, *Data Frame* e *Remote Frame*, e duas *Extended Frames*, *Data Frame* e *Remote Frame*, respectivamente.

	Standard Frame						
	SOF	Identifier	RTR	IDE	r0	..	EOF
Data Frame	0	01010101010	0	0	0	...	1111111
Remote Frame	0	01010101010	1	0	0	...	1111111

	Extended Frame									
	SOF	Identifier (Base ID)	SRR	IDE	Identifier (Extended ID)	RTR	r1	r0	..	EOF
Data Frame	0	01010101010	1	1	010101010101010101	0	0	0	...	1111111
Remote Frame	0	01010101010	1	1	010101010101010101	1	0	0	...	1111111

Figura 9 - Comparativo de arbitragem entre 4 frames de formatos e tipos diferentes

Tendo em conta que os bits a ‘0’ são dominantes, podemos concluir que uma mensagem no formato *Standard Frame* tem sempre prioridade sobre uma *Extended Frame*, e uma *Data Frame* tem sempre prioridade sobre uma *Remote Frame*, assumindo a situação em que os campos *Identifier* têm o mesmo valor.

2.3.6| Bit Timing

A especificação CAN define *Nominal Bit Ra*te (NBR) como o número de bits enviados por segundo no barramento, por um transmissor ideal e sem re-sincronização, e *Nominal Bit Time* (NBT) como o período de um bit.

Defina-se o *Nominal Bit Rate* (NBR), *Nominal Bit Time* (NBT) e a sua relação:

$$NBR = f_{bit}$$

$$NBT = t_{bit}$$

$$f_{bit} = \frac{1}{t_{bit}}$$

O período de um bit (t_{bit}) é ainda dividido em quatro segmentos principais:

- SyncSeg – Segmento de Sincronização

Tem como função sincronizar os nós com o barramento. São esperadas transições durante o decorrer deste segmento, para ser feita a sincronização.

- PropSeg – Segmento de Propagação

A sua função é compensar os atrasos de propagação de sinal entre os nós CAN, e é definido como o dobro da soma dos tempos de propagação no barramento, do atraso do comparador e do atraso do *driver*.

- PhaseSeg1 (PS1) – Segmento de Fase 1
- PhaseSeg2 (PS2) – Segmento de Fase 2

São usados para compensar os erros de fase de transição. Podem ser aumentados ou diminuídos pela re-sincronização.

$$t_{bit} = t_{SyncSeg} + t_{PropSeg} + t_{PS1} + t_{PS2}$$

Estes segmentos estão dispostos conforme apresenta a Figura 10.

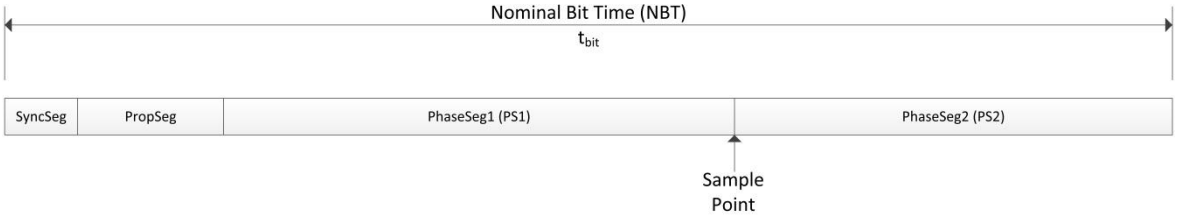

Figura 10 - Segmentação do período de um bit

A cada segmento do t_{bit} referido na equação anterior corresponde um determinado número de *Time Quanta* (TQ), que será determinado mais adiante neste documento.

Definam-se ainda os parâmetros *Sample Point, Synchronization Jump Width* (SJW) e *Information Processing Time* (IPT).

O *Sample Point* é o momento em que é feita a amostragem efectiva do nível lógico do bit, e está temporalmente situado entre o PS1 e o PS2.

Synchronization Jump Width (SJW) é a largura do ajuste feito nos valores de PS1 ou PS2, durante a re-sincronização.

Information Processing Time (IPT) é o tempo dado ao nó CAN para processar o nível lido no Sample Point, tendo início precisamente neste e tendo no máximo a duração de 2 TQ.

A Tabela 5 mostra a duração de cada segmento em TQ.

Tabela 5 - Duração dos segmentos estruturam o período de um bit

Segmento	Duração (TQ)
SyncSeg	1
PropSeg	1 a 8 (programável)
PhaseSeg1	1 a 8 (programável)
PhaseSeg2	2 a 8 (programável)
IPT	1 a 2
SJW	1 a Mínimo(4, PS1) (programável)

Note-se que a duração mínima do PhaseSeg2 é de 2 TQ. Isto verifica-se devido ao facto de o IPT ter início no momento entre PS1 e PS2, e de este ter a duração máxima de 2 TQ, portanto o valor de mínimo de PS2 tem que garantir o valor máximo de IPT.

$$PS2_{min} = IPT_{max} = 2\,TQ$$

A norma ISO 11898 define ainda que o número de TQ no período de um bit deve ser 16 ou 20, e que o *Sample Point* deverá estar a cerca de 80% do período de um bit.

No capítulo do controlador CAN encontram-se os cálculos necessários para a devida configuração do *Bit Timing*.

2.3.7| Erros

2.3.7.1| Detecção

O protocolo especifica cinco tipos de erros:

2.3.7.1.1| *Bit Error* – Erro de Bit

Cada nó monitoriza o barramento enquanto está a enviar uma mensagem, como tal caso esteja a enviar um bit para o barramento e detecte neste um bit diferente, gera um erro: o Erro de Bit.

Uma das excepções para este caso é quando o nó está a transmitir bit recessivo (‘1’) do Campo de Arbitragem – o *Arbitration Field* como foi referido anteriormente neste documento – e detecta um bit dominante no barramento, que é considerado um erro, mas sim uma questão de prioridade. A outra excepção é durante o ACK *Slot*, em que o nó transmissor envia este bit recessivo (‘1’) e o receptor coloca o bit dominante (‘0’) em simultâneo, sinalizando a recepção com sucesso.

Este erro é detectado durante o período do bit.

2.3.7.1.2| *Stuff Error*

Tal como já referido anteriormente neste documento, um nó CAN detecção um erro quando são detectados 6 bit consecutivos com o mesmo valor, trata-se pois de um *Stuff Error*. Este erro deve ser evitado utilizando a técnica de *bit stuffing*.

Este erro é detectado durante o período do bit.

2.3.7.1.3| *CRC Error* – Erro de Redundância Cíclica

Este erro ocorre quando o valor do campo CRC do receptor não coincide com o valor do campo CRC do transmissor.

2.3.7.1.4| *Form Error* – Erro de Formato

Ocorre um Erro de Formato quando o bit de um determinado campo da *frame* viola as regras estipuladas para esse campo nesse mesmo formato. Por exemplo, se o formato da mensagem prevê que um determinado bit tem que ser recessivo (‘1’) naquela posição da *frame*, e é detectado um bit dominante (‘0’), ou vice-versa.

Este erro é detectado durante o período do bit.

2.3.7.1.5| *Acknowledgement Error* – Erro de Recepção

Um nó transmissor deve detectar um *Acknowledgement Error* quando não detectar o bit ACK *Slot* dominante – bit que é enviado pelo transmissor como recessivo (‘1’) – que deve ser enviado pelo receptor quando este recebe correctamente a mensagem. Neste caso trata-se de um *Acknowledge Error*, e a mensagem será reenviada.

Este erro é detectado durante o período do bit.

2.3.7.2| Sinalização

Os erros detectados por um nó CAN são sinalizados em momentos diferentes em função do erro detectado. Esta sinalização é feita com o envio de uma *Error Frame*, constituída por uma *Error flag* e um *Error Delimiter* – conforme descrito anteriormente neste documento.

Para o caso de um erro de CRC o receptor inicia o envio da *Error Frame* a seguir ao ACK *Delimiter* da mensagem, sinalizando assim o erro de recepção da mensagem.

Para os outros tipos de erro, o envio da *Error Frame* é iniciado no bit seguinte à detecção do erro, já que estes são detectados durante o período do respectivo bit.

O que define qual o tipo de *flag* a ser enviada na sinalização de um erro é o estado do nó CAN que o detecta: nó "*error active*" ou nó "*error passive*".

A estes estados pode ser adicionado um terceiro, o "*bus-off*" em que o nó não pode transmitir para o barramento.

Um nó "*error active*" envia uma *Active Error Flag* e um nó "*error passive*" envia uma *Passive Error Flag*, quando detectam um erro.

A alternação de cada nó CAN entre cada estado deve-se a dois contadores internos do próprio nó: o contador de erros de transmissão (*Transmit Error Count*) e o contador de *error* de recepção (*Receive Error Count*).

Cada vez que é detectado um erro de transmissão ou de recepção, o contador de erros de transmissão ou recepção, respectivamente, é incrementado em 8.

Uma situação particular é o caso em que um nó é o único nó activo no barramento e este envia uma mensagem. Uma vez que nenhum nó recebe a mesma, vai ser detectado um *Acknowledgement Error* que será contabilizado. Neste caso o nó pode passar a "*error passive*" mas não passará a "*bus-off*" por esta razão.

Quando um nó envia ou recebe uma mensagem com sucesso – sem erros – os respectivos contadores são decrementados em 1, a menos que já tenham o valor 0.

Um nó é "*active error*" se ambos os contadores tiverem valores inferiores a 128, e é "*error passive*" se um dos contadores tem valor igual ou superior a 128.

Se o contador de erros de transmissão de um nó atingir o valor 256, este entra no estado "*bus-off*", não podendo assim enviar mensagens para o barramento, os *drivers* de transmissão estão desligados. Esta capacidade de desactivação dos nós com falhas permanentes permite aos outros nós continuar a comunicar normalmente.

Um nó em "*bus-off*" pode no entanto continuar monitorizar o barramento, e após 128 ocorrências de 11bits recessivos ('1') consecutivos, os seus contadores voltam a 0, passando novamente ao estado "*error active*".

2.3.8| Codificação

Relativamente a *Data Frames* e *Remote Frames*, os campos *Start of Frame*, *Arbitration Field*, *Control Field*, Data Field e *CRC Sequence* são codificados por *bit stuffing*. Este método visa evitar o envio de mais de 5 bits do mesmo valor consecutivos, adicionando um bit complementar de valor diferente a seguir ao quinto bit consecutivo.

Error Frames e *Overload Frames* não são codificados por este método.

2.4| ISO 15765 – Diagnósticos sobre Protocolo CAN

A norma ISO 15765 estabelece os requisitos para o funcionamento de uma comunicação CAN dentro dos parâmetros OBD2.

Entre outros aspectos, esta norma visa verificar o *baud rate* da comunicação, que tipo de CAN *Identifier* é suportado pelo veículo – 11 bits ou 29 bits – e estabelecer procedimentos de inicialização de comunicação.

Relativamente ao modelo de referência OSI, a Tabela 6 representa a aplicabilidade desta norma.

Tabela 6 - Norma ISO 15765 de acordo com o modelo de referência OSI

Layer / Camada	Norma / ISO
Application Aplicação	-
Presentation Apresentação	-
Session Sessão	ISO 15765
Transport Transporte	-
Network Rede	ISO 15765
Data Link Ligação de Dados	ISO 15765
Physical Física	ISO 15765

Antes da explicação do procedimento de inicialização definido por esta norma, é necessário referir alguns parâmetros que compõem uma mensagem OBD2 – definidos pela norma ISO 15031 mais adiante neste documento – que são:

- SID – *Service Identification*
- PID – *Parameter Identification*

É estabelecida então uma mensagem de inicialização que consiste num pedido de Serviço $01 – pedido de informação de diagnóstico actual (consultar capítulo ISO 15031 para mais informação) – e PID $00 (que visa solicitar ao sistema OBD2 a lista de PIDs suportados. Esta mensagem terá o formato *Standard* (11-bits *Identifier*) e/ou *Extended* (29-bit *Identifier*).

Existe ainda a variável *baud rate* que não é conhecida *A priori*, já que é opção do fabricante dentro das opções definidas na norma ISO 11898. Como tal, é necessário que o equipamento externo de diagnóstico verifique qual o *baud rate* do veículo durante o processo de inicialização.

Caso o processo de inicialização não seja concluído com sucesso, o dispositivo de diagnóstico deve efectuar a configuração do *baud rate* alternativo e repetir o processo.

Em caso de insucesso na segunda tentativa, considera-se então que o veículo não é compatível com a norma ISO 15765.

Esta norma define ainda que durante todo o processo de inicialização do sistema não é permitida qualquer intervenção por parte do utilizador.

A Figura 11 ilustra o processo de inicialização e verificação de compatibilidade com a norma ISO 15765.

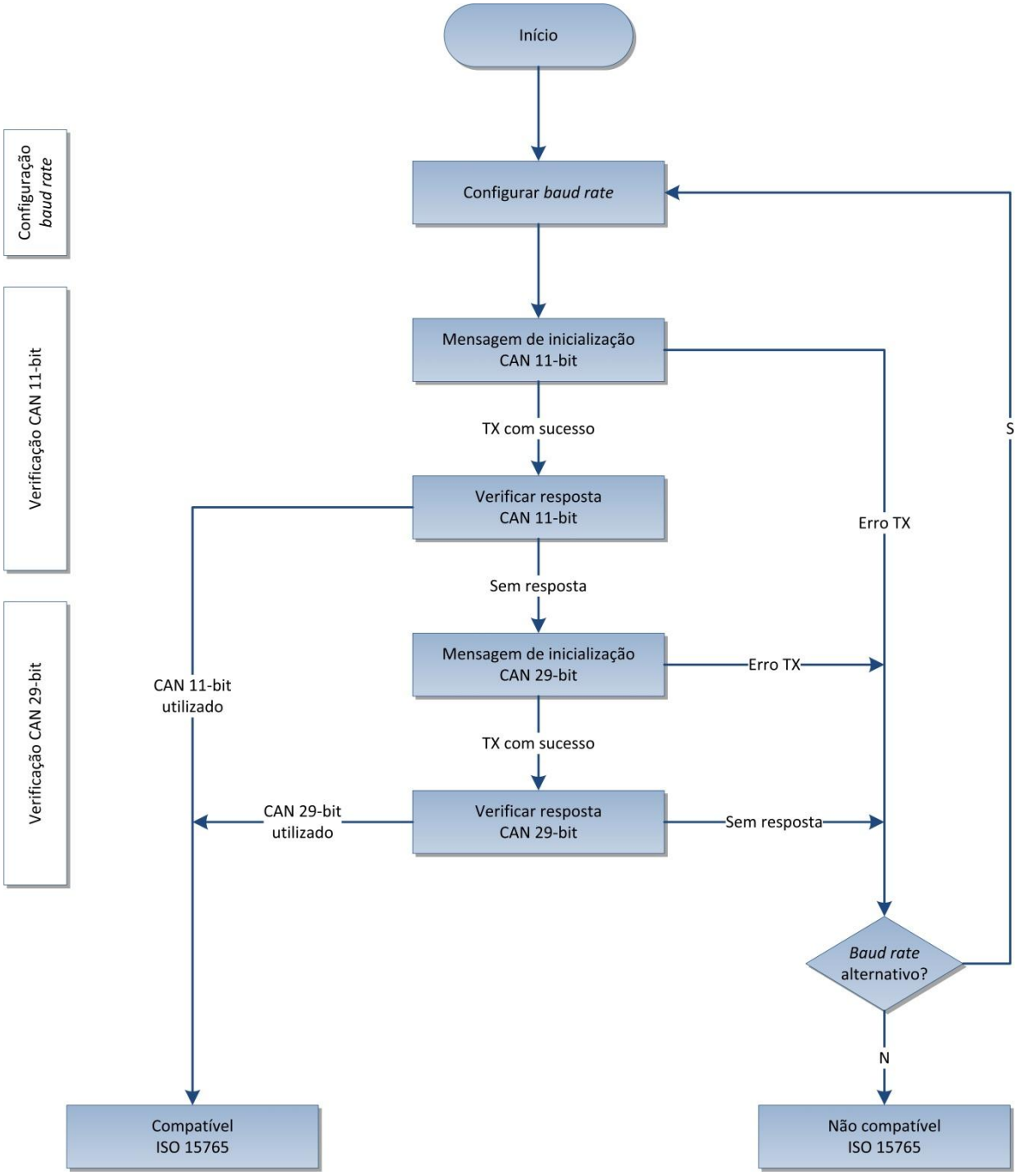

Figura 11 - Diagrama do processo de inicialização da norma ISO 15765

2.4.1| Camada Física (*Physical Layer*)

Do ponto de vista da Camada Física (*Physical Layer*) esta norma define as terminações físicas do circuito eléctrico do sistema externo de diagnóstico na ligação com o barramento.

A Figura 12 apresenta um esquema dessa ligação.

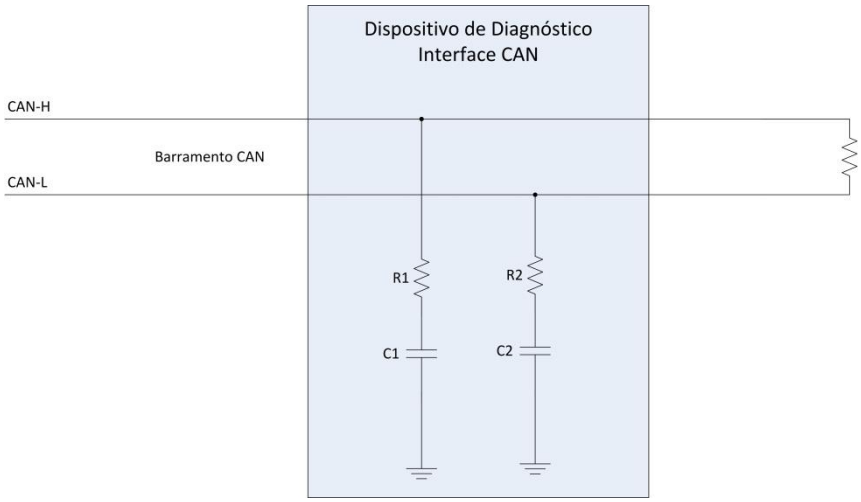

Figura 12 - Esquema de ligação entre o equipamento externo de diagnóstico e o barramento CAN

Os componentes R1, R2, C1 e C2 deverão respeitar os valores da Tabela 7.

Tabela 7 - Componentes da ligação física entre o equipamento externo de diagnóstico e o barramento CAN

Componente	Mínimo	Nominal	Máximo
R1, R2	90 Ω	100 Ω	110 Ω
C1, C2	470 pF	560 pF	640 pF
NOTA: **R1 = R2** **C1 = C2**			

2.4.2| Camada de Ligação de Dados (*Data Link Layer*)

Relativamente à Camada de Ligação de Dados (*Data Link Layer*) esta norma define alguns aspectos do endereçamento na comunicação OBD2.

Define que os equipamentos externos de diagnóstico devem ser compatíveis com os dois formatos de mensagens CAN – *Standard Frame* (11-bit *Identifier*) e *Extended Frame* (29-bit *Identifier*) – no entanto cada veículo só pode utilizar um dos formatos.

Esta norma define ainda o endereço funcional e os endereços físicos de cada ECU, para cada formato de mensagem CAN – *Standard* ou *Extended*.

As mensagens com o endereço funcional no campo *Identifier* são recebidas por todas as ECU, funcionando assim como endereço de *broadcast*.

Todas as mensagens de pedido do equipamento de diagnóstico deverão ser enviadas com o endereço funcional no campo *Identifier*, são considerados portanto *functional requests*.

Todas as mensagens de resposta do veículo (*physical response*) deverão ser enviadas com o endereço físico de resposta no campo *Identifier*.

O endereço físico de pedido só deverá ser usado no envio de *Flow Control Frames* (apresentadas mais adiante neste documento) para o veículo (*physical request*).

2.4.2.1| CAN *Identifier* 11 bits

Num sistema OBD2 que utilize o protocolo CAN com *Identifier* de 11 bit, cada ECU tem um endereço para o qual o equipamento externo envia o pedido e um endereço com o qual responde aos pedidos. O segundo é composto pela soma do primeiro e 8.

Tabela 8 – Endereços OBD2 para CAN *Identifier* 11 bits

CAN *Identifier* (hex)	Descrição
7DF	Endereço funcional (*broadcast*)
7E0	Endereço Físico ECU #1 – Pedido
7E8	Endereço Físico ECU #1 – Resposta
7E1	Endereço Físico ECU #2 – Pedido
7E9	Endereço Físico ECU #2 – Resposta
7E2	Endereço Físico ECU #3 – Pedido
7EA	Endereço Físico ECU #3 – Resposta
7E3	Endereço Físico ECU #4 – Pedido
7EB	Endereço Físico ECU #4 – Resposta
7E4	Endereço Físico ECU #5 – Pedido

7EC	Endereço Físico ECU #5 – Resposta
7E5	Endereço Físico ECU #6 – Pedido
7ED	Endereço Físico ECU #6 – Resposta
7E6	Endereço Físico ECU #7 – Pedido
7EE	Endereço Físico ECU #7 – Resposta
7E7	Endereço Físico ECU #8 – Pedido
7EF	Endereço Físico ECU #8 – Resposta

2.4.2.2| CAN *Identifier* 29 bits

Os endereços utilizados nos sistemas OBD2 com CAN *Identifier* de 29-bit apresentam algumas diferenças.

Tabela 9 - Definição dos endereços em relação ao tipo de CAN *Identifier*

CAN *Identifier*	Endereço Destino (hex)	Endereço Origem (hex)	Tipo
Functional Request	Sistema OBD2 (Veículo) 33	Equip. Ext. Diagnóstico F1	Funcional
Physical Response	Equip. Ext. Diagnóstico F1	ECU ##	Físico
Physical Request	ECU ##	Equip. Ext. Diagnóstico F1	Físico
## - Endereço físico da ECU			

Tabela 10 - Formato do CAN *Identifier* de 29 bits - Endereçamento

CAN *Identifier* (bits)	28 – 24	23 – 16	15 – 8	7 – 0
Funcional (hex)	18	DB	End. Destino	End. Origem
Físico (hex)	18	DA	End. Destino	End. Origem

Tabela 11 - Endereços OBD2 para CAN *Identifier* 29 bits

CAN *Identifier*	Descrição
18 DB 33 F1	Endereço funcional para envio de mensagens de pedido do equipamento externo de diagnóstico
18 DA ## F1	Endereço físico de pedido do equipamento externo de diagnóstico para a ECU ##
18 DA F1 ##	Endereço físico de resposta da ECU ## para o equipamento externo de diagnóstico

2.4.3| Camada de Rede (Network Layer)

Nesta camada do modelo de referência OSI, a norma ISO 15765 disponibiliza métodos de fragmentação de mensagens e/ou assemblagem de pacotes.

Em alguns casos poderá ser necessário transmitir uma mensagem que ultrapasse o limite de bytes de dados de uma *frame* do protocolo CAN. Nesses casos esta norma faz a fragmentação dos dados em várias mensagens CAN e utiliza uma técnica de controlo de fluxo para garantir que nenhum fragmento da mensagem é perdido.

A Figura 13 apresenta o exemplo da transmissão de uma mensagem normal, não-segmentada.

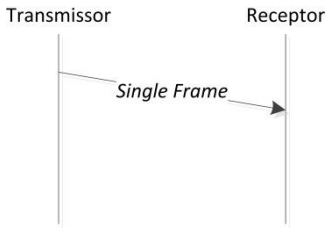

Figura 13 - Mensagem não segmentada

A Figura 14 apresenta o exemplo de uma mensagem segmentada e do mecanismo de controlo de fluxo implementado.

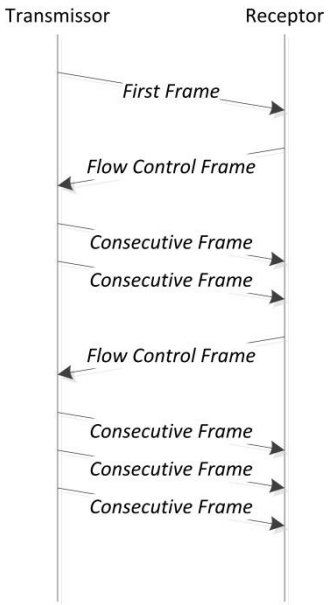

Figura 14 - Mensagem segmentada

A norma define então quatro tipos PDU (*Protocol Data Unit*) – ou unidade de dados de protocolo – que pode ser visto como o tipo de unidade de dados que esta camada envia para (e recebe de) camadas inferiores.

O campo de dados das mensagens CAN utilizado para transmitir dados estará limitado a 6/7 bytes, uma vez que este protocolo insere no início do mesmo o campo PCI (*Protocol Control Information*) como forma de identificar o tipo de *frame* e algumas informações necessárias para que seja possível a assemblagem da mensagem no receptor.

PDU	PCI	Descrição
Single Frame	0	Mensagem única. É enviada quando os dados a serem transmitidos ocupam no máximo 7 bytes.
First Frame	1	Primeira *frame*. Quando os dados a transmitir ocupam mais de 7 bytes, o protocolo faz a fragmentação dos mesmos e envia esta *frame* a sinalizar o receptor de se iniciou uma transmissão fragmentada.
Consecutive Frame	2	*Frame* consecutiva. É o tipo de *frame* que completa a mensagem fragmentada iniciada por uma *First Frame*.
Flow Control Frame	3	*Frame* de controlo de fluxo. Este tipo de frame é enviada pelo receptor para controlar o fluxo de informação de uma mensagem fragmentada.

O campo PCI pode ocupar até 3 bytes do campo de dados de uma mensagem CAN. A Tabela 12 representa a estrutura de cada campo PCI correspondente a cada tipo de PCU, e a forma como se posicionam no campo de dados das mensagens CAN.

Tabela 12 - Estrutura do campo de dados CAN correspondente aos bytes de PCI

		CAN *Data Field*			
		Byte 0		*Byte* 1	*Byte* 2
		Bits 7 - 4	Bits 3 - 0	Bits 7 - 0	Bits 7 - 0
PDU	*Single Frame*	PCI 0	Dimensão dos dados (nº de bytes adicionais) (1 – 7 bytes)	Dados	
	First Frame	PCI 1	Dimensão dos dados (nº de bytes adicionais) (7 – 4095 bytes)		Dados
	Consecutive Frame	PCI 2	Número de sequência (0 – 15)	Dados	
	Flow Control Frame	PCI 3	Flag de Estado/Controlo de Fluxo	Dimensão do Bloco (nº de *consecutive frames* a transmitir de seguida)	Tempo de Separação

É de salientar que o número de sequência – que identifica cada *Consecutive Frame* – não está limitado a 15, trata-se de um contador rotativo que reinicia a partir do 0 (zero) quanto atinge o valor máximo.

Tabela 13 – Definição de Flags de Estado/Controlo de Fluxo (*Flow Control Frame*)

Nome	Valor	Descrição
Continue to Send	0	Sinaliza o transmissor de que pode continuar a enviar *Consecutive Frames.*
Wait	1	Sinaliza o transmissor de que deve esperar por uma nova *Flow Control Frame* para continuar a transmissão de *Consecutive Frames.*
Overflow	2	Sinaliza o transmissor de que deve cancelar o envio de uma mensagem fragmentada. Só pode ser enviada na *Flow Control Frame* sucede a *First Frame* pois é usada quando o campo de dimensão de dados da mesma é superior À dimensão do *buffer* do receptor.

Tabela 14 - Definição dos valores da Dimensão do Bloco (*Flow Control Frames*)

Valor (hex)	Descrição
00	Informa o transmissor de quantas *Consecutive Frames* deve transmitir de seguida até ter que parar a transmissão e esperar por outra *Flow Control Frame*
01 – FF	Informa o transmissor de que deve enviar as restantes *Consecutive Frames* sem esperar por qualquer outra *Flow Control Frame*

Tabela 15 - Definição de Tempo de Espera (*Flow Control Frame*)

Valor (hex)	Descrição
00 – 7F	Valor absoluto em milissegundos (Ex: 0x0F = 15 ms)
80 – F0	Reservado
F1 – F9	Valor absoluto em centenas de microssegundos (Ex: 0xF3 = 300 µs)
FA – FF	Reservado

Caso seja enviada uma Flow Control Frame com um valor tempo de espera reservado, será considerado o maior valor especificado pela norma ISO 15765 (0x7F – 127 ms).

2.4.4| Camada de Sessão (*Session Layer*)

Nos dispositivos OBD2 é considerado início de sessão quando o equipamento externo de diagnóstico estabelece comunicação com o sistema OBD2 do veículo, e fica correctamente configurado. A sessão iniciada permanece activa enquanto a ECU estiver ligada.

2.5| ISO 15031 – Comunicação entre veículo e equipamento de diagnóstico

A camada Aplicação da comunicação via interface OBD2 é assegurada pela norma ISO 15031 que especifica os aspectos comunicação entre o veículo e equipamento externo de diagnóstico em termos de interpretação de dados e tempos de resposta.

Esta norma especifica também termos e definições que são utilizadas na comunicação – baseada em troca de mensagens – via OBD2.

É importante perceber que a comunicação não é feita directamente com os sensores ou actuadores, mas sim com uma ou várias ECU presentes no veículo – tipicamente uma.

A comunicação passa por uma troca de mensagens – de pedido e de resposta – em que é indicado o tipo de serviço e o parâmetro pretendido no caso de um pedido, e adicionalmente os valores do sensor correspondente ao parâmetro no caso de uma mensagem de resposta.

A norma define então:

- SID – *Service Identification* – Identificação do serviço solicitado, isto é, ler dados em tempo real ou ler os erros armazenados na ECU. A identificação do serviço consiste num número hexadecimal entre $01 e $09. (Exemplo: SID $01 – *Request current powertrain diagnostic data* – Pedido de dados actuais de diagnóstico).
- PID – *Parameter Identification* – Identificação dos parâmetros que se podem solicitar à ECU. Estes parâmetros são identificados por um número em hexadecimal entre $00 e $FF (Exemplo: PID $0D – *Vehicle Speed Sensor* – Sensor de Velocidade do Veículo).

As tabelas dos SIDs e PIDs disponíveis e as respectivas descrições encontram-se nos capítulos Anexo A, Anexo B e Anexo C.

2.5.1| Mensagem de Pedido – *Request Message*

As mensagens de pedido de informação – enviadas pelo equipamento externo de diagnóstico para o veículo – têm a estrutura apresentada na Figura 15.

CAN *Data Field*							
Byte 0	*Byte* 1	*Byte* 2	*Byte* 3	*Byte* 4	*Byte* 5	*Byte* 6	*Byte* 7
Número de *bytes* adicionais	SID	PID	*Bit Stuffing*	*Bit Stuffing*	*Bit Stuffing*	*Bit Stuffing*	*Bit Stuffing*

Figura 15 - Estrutura da mensagem de pedido

2.5.1.1| Mensagem de Início de Sessão

Esta norma define que a mensagem a ser enviada pelo equipamento externo de diagnóstico deve ser composta pelo SID $01 (*Request current powertrain diagnostic data*) e pelo PID $00 (*Supported PIDs ($01-$20)*), e que todas as ECUs deverão suportar pelo menos estes SID e PID.

2.5.2| Mensagem de Resposta – *Response Message*

As mensagens de resposta – enviadas pelo veículo para o equipamento externo de diagnóstico – têm a estrutura apresentada na Figura 16.

CAN *Data Field*							
Byte 0	*Byte* 1	*Byte* 2	*Byte* 3	*Byte* 4	*Byte* 5	*Byte* 6	*Byte* 7
Número de *bytes* adicionais	SID + 0x40	PID	*Data* A	*Data* B Opcional	*Data* C Opcional	*Data* D Opcional	*Bit Stuffing*

Figura 16 - Estrutura da mensagem de resposta

A mensagem de resposta poderá conter desde 1 até 4 bytes de informação, dependendo do PID em questão.

2.5.3| Tempo Máximo de Resposta

A norma ISO 15031 define um limite máximo de tempo de resposta por parte de um veículo a um pedido do equipamento externo de diagnóstico, após o qual pode fazer um novo pedido. Desta forma evita-se a espera por tempo indeterminado por parte do mesmo pela resposta a um pedido de leitura de um PID não suportado.

Define-se então como limite máximo de tempo de resposta – ou tempo máximo entre o envio de uma mensagem de pedido e a recepção de uma mensagem de resposta – o parâmetro $P2_{CAN}$ = 50 ms.

2.5.4| Conector OBD2 – SAE J1962

Durante vários anos os conectores utilizados nos veículos para comunicação OBD assumiram várias formas, algumas específicas para cada fabricante.

No entanto com toda a normalização que tem verificou nos últimos anos, o conector passou a ser também especificado. Esta norma define portanto que o conector utilizado num veículo compatível com a norma OBD2 deverá ser conector o SAE J1962.

Cada protocolo de comunicação legislado utiliza pinos específicos no conector SAE J1962.

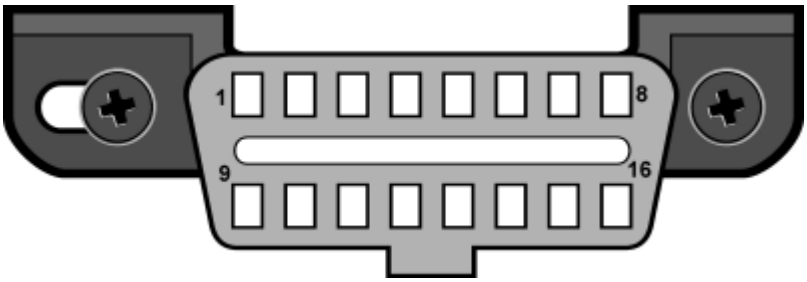

Figura 17 - Conector SAE J1962 fêmea (automóvel)

Tabela 16 - Descrição dos pinos do conector SAE J1962

Pino	Descrição
1	Não definido
2	SAE J1850 (+)
3	Não definido
4	Massa (Chassis)
5	Massa (Sinal)
6	ISO 15765 CAN-H
7	Linha K – ISO 9141 e ISO 14230
8	Não definido
9	Não definido
10	SAE J1850 (-)
11	Não definido

12	Não definido
13	Não definido
14	ISO 15765 CAN-L
15	Linha L – ISO 9141 e ISO 14230
16	+12 V

A versão do conector presente nos veículos é a versão fêmea (Figura 18). Nos equipamentos externos de diagnóstico é utilizado o conector macho (Figura 19).

Figura 18 - Conector SAE J1962 fêmea

Figura 19 - Conector SAE J1962 macho

2.6| Protocolo SPI

O protocolo SPI (*Serial Peripheral Interface*) tornou-se muito popular na comunicação série e é suportado por *hardware* em muitos microcontroladores utilizados no desenvolvimento de sistemas embebidos.

As entidades presentes numa comunicação SPI são *Master* – o dispositivo que fornece sinal de relógio de sincronismo e que controla portanto a comunicação – e um ou mais *Slaves*.

Algumas das características-chave do protocolo SPI são:

- *Master-Slave*
- *Multi-Slave*
- *Full-duplex*

2.6.1| Interface

A interface deste protocolo é tipicamente constituída por 4 ligações, conforme apresenta a Figura 20.

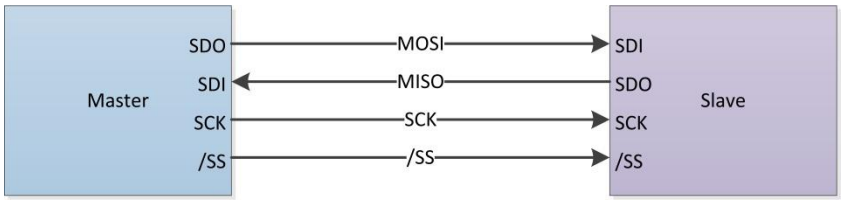

Figura 20 - Esquema de ligação SPI

Da Figura 20 podem identificar-se 4 pinos:

- **SDO** – *Serial Data Out* – Porta de transmissão de dados
- **SDI** – *Serial Data In* – Porta de recepção de dados
- **SCK** – *Serial Clock* – Sinal de relógio de sincronismo
- **SS** – *Slave Select* – Selecção do dispositivo *Slave*

Dadas as descrições acima podem ser definidas as seguintes ligações:

- **MOSI** – *Master Out Slave In* – Ligação que envia informação do *Master* para o *Slave*

- **MISO** – *Master In Slave Out* – Ligação que envia informação do *Slave* para o *Master*
- **SCK** – *Serial Clock* – Ligação através da qual o *Master* fornece sinal de sincronismo ao *Slave*
- **SS** – *Slave Select* – Ligação através da qual o *Master* selecciona o *Slave* com o qual quer comunicar

Os bits são enviados sequencialmente em cada linha – MOSI, MISO ou ambas - em sincronismo com o sinal de relógio. O facto de ser possível a transmissão de dados em simultâneo nas duas linhas confere ao protocolo a capacidade de *full-duplex*.

Em algumas situações, o protocolo SPI pode funcionar com apenas 2 ou 3 ligações.

Quando existe apenas um dispositivo *Slave*, o pino /SS do *Slave* pode ser ligado à massa (lógica activa baixa), ficando este sempre seleccionado e pronto a comunicar, sem necessidade de estar ligado ao pino /SS do *Master*. Neste caso a comunicação SPI funcionará com apenas 3 ligações.

Adicionalmente, caso se trate de comunicação unidireccional com apenas um Slave – por exemplo apenas o Master e envia informação para o Slave – esta poderá ser feita com apenas 2 ligações, MOSI e SCK, assumindo a situação do parágrafo anterior.

Quando este protocolo é utilizado com vários Slaves, o Master deverá ter uma linha SS para cada Slave e activar a respectiva linha para comunicar com cada um.

A Figura 21 apresenta o exemplo de uma ligação SPI entre um Master e três Slaves.

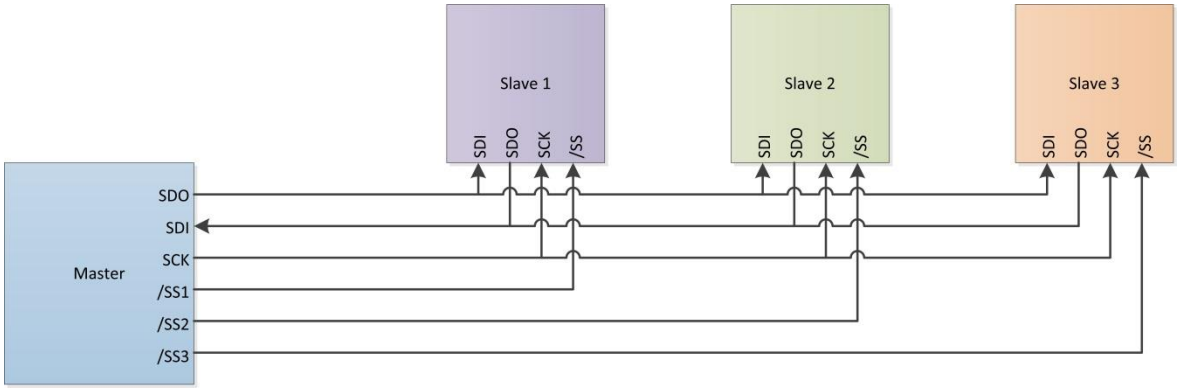

Figura 21 - Esquema de ligação SPI com múltiplos *slaves*

3| Trabalho desenvolvido – Sistema OBD2/EOBD Reader para Veículos Automóveis

O sistema desenvolvido tem como principal função a leitura de dados em tempo real a partir da porta OBD do veículo.

Este dispositivo suporta apenas veículos com CAN uma vez que se tornou o protocolo principal na implementação dos sistemas de diagnóstico automóvel.

O objectivo deste sistema é permitir o acesso por parte do utilizador, a informações que não estão normalmente disponíveis no quadrante do veículo – ou em qualquer outro dispositivo a bordo – mas que poderão ser importantes na verificação dos sistemas do veículo ou até na despistagem e detecção de avarias.

3.1| Características

Protocolo de Comunicação: CAN 2.0

Alimentação: Auto-alimentado pela porta OBD (12V)

Visualização: LCD retro-iluminado 128x64

Controlo/Navegação: Teclado de 5 botões

3.2| Hardware

Este sistema é essencialmente composto por um microcontrolador Microchip™ PIC 18F46K22, um controlador CAN Microchip™ MCP2515, um *transceiver* CAN Microchip™ MCP2551, um ecrã LCD gráfico e um teclado de 5 botões.

Todo o *hardware* é apresentado em maior detalhe nos capítulos seguintes deste documento.

Os esquemas eléctricos e respectivas PCBs das partes que compõem o sistema desenvolvido encontram-se no Anexo D deste documento.

O diagrama da Figura 22 apresenta uma visão geral sobre a arquitectura do sistema desenvolvido.

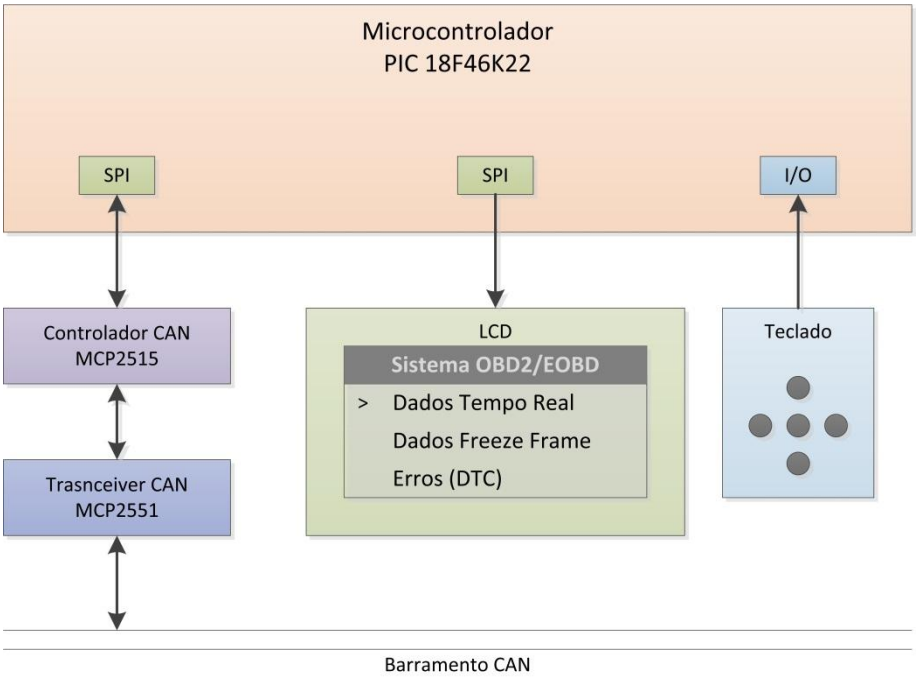

Figura 22 - Diagrama da arquitectura do sistema OBD2/EOBD

3.2.1| Microcontrolador: Microchip™ PIC 18F46K22

O microcontrolador utilizado na implementação deste sistema é o Microchip™ PIC 18F46K22, um microcontrolador de 8-bits de alta performance – que se mostrou adequado para as operações efectuadas – do qual destaco as seguintes características:

- 3896 *Bytes* de SRAM
- 64 *KBytes* de *Program Memory* (*Flash*)
- 2 SPI
- 3 Portas de Interrupção por H/W
- *Timers* 8-bits e 16-bits
- Oscilador Interno 16MHz
- ICSP (*In-Circuit Serial Programming*)

A escolha do microcontrolador recaiu sobre este dispositivo principalmente devido à familiarização com os dispositivos da Microchip™, aos periféricos que este modelo oferece e à memória disponível. Apesar de existirem modelos da mesma família com módulo ECAN integrado, optei por utilizar um controlador CAN externo, também da Microchip™, o MCP2515 descrito mais adiante neste documento, por uma questão de modularidade do sistema.

É também um dispositivo bastante económico – abaixo dos 2 € – tendo em conta as funcionalidades que oferece.

Para efeitos de prototipagem foi utilizado o encapsulamento PDIP de 40 pinos (Figura 23) e a assemblagem da versão final foi feita com a versão TQFP de 44 pinos (Figura 24).

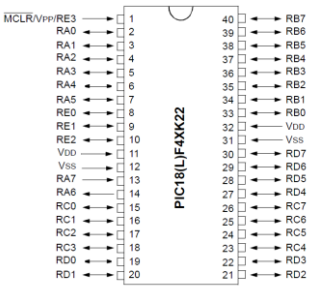

Figura 23 - PIC 18F46K22 PDIP

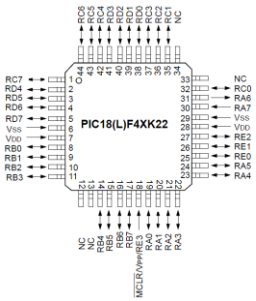

Figura 24 - PIC 18F46K22 TQFP

3.2.1.1| Memória

A utilização intensiva de *strings* de texto exige bastante memória neste tipo de dispositivos, sendo este outro motivo da escolha deste modelo, visto ter 3896 *Bytes* de memória SRAM e 64 *KBytes* de *Program Memory*.

3.2.1.2| SPI

As ligações ao controlador CAN e ao LCD são feitas com protocolo SPI, criando assim a necessidade de existência de dois módulos SPI. Este protocolo permite a comunicação alternada com vários dispositivos ligados ao mesmo módulo utilizado as portas SS (*Slave Select*) de cada dispositivo *slave* – conforme descrito anteriormente neste documento – no entanto por limitações do controlador do LCD não foi possível projectar o sistema dessa forma, já que este não permite que haja variações nas linhas MOSI (*Master Out Slave In*) e SCLK (*Serial Clock*) quando o respectivo SS está inactivo.

3.2.1.3| Oscilador

O oscilador interno deste microcontrolador permite que este funcione com um mínimo de componentes externos, e evitando a utilização osciladores de quartzo externos.

O oscilador interno suporta velocidades até 16 MHZ sem utilização da funcionalidade PLL, que se mostrou suficiente para o desenvolvimento deste projecto. Desta forma trabalha também à mesma frequência que o controlador CAN.

3.2.1.4| Temporizadores – *Timer0* e *Timer1*

Este microcontrolador disponibiliza vários temporizadores, dos quais são utilizados o *Timer0* (16 bits) e o *Timer1* (8 bits).

O primeiro tem como função controlar a frequência de actualização do ecrã na listagem de valores lidos em tempo real. O segundo controla o tempo de resposta da ECU verificando se este ultrapassa o limite definido pela norma ISO 15031 ($P2_{CAN}$), estabelecido 50 ms.

3.2.1.5| Interrupções

Dos 3 pinos de interrupção por *hardware* disponibilizados por este microcontrolador, são utilizados 2.

O pino INT1 tem como função detectar as interrupções originadas pelo controlador CAN. O pino INT2 detecta as acções do teclado, isto é, a cada botão pressionado é gerada uma interrupção em INT2.

3.2.1.6| IDE e Compilador

Para o desenvolvimento do código fonte deste sistema, foram utilizados o ambiente de programação MPLAB IDE™ v8.76 juntamente com o compilador MPLAB C18 Compiler™ – versão 3.40 variante académica – ambos da Microchip™. O MPLAB C18 Compiler™ é o compilador disponibilizado pelo fabricante para compilar código para os dispositivos da família 18F.

A Figura 25 apresenta a janela do MPLAB IDE™ v8.76.

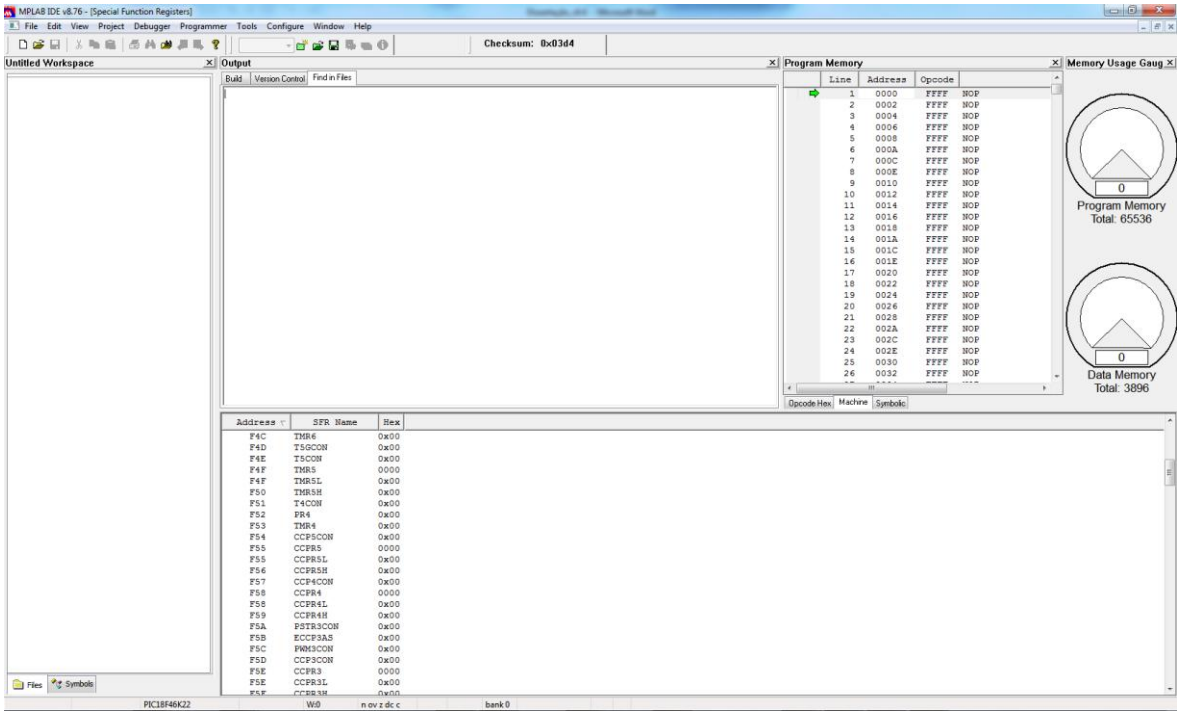

Figura 25 - Janela do MPLAB IDE™ v8.76

3.2.1.7| ICSP

O microcontrolador utilizado tem possibilidade de programação no circuito (ICSP) permitindo assim eventuais actualizações de *firmware*.

Para efectuar o a programação do microcontrolador foi utilizado o dispositivo Microchip PICkit™ 3, apresentado na Figura 26.

Além de programador compatível com quase todos os modelos de microcontroladores da Microchip™, apresenta funcionalidades de *debug* para melhor detecção e correcção de erros de programação, possibilidade de alimentação do circuito – pode programar microcontroladores sem uma fonte de alimentação externa – e ainda a função *Programmer-to-Go* em que o ficheiro a carregar no microcontrolador é armazenado na memória (do programador) para que seja possível fazer o seu carregamento para o microcontrolador sem necessidade de um computador.

Figura 26 – Programador PICkit 3

3.2.2| Controlador CAN: Microchip™ MCP2515

Tal como referido anteriormente neste documento, a opção de utilizar um controlador CAN externo ao microcontrolador, ao invés de utilizar um microcontrolador com módulo CAN integrado, deve-se ao facto de se pretender criar um determinado nível de modularidade no sistema.

Destaco das características deste controlador:

- Compatibilidade com CAN 2.0B até 1Mbit/s
- 2 *Buffers* de Recepção com Filtros e Máscaras programáveis de 29 bits
- 3 *Buffers* de Transmissão
- Interface SPI
- Porta de Interrupção

Para efeitos de prototipagem foi utilizada a versão encapsulada em SPDIP (18 pinos), e para assemblagem da versão final a versão SOIC 300mil (18 pinos). A disposição dos pinos (de ambas as versões) está apresentada na Figura 27.

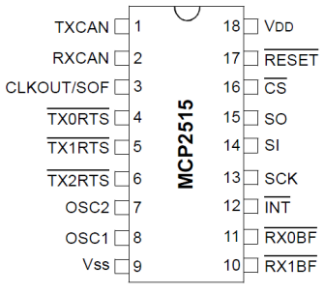

Figura 27 - Controlador CAN MCP2515 SPDIP/SOIC

3.2.2.1| Compatibilidade

Este módulo é compatível com a última especificação do protocolo CAN, e permite comunicação até 1Mbit/s, que se mostra mais do que suficiente para este sistema, dado que os sistemas de diagnóstico automóvel funcionam tipicamente a 125Kbit/s, 250Kbit/s ou 500Kbit/s.

A compatibilidade com a última revisão do protocolo prevê ainda que seja compatível com ambos os formatos de mensagens CAN, *Standard Frame* e *Extended Frame*. É também compatível com os quatro tipos de mensagem: *Data Frame*, *Remote Frame*, *Error Frame* e *Overload Frame*, sendo que as duas primeiras são da

responsabilidade do programador, e as duas últimas da responsabilidade do dispositivo de detecção de erros, conforme descrito mais adiante neste documento.

3.2.2.2| Recepção de Mensagens

Os dois *buffers* de recepção com filtros e máscaras programáveis de 29 bits demonstram uma elevada utilidade para este dispositivo, uma vez que permitem que depois de correctamente programados, apenas sejam aceites mensagens que respeitem determinados parâmetros.

O *buffer* RX0 permite a configuração de uma máscara e dois filtros, o *buffer* RX1 permite a configuração de uma máscara e quatro filtros.

No caso das mensagens do tipo *Extended* os filtros e máscaras são aplicados ao campo *Identifier* (29 bits), no caso das mensagens do tipo *Standard*, estes são aplicados ao campo *Identifier* (11 bits) e aos 16 bits correspondentes aos primeiros dois *bytes* do campo de dados (*Data Field*), *Data 0* e *Data 1* diminuindo assim a carga do CPU na filtragem de mensagens.

Na realidade existe um terceiro *buffer* de recepção, o MAB (*Message Assembly Buffer*), que recebe inicialmente todas as mensagens do barramento. Depois de comparadas com as máscaras e filtros dos *buffers* RX0 e RX1, as mensagens são transferidas para um destes. Caso sejam aceites pelos filtros e máscaras de ambos os *buffers* (RX0 e RX1), o *buffer* RX0 tem sempre prioridade mais alta. O facto de este ter apenas dois filtros de recepção torna-o mais restritivo, atribui-lhe portanto uma maior prioridade.

Uma funcionalidade adicional é que o *buffer* RX1 poder ser usado para *rollover*, isto é, se configurado como tal, caso o *buffer* RX0 tenha uma mensagem válida e chegue uma nova mensagem ao MAB que seja igualmente válida para os filtros e máscara do *buffer* RX0, esta será transferida para o *buffer* RX1, mesmo que não respeite a máscara e os filtros deste, evitando assim um erro de *overflow*.

Quando uma mensagem chega ao MAB é comparada com a máscara do *buffer* RX0, e todos os bits a ‘0’ representam as posições dos bits que serão ignorados posteriormente na comparação com os filtros, os bits a ‘1’ representam as posições dos bits que serão verificados. De seguida os bits a verificar são comparados com os filtros e

apenas serão aceites mensagens cujo campo a verificar tenha correspondência com pelo menos um dos filtros.

Tabela 17 - Tabela de verdade de filtros e máscaras

Máscara Bit n	Filtro Bit n	Identificador Bit n	Aceitação
0	X	X	Aceite
1	0	0	Aceite
1	0	1	Rejeitado
1	1	0	Rejeitado
1	1	1	Aceite

3.2.2.3| Transmissão de Mensagens

Como já referido anteriormente neste documento, as mensagens CAN têm um sistema de arbitragem implícito, em que a mensagem com menor valor no campo Identifier tem maior prioridade.

Este controlador permite a atribuição de prioridade a cada *buffer* de transmissão numa escala de 4 níveis (2 bits), independentemente da prioridade implícita no campo *Identifier*.

Antes de qualquer *buffer* transmitir a mensagem que contém para o barramento, todas as prioridades são comparadas, e são depois enviadas as mensagens pela respectiva ordem de prioridade. No caso de dois ou mais *buffers* terem o mesmo nível de prioridade, ordem de transmissão passa a ser definida pelo número do *buffer*, isto é, o *buffer* TX2 tem maior prioridade, o *buffer* TX0 tem menor prioridade.

3.2.2.4| Bit Timing

Apesar da liberdade de escolha de velocidade de comunicação e de características do barramento que o protocolo CAN oferece, todos os nós têm que respeitar essas características estabelecidas no desenvolvimento do sistema. Como tal, este controlador CAN tem que ser configurado de acordo com as normas estabelecidas pela especificação OBDII/EOBD, mais concretamente na questão da velocidade de comunicação, no que toca a este capítulo.

Devido ao facto de o protocolo CAN utilizar codificação NRZ (*Non-Return to Zero*), o sinal de relógio não é codificado nos dados, e como tal deve ser utilizado um outro

mecanismo de recuperação de sinal de relógio para garantir o sincronismo entre transmissor e receptor.

De forma a contornar este problema, o controlador CAN MCP2515 tem um sistema de sincronismo denominado de DPLL (*Digital Phase Lock Loop*). Este sistema é sincroniza-se a partir das tramas recebidas, bem como fornece sinal de sincronismo quando transmite mensagens.

Dado ser utilizada a codificação NRZ, e para que este sistema funcione devidamente, é necessário que seja introduzido *bit-stuffing* nas tramas de forma a garantir que não existem mais de seis bits consecutivos sem haver transição no barramento, pois são as transições que permitem ao sistema DPLL manter o sincronismo do módulo.

É ainda importante dizer que a velocidade de comunicação (*baud rate*) não limita a frequência de relógio do controlador CAN a um valor fixo, isto é, o dispositivo responsável pelo sincronismo do controlador pode ser programado para funcionar a um determinado *baud rate* com diversas frequências de oscilador principal, permitindo também o inverso, a configuração de diversas velocidades de comunicação com o mesmo oscilador principal. Esta característica do controlador permite colmatar a questão das diferentes velocidades de comunicação utilizadas na comunicação CAN no protocolo OBDII/EOBD.

A escolha do oscilador recaiu sobre um cristal de quartzo de 16 Mhz. Esta velocidade é o valor mais adequado para que se torne possível configurar os *baud rates* utilizados no protocolo OBD sem alteração do oscilador.

Uma particularidade deste módulo CAN em relação ao *Sample Point* é que permite que a amostragem seja feita uma única vez por bit (conforme referido na especificação CAN 2.0B), ou três vezes por bit, sendo depois escolhido o nível lógico com maior número de amostragens. Nesse caso, as duas amostragens adicionais são feitas antes do final de PS1 em intervalos de meio TQ.

De seguida é explicado o processo de cálculo e configuração do mecanismo de sincronismo deste controlador CAN.

Defina-se o *Nominal Bit Rate* (NBR), *Nominal Bit Time* (NBT) e a sua relação:

$$NBR = f_{bit}$$

$$NBT = t_{bit}$$

$$f_{bit} = \frac{1}{t_{bit}}$$

Correspondem a velocidade de comunicação e período de um bit, respectivamente.

O funcionamento deste sistema parte de uma divisão do período de um bit (t_{bit}) em múltiplos segmentos de igual período, aos quais se dá o nome de *Time Quanta* (TQ). Esta divisão é feita através da programação de um *prescaler* (BRP) de forma a se respeitar a equação mais adiante.

A Figura 28 representa a relação entre t_{osc}, TQ e t_{bit}.

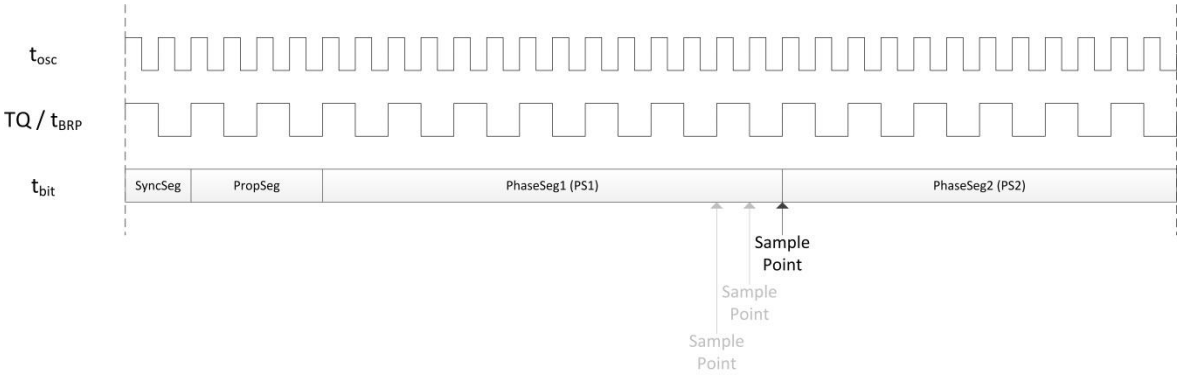

Figura 28 - Diagrama temporal - t_{osc}, TQ e t_{bit}

Para configurar este dispositivo é necessário ter em conta a seguinte equação.

$$TQ = 2 \cdot BRP \cdot t_{OSC} = \frac{2 \cdot BRP}{f_{OSC}}$$

E ainda as seguinte condições:

$$PropSeg + PS1 \geq PS2$$

$$PropSeg + PS1 \geq t_{DELAY}$$

$$PS2 > SJW$$

Considere-se então que se pretende configurar o controlador para comunicar a uma velocidade de 500 Kbit/s (500 KHz), assumindo a utilização de um oscilador de 16 MHz.

$$f_{OSC} = 16 \times 10^6\ Hz$$

$$t_{OSC} = 62{,}5\ \times 10^{-9}\ s$$

$$f_{bit} = 500 \times 10^3\ Hz$$

$$t_{bit} = 2\ \times 10^{-6}\ s$$

Para se poder dividir cada segmento do período de um bit em vários TQ, consideremos que este será composto por 16TQ.

$$t_{bit} = 16\ \cdot\ 2\ \cdot BRP\ \cdot\ t_{OSC}$$

$$BRP = \frac{t_{bit}}{32\ \cdot\ t_{OSC}} = 1$$

$$BRP_{500} = \frac{2\ \times 10^{-6}}{32\ \cdot\ 62{,}5\ \times 10^{-9}} = 1$$

A partir da equação anterior é possível calcular o valor de BRP para a outra velocidade – 250 Kbit/s – já que o seu valor é divisor de 500 Kbit/s.

$$f_{bit\,250} = 250 \times 10^3\ Hz$$

$$t_{bit\,250} = 4\ \times 10^{-6}\ s$$

$$BRP_{250} = \frac{4\ \times 10^{-6}}{32\ \cdot\ 62{,}5\ \times 10^{-9}} = 2$$

Resumindo os valores calculados:

Tabela 18 - Baud Rate Prescalers

Baud Rate (Kbit/s)	BRP
250	2
500	1

De acordo com a norma ISO 11898, o *Sample Point* deverá posicionar-se a cerca 80% do período de um bit.

Desta foram aribuindo 1 TQ ao *SyncSeg* (definido pela especificação do protocolo), 2TQ ao *PropSeg* e 10 TQ ao PS1, o *Sample Point* – que ocorre após o segmento PS1 – fica situado após o 13º TQ que corresponde a 81,25% do período de um bit.

Os restantes 3 TQ são atribuídos ao segmento PS2.

Resta portanto calcular o valor de SJW, que pode ser qualquer valor da gama permitida (1 a 4 TQ) e neste caso a opção recaiu sobre 1 TQ. Os testes efectuados revelaram estabilidade e portanto não houve necessidade de aumentar este valor.

Desta forma, os segmentos que compõem os períodos de um bit terão as durações (em TQ) da Tabela 19.

Tabela 19 - Duração dos segmentos de t_{bit} (TQ)

Segmento	Duração (TQ)
SyncSeg	1
PropSeg	2
PS1	10
PS2	3
SJW	1

3.2.3| Transceiver CAN: Microchip™ MCP2551

A camada física do protocolo CAN é gerida pelo *transceiver* CAN Microchip™ MCP2551. Este dispositivo suporta velocidades até ao valor máximo especificado pelo protocolo (1 Mbit/s) e é totalmente compatível com a norma ISO 11898.

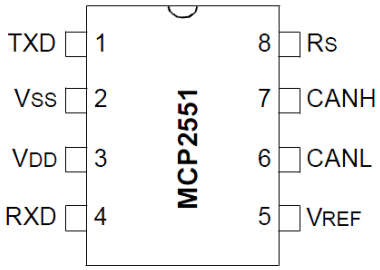

Figura 29 - Transceiver CAN MCP2551

Na Figura 29 está a disposição dos pinos do circuito integrado, cuja descrição se encontra na tabela seguinte.

Para efeitos de prototipagem foi utilizada a versão encapsulada em PDIP (8 pinos), e para assemblagem da versão final a versão SOIC 150mil (8 pinos).

Tabela 20 - Descrição dos pinos do MCP2551

Pino	Descrição
TXD	Entrada de Dados a Transmitir
VSS	Massa
VDD	Tensão de alimentação
RXD	Saída de Dados Recebidos
VREF	Tensão de Referência (Saída)
CANL	CAN-Low (barramento CAN)
CANH	CAN-High (barramento CAN)
RS	Slope-Control

O MCP2551 tem como função converter os dados recebidos do controlador CAN no pino RXD e transmiti-los para o barramento CAN pelos pinos CANH e CANL. Os dados recebidos nos pinos CANH e CANL serão enviados pelo pino RXD para controlador CAN.

Este *transceiver* MCP2551 permite a selecção do seu modo de operação de entre 3 disponíveis:

- *High-Speed*
- *Slope Control*
- *Standby*

O modo *High-Speed* é o que suporta comunicação à velocidade máxima. É seleccionado ligando o pino RS ao pino VSS.

Slope-Control é o modo em que se limitam as velocidades de transição de nível nos *drivers* de saída – CANH e CANHL – diminuindo assim eventuais interferências electromagnéticas (IEM). Este modo é seleccionado ligando uma resistência entre o pino RS e o pino VSS. O declive da transição é proporcional ao valor da corrente eléctrica que flui na resistência.

O modo *Standby* – ou modo *Sleep* – é seleccionado ligando o pino RS ao pino VDD. Neste cenário não existem comunicação nos *drivers* de saída CANH e CANL.

Neste sistema o *tranceiver* MCP2551 está permanentemente no modo *High-Speed*. Vários testes foram efectuados e não se demonstraram problemas de estabilidade devido a IEM.

3.2.4| Teclado

O sistema desenvolvido é controlado por um teclado de 5 botões, que na realidade são 5 botões de pressão com 5 resistências de *pull-down*, conectados a portas I/O do microcontrolador.

Adicionalmente existem 5 díodos – cada um ligado a um botão – conectados a um pino de interrupção por *hardware* do microcontrolador, cujo objectivo é sinalizar o microcontrolador de cada vez que é pressionado um botão. Desta forma é eliminada a necessidade de *polling* ao teclado, optimizando assim o tempo de processamento.

A Figura 30 mostra a disposição dos vários botões do teclado.

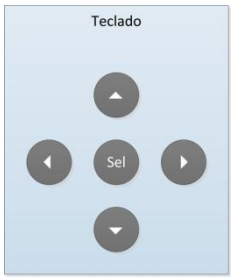

Figura 30 - Diagrama do teclado

3.2.5| LCD

O visor do sistema desenvolvido é um LCD gráfico com resolução 128x64 píxeis.

O modelo utilizado tem um controlador Sitronix ST7565P com interface série e paralela.

A interface série foi a escolhida para a comunicação com o microcontrolador, através de uma das portas SPI disponíveis. Esta interface torna-se menos complexa em termos de ligações físicas.

3.2.6| Ligação ao Veículo

A ligação ao veículo é feita através do conector *standard* SAE J1962.

Uma vez que este dispositivo é direccionado a veículos compatíveis com as normas ISO 11898 e ISO 15765, isto é protocolo CAN, apenas são utilizados 4 contactos deste conector.

A Figura 31 representa os pinos utilizados e a sua descrição.

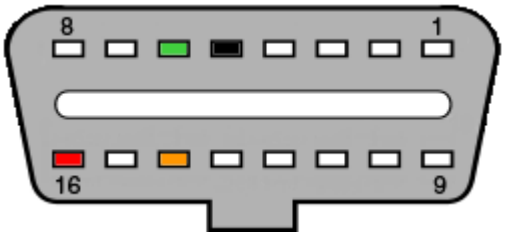

Figura 31 - Conector SAE J1962

Massa (GND)
CAN-H
CAN-L
+12 V

3.3| Software (Microcontrolador)

O compilador Microchip™ C18 Compiler disponibiliza uma biblioteca de funções para comunicação com o controlador CAN utilizado neste sistema. Algumas funções foram optimizadas para se ajustarem da melhor forma ao funcionamento deste sistema.

A biblioteca de funções relacionadas com o ecrã LCD foi desenvolvida juntamente com o *software* deste sistema OBD2/EOBD.

O cálculo e a representação de números em vírgula flutuante é normalmente um problema destes microcontroladores, já que é bastante exigente do ponto de vista de processamento.

Para minimizar o esforço dos cálculos é utilizado o método do ponto fixo que se torna bastante menos exigente para este tipo de microcontroladores.

Este método consiste em fazer todos os cálculos multiplicando o valor por um factor de equivalente ao grau número de casas decimais pretendidas, isto é, para 2 casas decimais é utilizado um factor de 100.

No final dos cálculos, a representação dos valores é feita imprimindo a divisão do resultado inicial pelo factor utilizado (parte inteira), de seguida o ponto ou vírgula e finalmente a parte decimal, que corresponde à subtracção do produto da parte inteira com o factor utilizado ao resultado inicialmente calculado.

Por exemplo, para representar o valor do sensor de massa de ar medindo um fluxo de 872,15 gramas por segundo.

$$res_{inicial} = 87215 \qquad factor = 100$$

$$parte_{inteira} = \frac{res_{inicial}}{factor} = \frac{87215}{100} = 872 \quad parte_{deciaml} = res_{inicial} - (872 \times 100) = 15$$

$$resultado = 872{,}15\ g/s$$

3.4| Funcionamento do Sistema

Os PIDs (*Parameter Identification*) definidos pela norma ISO 15031 estão listados na memória do sistema sob a forma de um *array*, estando em cada posição do mesmo, o código de cada PID em hexadecimal.

Estão também armazenadas sob a forma de *arrays* a lista de identificações – ou etiquetas – dos PIDs e as unidades (métrica) dos valores devolvidos, de forma a poderem ser identificados pelo utilizador, estando estas nos índices – ou posições do *array* – correspondentes aos respectivos PIDs.

Existem ainda um outro *array* que funciona como mapa de bits de PIDs suportados, que é inicializado a '0' (zero) mas que é depois devidamente alterado durante o processo de inicialização.

O tempo de resposta do veículo às mensagens de pedido por parte do equipamento de diagnóstico é monitorizado através de um temporizador (*Timer1*).

3.4.1| Inicialização

Devido ao facto de serem utilizadas diferentes velocidades de comunicação CAN nos actuais sistemas de diagnóstico automóvel, é necessário verificar qual a velocidade utilizada por cada veículo em que o sistema vai ser utilizado.

Adicionalmente é necessário verificar se o veículo suporta mensagens CAN do formato *Standard* ou *Extended*.

Outra variável dos sistemas de diagnóstico automóvel, é o número de PIDs que cada veículo suporta, isto é, quais os PIDs a que é possível aceder e ler o respectivo valor.

A fase de inicialização automática do sistema visa precisamente determinar essas variáveis de forma a configurar devidamente o controlador CAN e a lista de PIDs disponíveis no menu correspondente do sistema de diagnóstico OBD2/EOBD.

O processo começa pela configuração do controlador CAN com um dos *baud rates* estabelecidos pela norma, 500 Kbit/s.

De seguida é enviada uma mensagem de inicialização no formato Standard com o SID $01 e PID $00. Em caso de sucesso na transmissão é inicializado o *Timer1*.

Se no final da contagem do *Timer1* não tiver sido recebida uma resposta por parte do veículo, é reenviada a mensagem de inicialização no formato *Extended*, e reinicializado o *Timer1*. Caso seja recebida uma resposta – quer à mensagem Standard quer à Extended – o sistema passa à próxima fase do processo de inicialização.

Caso são seja recebida uma resposta à mensagem *Extended* ou se verifique um erro de transmissão em qualquer das mensagens – *Standard* ou *Extended* – é feita a reconfiguração do controlador CAN para o *baud rate* alternativo, 250 Kbit/s.

De seguida o processo anterior é repetido e em caso de erro de transmissão pode concluir-se que o veículo não é compatível com a norma ISO 15765, e portanto não é possível comunicar com o mesmo a partir deste sistema. Ou poderá existir alguma falha ou avaria no sistema de diagnóstico do veículo impossibilitando a comunicação com o mesmo. Neste cenário o sistema OBD2/EOBD deverá mostrar uma mensagem no ecrã a informar o utilizador de que é impossível comunicar com o veículo.

Caso haja uma resposta a uma das mensagens de inicialização – *Standard* ou *Extended* – com qualquer um dos *baud rates* configurados – 500 Kbit/s ou 250 Kbit/s – o sistema guarda a informação sobre o formato de mensagens suportado pelo veículo, não sendo necessário que o mesmo seja feito em relação ao *baud rate* utilizado uma vez que este permanecerá inalterado até uma reinicialização do sistema OBD2/EOBD.

Após a fase de inicialização do sistema relacionada com o protocolo CAN, é apresentada uma mensagem ao utilizador indicando o formato CAN utilizado – *Standard* (11-bit) ou *Extended* (29-bit) – e o *baud rate* de comunicação.

Figura 32 - Mensagem de informação do sistema: Formato CAN e *baud rate* de comunicação

Em caso de insucesso, é exibida uma mensagem a indicar ao utilizador que o veículo não é suportado – não é compatível com o protocolo CAN – ou a ignição poderá desligada.

A fase seguinte deste processo de inicialização passa por enviar novas mensagens de pedido com o SID $01 e com os PIDs de informação sobre os PIDs suportados. Desta forma o mapa de bits de PIDs suportados é devidamente preenchido.

De seguida o sistema fará a configuração automática dos menus. Esta fase é feita com a informação armazenada no mapa de bits referido no parágrafo anterior, colocando as respectivas etiquetas de cada PID no menu Lista de PIDs, evitando assim a listagem de PIDs não suportados no menu que dificultariam a navegação no mesmo, e induziriam o utilizador em erro.

Durante todo este processo o ecrã do dispositivo apresentará apenas uma mensagem a informar o utilizador de que o sistema se encontra em processo de inicialização, uma vez que a norma ISO 15765 especifica que não é permitida qualquer intervenção no sistema por parte do utilizador durante este processo.

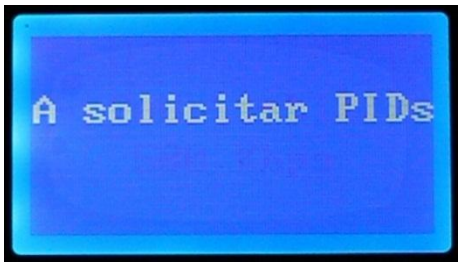

Figura 33 - Mensagem de informação de estado do sistema: A solicitar PIDs

Após a solicitação de PIDs à ECU do veículo, o utilizador é informado sobre o número de PIDs suportados pelo veículo, conforme apresentado na Figura 34.

Figura 34 - Mensagem de informação do sistema: PIDs suportados pelo veículo

Concluído este processo de inicialização o sistema passará ao Menu Principal, apresentado na Figura 36.

O diagrama da Figura 35 demonstra o decorrer da fase de inicialização do sistema.

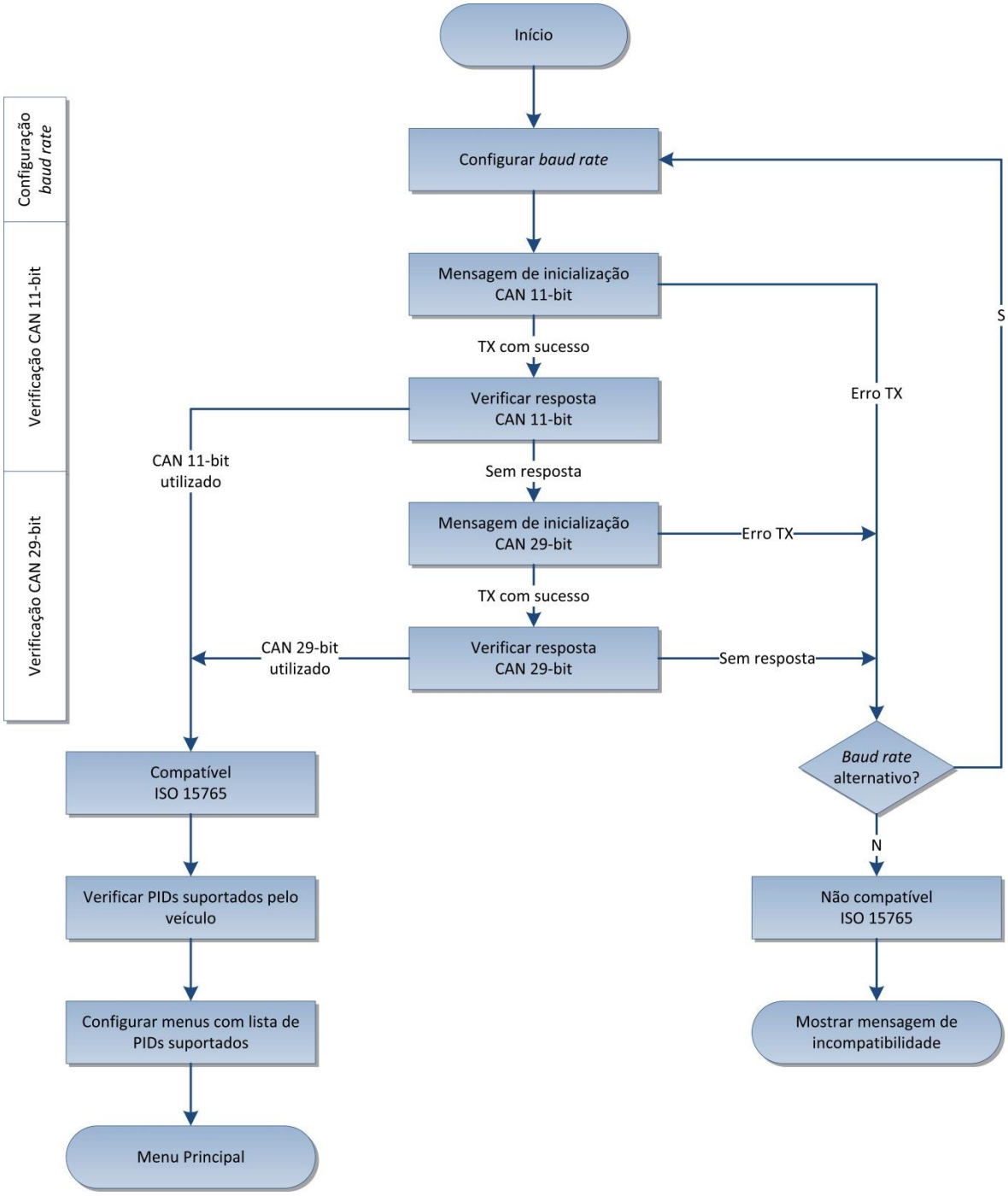

Figura 35 - Diagrama do processo de inicialização do sistema desenvolvido

3.4.2| Menu

3.4.2.1| Estrutura

O menu apresentado no ecrã do sistema é dividido em 3 níveis.

O primeiro nível é o Menu Principal onde se pode optar por ler dados de sensores em tempo real, obter dados *freeze frame* ou ler possíveis erros registados na unidade de comando do veículo.

O Menu Principal está representado na Figura 36.

Figura 36 - Menu Principal

Seleccionando a leitura de dados em tempo real, passa-se a um menu de segundo nível onde estão listados os PIDs suportados pelo veículo, verificados na fase inicialização do sistema. A lista é dividida em páginas de 3 PIDs que funciona de forma continua, isto é, a mudança de página é automática durante a navegação na lista.

Essa lista é apresentada pela Figura 37, pela Figura 38 e pela Figura 39.

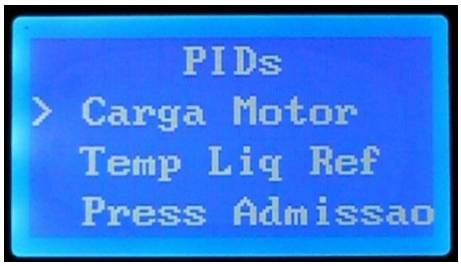

Figura 37 - Lista PIDs – Pág. 1

Figura 38 - Lista PIDs – Pág. 2

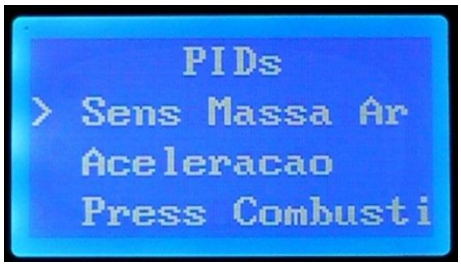

Figura 39 - Lista PIDs – Pág. 3

Da lista de PIDs é possível seleccionar até 3 elementos em simultâneo (de qualquer página), e de seguida passar ao terceiro nível. O sistema passa então a ler continuamente os valores dos PIDs seleccionados e a mostrar os respectivos valores precedidos da identificação do PID. Regressando ao menu anterior é possível desseleccionar os PID actualmente escolhidos e repetir o processo.

As figuras seguintes – Figura 40, Figura 41 e Figura 42 – representam um exemplo de selecção de 3 PIDs de páginas distintas.

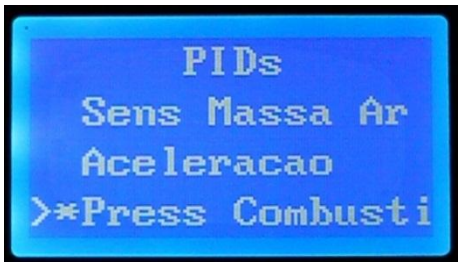

Figura 40 - Selecção de PID – Pág. 1

Figura 41 - Selecção de PID – Pág. 2

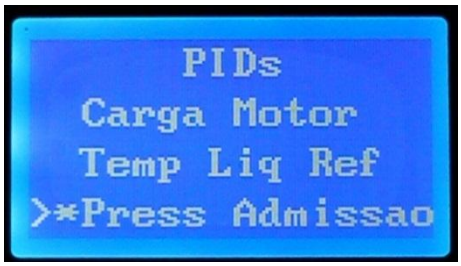

Figura 42 - Selecção de PID – Pág. 3

A segunda opção – Leitura de Valores *Freeze Frame* – está estruturada de forma idêntica à opção anterior. Os PIDs estão listados – os mesmos da lista anterior – e o utilizador selecciona até um máximo de 3 PIDs em simultâneo para ler os valores de *freeze frame* correspondentes.

Voltando ao menu principal, caso se escolha terceira opção – leitura de erros conforme a Figura 43 – chega-se a outro de segundo nível, que informa (durante 2 seg.) o utilizador do números de erros armazenados na ECU – nenhum erro no caso da Figura 44 e 2 erros no caso da Figura 45 – e lista depois os erros registados (caso existam) conforme a Figura 46 ou então regressa ao Menu Principal.

Figura 43 - Menu Principal

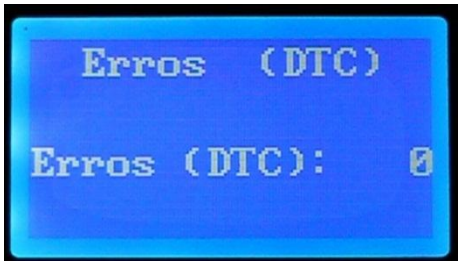

Figura 44 - Informação de Erros DTC: Sem erros

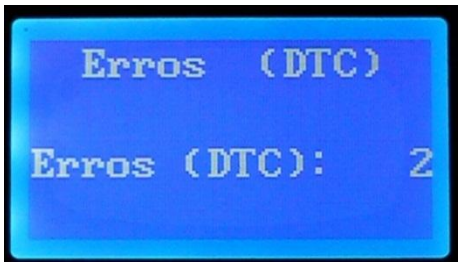

Figura 45 - Informação de Erros DTC: Com erros

Figura 46 - Informação de Erros DTC: Lista de erros

É possível eliminar os erros armazenados na ECU, sendo para isso necessário que o motor esteja desligado e a ignição na posição *ON*. Caso estas condições não se verifiquem, o veículo devolve uma mensagem de erro e o utilizador é informado no ecrã, conforme demonstra a Figura 47.

Figura 47 - Informação de Erros DTC: Condições necessárias para a eliminação de erros

A Figura 48 exemplifica a estrutura dos menus no sistema.

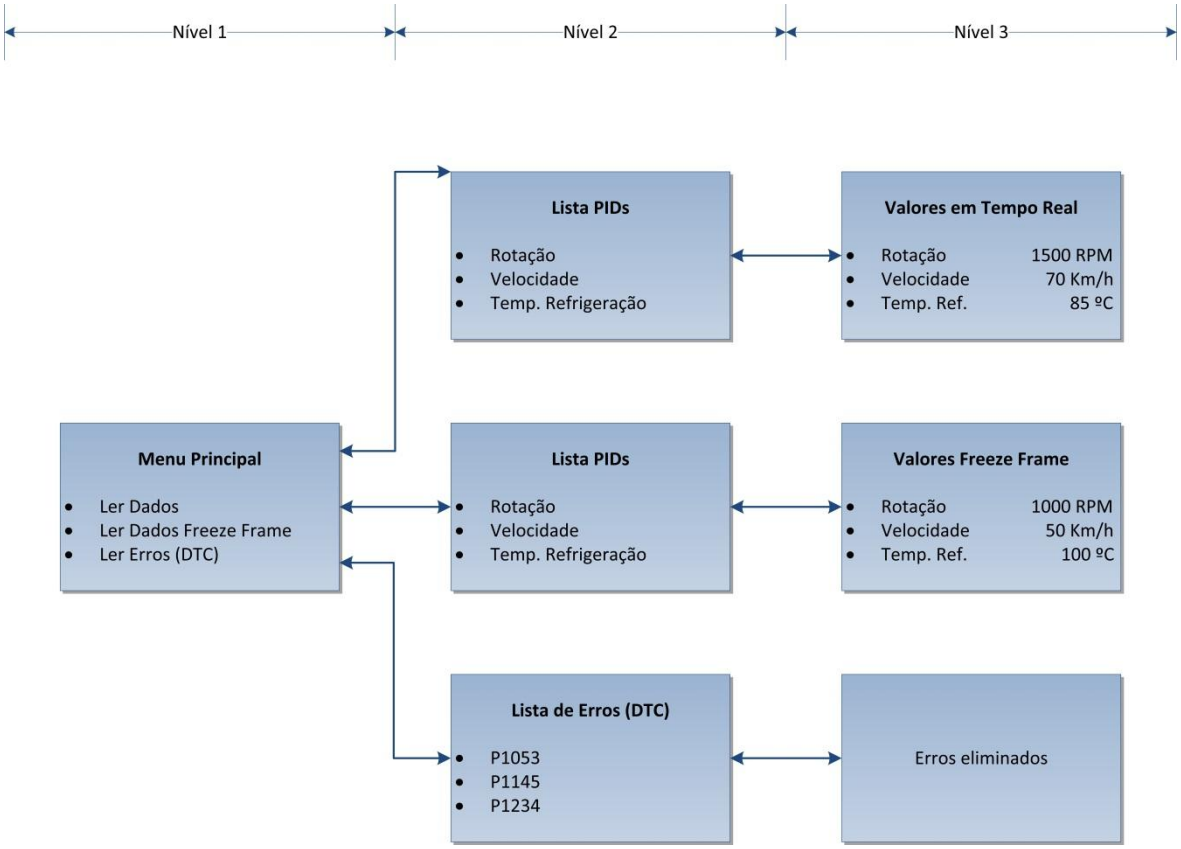

Figura 48 - Diagrama da estrutura dos menus

3.4.2.2| Navegação

A navegação nos menus do sistema é controlada pelo teclado de 5 botões.

Os botões Cima e Baixo têm a função de navegar pelas opções de uma lista, por exemplo a lista de erros.

Para navegar entre os vários níveis dos menus, são utilizados o botão “Esquerda” ou “Anterior” para voltar ao menu anterior, ou o botão “Direita” ou “Seguinte” para avançar para o próximo menu.

Finalmente, o botão “Sel” tem como função seleccionar ou desseleccionar cada membro de uma lista, como por exemplo cada PID no menu de leitura de dados em tempo real.

No menu de listagem de erros (DTC), o botão direita envia um comando de eliminação de erros.

Apesar de o teclado gerar uma interrupção a cada botão pressionado, as funções atribuídas a cada botão não estão na rotina de atendimento à interrupção. Ao invés disso é utilizado um sistema de *flags* que activa a *flag* correspondente a cada botão cada vez que este é pressionado.

O intervalo máximo entre cada verificação das *flags* é suficientemente curto para que o utilizador não tenha qualquer percepção de atraso entre o momento em que o botão é pressionado e a acção é realizado pelo microcontrolador, criando uma noção de interactividade em tempo real.

3.4.3| Leitura de Dados em Tempo Real

Durante todo o processo de inicialização e configuração do sistema, todos os PIDs suportados pelo veículo são listados no respectivo menu.

Quando o utilizador entra no menu Lista de PIDs tem a possibilidade de seleccionar até 3 PIDs para ler os seus valores em tempo real. Quando passa ao ecrã de visualização dos dados o sistema OBD2/EOBD inicia um ciclo em que envia mensagens de pedido consecutivas com SID $01 (*Request current powertrain diagnostic data*) e com os respectivos PIDs seleccionados.

No final de cada conjunto de mensagens de pedido enviadas – uma por cada PID seleccionado – o sistema calcula os respectivos valores a partir das mensagens recebidas da ECU e actualiza o ecrã.

Os cálculos são efectuados numa função que utiliza a fórmula correspondente a cada PID.

Depois de devidamente calculado o valor, a rotina de actualização do ecrã encarregar-se-á de fazer a respectiva representação com o método do ponto fixo, anteriormente descrito neste documento.

O sistema OBD2/EOBD permanecerá neste ciclo até que o utilizador regresse ao menu anterior utilizando o teclado.

3.4.4| Leitura de Valores de *Freeze Frame*

Esta funcionalidade tem como objectivo aceder aos valores que cada sensor associado a um PID media durante a ocorrência de um erro (DTC).

O funcionamento é idêntico ao da função de leitura de valores em tempo real, em que o utilizador tem uma lista de PIDs suportados pelo veículo e pode escolher até 3 PIDs em simultâneo para visualizar os valores de *freeze frame* correspondentes.

As diferenças residem na mensagem de pedido que tem o SID $02 (*Request powertrain freeze frame data*) e no facto de este pedido não ser cíclico, isto é, o pedido é feito à ECU apenas uma vez, já que esta informação está armazenada na memória e não se alterará até nova ocorrência de erros.

3.4.5| Leitura/Eliminação de Erros (DTC)

Quando o utilizador entra no menu de leitura de erros (DTC) o sistema OBD2/EOBD faz a leitura dos erros actualmente guardados na ECU.

É enviada uma mensagem com o SID $03 (*Request emission-related diagnostic trouble codes*) para verificar os erros armazenados na ECU.

Caso seja recebida uma resposta da ECU a indicar 0 (zero) erros, é enviada uma mensagem para o ecrã a informar o utilizador de que não existem erros armazenados. Caso contrários os mesmos serão listados de forma semelhante ao que acontece com a lista de PIDs no menu de leitura de dados em tempo real.

Para eliminar os erros armazenados o utilizador apenas terá que pressionar o botão “Seguinte”. O sistema envia então uma mensagem com o SID $04 (*Clear/reset emission-related diagnostic trouble codes*) e aguarda uma mensagem de confirmação de eliminação dos erros, para mostrar de seguida uma mensagem no ecrã a informar o utilizador se os erros foram correctamente eliminados da ECU.

4| Conclusões e Visão para o Futuro

O equipamento desenvolvido cumpre os objectivos iniciais do trabalho.

Foram efectuados vários testes em diversos veículos, indicados na Tabela 21.

Tabela 21 - Veículos de teste do equipamento

Marca	Modelo	Motorização	Ano
Audi	TT 2.0	2.0 TFSI	2009
BMW	123d	2.0 d	2007
Mini	Cooper D	1.6 HDI	2009
Renault	Megane III	1.5 dCi	2009
Smart	ForTwo	0.8 CDI	2008
Volkswagen	Golf V	1.9 TDI	2008
Volkswagen	Tiguan	1.4 TFSI	2009

Em todos os testes o dispositivo funcionou correctamente, e cumpriu os requisitos:

- Leitura correcta dos valores de cada PID
- Leitura de erros DTC
- Eliminação de erros DTC

Foram também realizados alguns testes ao funcionamento do temporizador que monitoriza o tempo de resposta do veículo aos pedidos do equipamento de diagnóstico (*Timer1*), e verificou-se que os tempos máximos de resposta são respeitados.

Em relação a trabalho futuro, seria bastante interessante complementar as suas funcionalidades com monitorização de eficiência energética. Isto é, implementar uma função de monitorização de consumos e indicar ao condutor se está a praticar uma condução ecologicamente eficiente, ou se pelo contrário, está a praticar uma condução descuidada, e consequentemente mais poluente e dispendiosa em termos de consumo de combustível.

Outro ponto de interesse, seria a implementação deste sistema com uma maior capacidade de memória, que possibilitaria a identificação e descrição detalhada dos erros DTC e o armazenamento de informações lidas a partir dos PIDs (*data log*).

Outra funcionalidade de interesse seria a possibilidade de alteração de parâmetros de outros módulos do veículo, por exemplo activar/desactivar funcionalidades como o sistema "*Coming Home*" ou "*Follow Me Home*" que activa a iluminação exterior do veículo durante alguns segundos. Em vários modelos de veículos automóveis actuais essas funcionalidades vêm instaladas de fábrica, sendo apenas activadas (ou não) em função dos extras solicitados durante a compra, pela porta OBD2 do veículo. No entanto esse trabalho não será facilitado uma vez que actualmente os endereços para comunicação com esses módulos e os respectivos comandos são proprietários e não estão disponíveis ao público.

Bibliografia

ISO (International Organization of Standardization). (2001). ISO 15031 - Road vehicles - Communication between vehicle and external equipment for emissions-related diagnostics.
ISO (International Organization of Standardization). (2003). ISO 11898 - Road vehicles - Controller Area Network (CAN).
ISO (International Organization of Standardization). (2004). ISO 15765 - Road vehicles - Diagnostics on Controller Area Networks.
Microchip. (1999). AN713 - Controller Area Network (CAN) Basics.
Microchip. (2002). MPLAB C18 C Compiler Libraries.
Microchip. (2003). MCP2551 - High-Speed CAN Transceiver.
Microchip. (2005). MCP2515 - Stand-Alone CAN Controller With SPI Interface.
Microchip. (2005). MPLAB C18 C Compiler User's Guide.
Microchip. (2010). PIC18(L)F2X/4XK22 Data Sheet - 28/40/44-Pin, Low-Power, High-Performance Microcontrollers, with nanoWatt XLP Technology.
Microchip. (s.d.). Optimizing the Interrupt Service Routines in MPLAB C18 Compiler.
Robert Bosch GmbH. (1991). CAN Specification - Version 2.0.
Sitronix. (2009). ST7565P - 65 x 132 Dot Matrix LCD Controller/Driver.

Anexo A – Tabela de SIDs

A Tabela 22 apresenta a lista de SID disponíveis na comunicação via OBD2.

Tabela 22 - Tabela de SIDs

SID	Descrição
$01	*Request current powertrain diagnostic data*
$02	*Request powertrain freeze frame data*
$03	*Request emission-related diagnostic trouble codes*
$04	*Clear/reset emission-related diagnostic trouble codes*
$05	*Request oxygen sensor monitoring test results*
$06	*Request on-board monitoring test results for specific monitored systems*
$07	*Request emission-related diagnostic trouble codes detected during current or last completed driving cycle*
$08	*Request control of on-board system/test/component*
$09	*Request vehicle information*

Anexo B – Tabela de PIDs (*Request Supported PIDs*)

A Tabela 23 apresenta a lista de PIDs que verificam os PIDs suportados pelo veículo e a respectiva interpretação.

Tabela 23 - Tabela de PIDs de verificação de PIDs suportados

PID	Byte	Supported PID	Supported
$00	Data A bit 7 Data A bit 6 . . Data D bit 0	$01 $02 . . $20	0 = not supported 1 = supported
$20	Data A bit 7 Data A bit 6 . . Data D bit 0	$21 $22 . . $40	0 = not supported 1 = supported
$40	Data A bit 7 Data A bit 6 . . Data D bit 0	$41 $42 . . $60	0 = not supported 1 = supported
$60	Data A bit 7 Data A bit 6 . . Data D bit 0	$61 $62 . . $80	0 = not supported 1 = supported
$80	Data A bit 7 Data A bit 6 . . Data D bit 0	$81 $82 . . $A0	0 = not supported 1 = supported
$A0	Data A bit 7 Data A bit 6 . . Data D bit 0	$A1 $A2 . . $C0	0 = not supported 1 = supported
$C0	Data A bit 7 Data A bit 6 . . Data D bit 0	$C1 $C2 . . $E0	0 = not supported 1 = supported
$E0	Data A bit 7 Data A bit 6 . Data D bit 1 Data D bit 0	$E1 $E2 . $FF ISO/SAE Reserved	0 = not supported 1 = supported

Anexo C – Tabela de PIDs

A Tabela 24 apresenta a lista de PID disponíveis para os SID $01 e $02.

Tabela 24 - Tabela de PIDs para SIDs $01 e $02

PID	Descrição	Byte	Escalonamento	Métrica Unidades
$00	Supported PIDs ($01-$20)		Anexo B	
$01	Monitor status since DTCs cleared	A, B, C, D	Anexo C - Tabela 25	
$02	DTC that caused required freeze frame data storage	A, B	Hexadecimal Ex: P01AB	DTCFRZF: Pxxxx Cxxxx Bxxxx Uxxxx
$03	Fuel system status		Anexo C - Tabela 26	
$04	Calculated LOAD Value	A	A*100/255	LOAD_PCT: xxx.x %
$05	Engine Coolant Temperature	A	A-40	ECT: xxx °C
$06	Short Term Fuel Trim – Bank 1/3	A (1) B (3)	(A-128) * 100/128 (B-128) * 100/128	SHRTFT1: xxx.x % SHRTFT3: xxx.x %
$07	Long Term Fuel Trim – Bank 1/3	A (1) B (3)	(A-128) * 100/128 (B-128) * 100/128	LONGFT1: xxx.x % LONGFT3: xxx.x %
$08	Short Term Fuel Trim - Bank 2/4	A (1) B (3)	(A-128) * 100/128 (B-128) * 100/128	SHRTFT2: xxx.x % SHRTFT4: xxx.x %
$09	Long Term Fuel Trim – Bank 2/4	A (1) B (3)	(A-128) * 100/128 (B-128) * 100/128	LONGFT2: xxx.x % LONGFT4: xxx.x %
$0A	Fuel Rail Pressure (gauge)	A	A*3	FRP: xxx kPa
$0B	Intake Manifold Absolute Pressure	A	A	MAP: xxxx.x kPa
$0C	Engine RPM	A, B	((A*256)+B)/4	RPM: xxxxx min^{-1}
$0D	Vehicle Speed Sensor	A	A	VSS: xxx km/h
$0E	Ignition Timing Advance for #1 Cylinder	A	A/2-64	SPARKADV: xx.x °
$0F	Intake Air Temperature	A	A-40	IAT: xxx °C
$10	Air Flow Rate from Mass Air Flow Sensor	A, B	((A*256)+B) / 100	MAF: xxxx.xx g/s
$11	Absolute Throttle Position	A	A*100/255	TP: xxx.x %
$12	Commanded Secondary Air Status			
$13	Location of Oxygen Sensors			
$14	Bank 1 – Sensor 1	A B	A/200 (B-128) * 100/128	
$15	Bank 1 – Sensor 2	A B	A/200 (B-128) * 100/128	
$16	Bank 1 – Sensor 3	A B	A/200 (B-128) * 100/128	
$17	Bank 1 – Sensor 4	A B	A/200 (B-128) * 100/128	
$18	Bank 2 – Sensor 1	A B	A/200 (B-128) * 100/128	
$19	Bank 2 – Sensor 2	A B	A/200 (B-128) * 100/128	
$1A	Bank 2 – Sensor 3	A B	A/200 (B-128) * 100/128	
$1B	Bank 2 – Sensor 4	A B	A/200 (B-128) * 100/128	
$1C	OBD requirements to which vehicle is designed	A	01 02 03 04	OBD2 OBD OBD and OBD II OBD I

			05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 – FA FB – FF	NO OBD EOBD EOBD and OBD II EOBD and OBD EOBD, OBD and OBD II JOBD JOBD and OBD II JOBD and EOBD JOBD, EOBD, and OBD II EURO IV B1 EURO V B2 EURO C EMD - SAE J1939 special meaning
$1D	Location of oxygen sensors		Anexo C - Tabela 27	
$1E	Auxiliary Input Status		Anexo C - Tabela 28	
$1F	Time Since Engine Start	A, B	(A*256)+B	RUNTM: xxxxx sec.
$20	Supported PIDs ($21-$40)		Anexo B	
$21	Distance Travelled While MIL is Activated	A, B	(A*256)+B	MIL_DIST: xxxxx km
$22	Fuel Rail Pressure relative to manifold vacuum	A, B	((A*256)+B) * 0.079	FRP: xxxx.x kPa
$23	Fuel Rail Pressure	A, B	((A*256)+B) * 10	FRP: xxxxxx kPa
$24	Bank 1 – Sensor 1 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 or ((A*256)+B)/32768 ((C*256)+D)*8/65535 or ((C*256)+D)/8192	
$25	Bank 1 – Sensor 2 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$26	Bank 1 – Sensor 3 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$27	Bank 1 – Sensor 4 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$28	Bank 2 – Sensor 1 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$29	Bank 2 – Sensor 2 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$2A	Bank 2 – Sensor 3 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$2B	Bank 2 – Sensor 4 (wide range O2S)	A, B C, D	((A*256)+B)*2/65535 ((C*256)+D)*8/65535	
$2C	Commanded EGR	A	A*100/255	EGR_PCT: xxx.x %
$2D	EGR Error	A	(A-128) * 100/128	EGR_ERR: xxx.x %
$2E	Commanded Evaporative Purge	A	A*100/255	EVAP_PCT: xxx.x %
$2F	Fuel Level Input	A	A*100/255	FLI: xxx.x %
$30	Number of warm-ups since diagnostic trouble codes cleared	A	A	WARM_UPS: xxx
$31	Distance since diagnostic trouble codes cleared	A, B	(A*256)+B	CLR_DIST: xxxxx km
$32	Evap System Vapour Pressure	A, B	((A*256)+B)/4 (A is signed)	EVAP_VP: xxxx.x Pa
$33	Barometric Pressure	A	A	BARO: xxx kPa
$34	Bank 1 – Sensor 1 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$35	Bank 1 – Sensor 2 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$36	Bank 1 – Sensor 3 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$37	Bank 1 – Sensor 4 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$38	Bank 2 – Sensor 1 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	

$39	Bank 2 – Sensor 2 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$3A	Bank 2 – Sensor 3 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$3B	Bank 2 – Sensor 4 (wide range O2S)	A, B C, D	((A*256)+B)/32,768 ((C*256)+D)/256 - 128	
$3C	Catalyst Temperature Bank 1, Sensor 1	A, B	((A*256)+B)/10 - 40	CATEMP11: xxxx °C
$3D	Catalyst Temperature Bank 2, Sensor 1	A, B	((A*256)+B)/10 - 40	CATEMP21: xxxx °C
$3E	Catalyst Temperature Bank 1, Sensor 2	A, B	((A*256)+B)/10 - 40	CATEMP12: xxxx °C
$3F	Catalyst Temperature Bank 2, Sensor 2	A, B	((A*256)+B)/10 - 40	CATEMP22: xxxx °C
$40	Supported PIDs ($41-$60)		Anexo B	
$41	Monitor status this driving cycle			
$42	Control module voltage	A, B	((A*256)+B)/1000	VPWR: xx.xx V
$43	Absolute Load Value	A, B	((A*256)+B)*100/255	LOAD_ABS: xxxxx.x %
$44	Commanded Equivalence Ratio	A, B	((A*256)+B)/32768	EQ_RAT: xxx.xxx
$45	Relative Throttle Position	A	A*100/255	TP_R: xxx.x %
$46	Ambient air temperature	A	A-40	AAT: xxx °C
$47	Absolute Throttle Position B	A	A*100/255	TP_B: xxx.x %
$48	Absolute Throttle Position C	A	A*100/255	TP_C: xxx.x %
$49	Accelerator Pedal Position D	A	A*100/255	APP_D: xxx.x %
$4A	Accelerator Pedal Position E	A	A*100/255	APP_E: xxx.x %
$4B	Accelerator Pedal Position F	A	A*100/255	APP_F: xxx.x %
$4C	Commanded Throttle Actuator Control	A	A*100/255	TAC_PCT: xxx.x %
$4D	Time run by the engine while MIL is activated	A, B	(A*256)+B (minutes)	MIL_TIME: xxxx hrs, xx min
$4E	Time since diagnostic trouble codes cleared	A, B	(A*256)+B (minutes)	CLR_TIME: xxxx hrs, xx min
$4F	Maximum value for Equivalence Ratio	A, B, C, D	A, B, C, D*10	These values are not intended for display to the service technician.
$50	Maximum value for Air Flow Rate from Mass Air Flow Sensor	A, B, C, D	A*10, B, C, and D are reserved for future use	These values are not intended for display to the service technician.
$51	Type of fuel currently being utilized by the vehicle			
$52	Alcohol Fuel Percentage	A	A*100/255	ALCH_PCT: xxx.x %
$53	Absolute Evap System Vapour Pressure	A	A/200 per bit	EVAP_VPA: xxx.xxx kPa
$54	Evap System Vapour Pressure	A, B	A*256+B - 32768	EVAP_VP: xxxxx Pa
$55	Short Term Secondary O2 Sensor Fuel Trim – Bank 1/3	A (1) B (3)	(A-128)*100/128 (B-128)*100/128	STSO2FT1: xxx.x % STSO2FT3: xxx.x %
$56	Long Term Secondary O2 Sensor Fuel Trim – Bank 1/3	A (1) B (3)	(A-128)*100/128 (B-128)*100/128	LGSO2FT1: xxx.x % LGSO2FT3: xxx.x %
$57	Short Term Secondary O2 Sensor Fuel Trim - Bank 2/4	A (2) B (4)	(A-128)*100/128 (B-128)*100/128	STSO2FT2: xxx.x % STSO2FT4: xxx.x %
$58	Long Term Secondary O2 Sensor Fuel Trim – Bank 2/4	A (2) B (4)	(A-128)*100/128 (B-128)*100/128	LGSO2FT2: xxx.x % LGSO2FT4: xxx.x %
$59	Fuel Rail Pressure (absolute)	A, B	((A*256)+B) * 10	FRP: xxxxxx kPa
$5A	Relative Accelerator Pedal Position	A	A*100/255	APP_R: xxx.x %
$5B - $FF	ISO/SAE reserved			

Tabela 25 - Mapa de bits SID $01 PID $01

Byte	Bit	Nome	Escalonamento	Métrica Unidades
A	7	MIL - Malfunction Indicator Lamp	0 = OFF 1 = ON	OFF ON
A	6-0	Number of DTCs stored in this ECU	Hex to decimal	xx d
B	7	ISO/SAE reserved (bit shall be reported as “0”)		
B	6	Comprehensive component monitoring ready	0 = monitor complete 1 = monitor not complete	YES NO
B	5	Fuel system monitoring ready	0 = monitor complete 1 = monitor not complete	YES NO
B	4	Misfire monitoring ready	0 = monitor complete 1 = monitor not complete	YES NO
B	3	ISO/SAE reserved (bit shall be reported as “0”)		
B	2	Comprehensive component monitoring supported	0 = monitor not supported 1 = monitor supported	NO YES
B	1	Fuel system monitoring supported	0 = monitor not supported 1 = monitor supported	NO YES
B	0	Misfire monitoring supported	0 = monitor not supported 1 = monitor supported	NO YES
C	7	EGR system monitoring supported	0 = monitor not supported 1 = monitor supported	NO YES
C	6	Oxygen sensor heater monitoring supported		
C	5	Oxygen sensor monitoring supported		
C	4	A/C system refrigerant monitoring supported		
C	3	Secondary air system monitoring supported		
C	2	Evaporative system monitoring supported		
C	1	Heated catalyst monitoring supported		
C	0	Catalyst monitoring supported		
D	7	EGR system monitoring ready	0 = monitor complete 1 = monitor not complete	YES NO
D	6	Oxygen sensor heater monitoring ready		
D	5	Oxygen sensor monitoring ready		
D	4	A/C system refrigerant monitoring ready		
D	3	Secondary air system monitoring ready		
D	2	Evaporative system monitoring ready		
D	1	Heated catalyst monitoring ready		
D	0	Catalyst monitoring ready		

Tabela 26 - Mapa de bits SID $01 PID $03

Byte	Bit	Nome	Escalonamento	Métrica Unidades
		Fuel system 1 status		FUELSYS: 1
A	7-5	ISO/SAE reserved (bits shall be reported as ‘0’)		
A	4	1 = Closed loop, but fault with at least one oxygen sensor - may be using single oxygen sensor for fuel control		CL-Fault
A	3	1 = Open loop - due to detected system fault		OL-Fault

A	2	1 = Open loop due to driving conditions (e.g. power enrichment, deceleration enleanment)	OL-Drive
A	1	1 = Closed loop - using oxygen sensor(s) as feedback for fuel control	CL
A	0	1 = Open loop - has not yet satisfied conditions to go closed loop	OL
		Fuel system 2 status	FUELSYS: 2
B	7-5	ISO/SAE reserved (bit shall be reported as “0”)	
B	4	1 = Closed loop, but fault with at least one oxygen sensor - may be using single oxygen sensor for fuel control	CL-Fault
B	3	1 = Open loop - due to detected system fault	OL-Fault
B	2	1 = Open loop due to driving conditions (e.g. power enrichment, deceleration enleanment)	OL-Drive
B	1	1 = Closed loop - using oxygen sensor(s) as feedback for fuel control	CL
B	0	1 = Open loop - has not yet satisfied conditions to go closed loop	OL

Tabela 27 - Definição PID $1D

Byte	Bit	Nome	Escalonamento	Métrica Unidades
		Location of oxygen sensors		O2SLOC:
A	7	1 = Bank 4 - Sensor 2 present at that location		O2S42
A	6	1 = Bank 4 - Sensor 1 present at that location		O2S41
A	5	1 = Bank 3 - Sensor 2 present at that location		O2S32
A	4	1 = Bank 3 - Sensor 1 present at that location		O2S31
A	3	1 = Bank 2 - Sensor 2 present at that location		O2S22
A	2	1 = Bank 2 - Sensor 1 present at that location		O2S21
A	1	1 = Bank 1 - Sensor 2 present at that location		O2S12
A	0	1 = Bank 1 - Sensor 1 present at that location		O2S11

Tabela 28 - Definição PID $1E

Byte	Bit	Nome	Escalonamento	Métrica Unidades
A	7-1	ISO/SAE reserved (Bits shall be reported as ‘0’.)		
A	0	Power Take Off (PTO) Status	0 = PTO not active (OFF) 0 = PTO not active (OFF);	Auxiliary Input Status

Anexo D – Esquemas eléctricos, PCBs e outras figuras

Módulo OBD2/EOBD

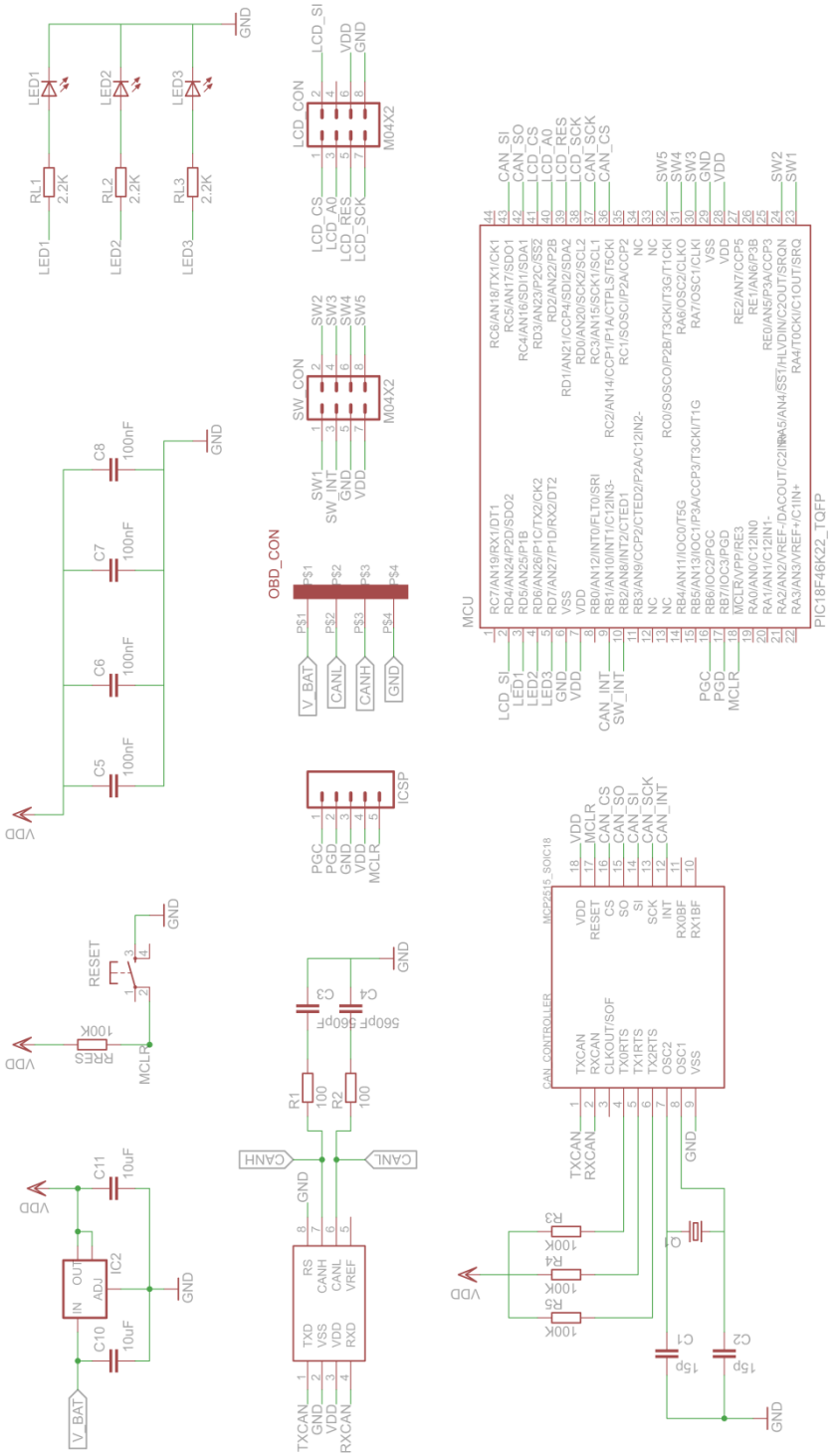

Figura 49 - Esquema eléctrico do sistema OBD2/EOBD

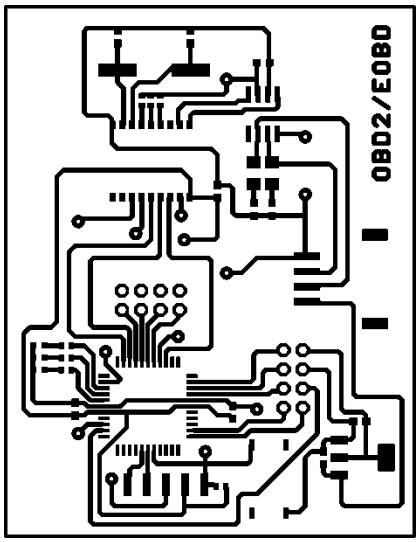

Figura 50 - PCB do sistema OBD2/EOBD (*Top Layer*)

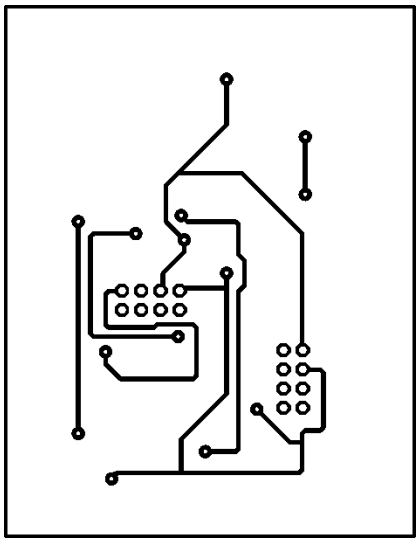

Figura 51 - PCB do sistema OBD2/EOBD (*Bottom Layer*)

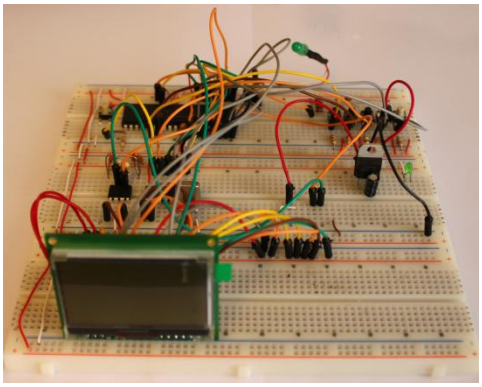

Figura 52 - Protótipo do módulo OBD2/EOBD (vista 1)

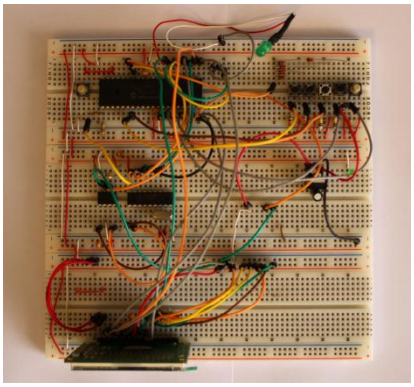

Figura 53 - Protótipo do módulo OBD2/EOBD (vista 2)

Teclado

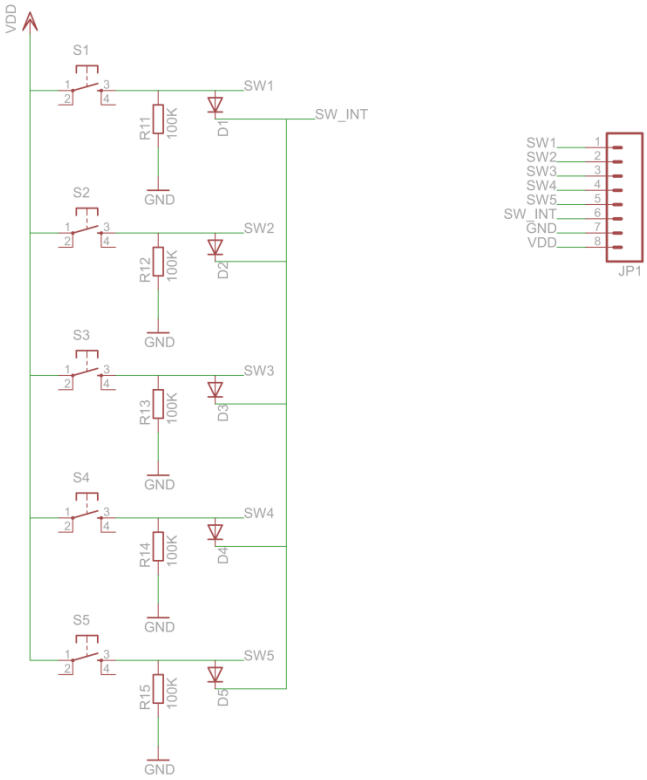

Figura 54 - Esquema eléctrico do teclado

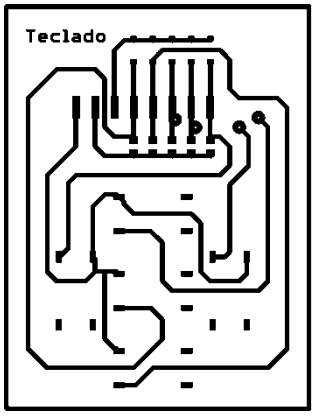

Figura 55 - PCB do teclado