FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Sistema de Comunicações Tolerante a Falhas e de Baixa Complexidade para um Veículo Eléctrico

Bruno Laranjo dos Santos

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientador: Paulo José Lopes Machado Portugal (Professor Doutor)

Julho de 2012

Resumo

Esta dissertação propõe uma plataforma de *hardware* e *software* para o futuro desenvolvimento de aplicações de controlo por parte dos alunos do MIEEC (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) no VEC (Veículo Elétrico de Competição) da FEUP (Faculdade de Engenharia da Universidade do Porto). Neste sentido, propõe um sistema de comunicação tolerante a falhas e de baixa complexidade em que seja possível a troca de mensagens seguros na rede.

O mecanismo de acesso ao meio consiste num TDMA (*Time Division Multiple Access*), com o uso de barramento redundantes. Este mecanismo utiliza um microcontrolador com 2 módulos CAN.

Os dados a enviar são escalonados nas mensagens na fase de projeto, sendo as mensagens trocadas entre nodos dentro do ciclo TDMA. Este mecanismo de acesso ao meio tolerante a falhas é completado com a implementação de uma *safety layer* de comunicação para o aumento da confiabilidade das mensagens trocadas na rede.

A implementação das soluções de comunicação é feita no RTOS (*Real Time Operating System*) *Erika Enterprise & RT Druid* do tipo OSEK/VDX (Open Systems and their Interfaces for the Electronics in Motor Vehicles) da marca *Evidence*. Na aplicação implementada no RTOS, é possível trocar mensagens com o uso do mecanismo de acesso ao meio TDMA.

Palavras chave - Comunicações, veículos, CAN (*Controller Area Network*), TDMA (*Time Division Multiple Access*), *safety layer*, Sistema operativo tempo real, OSEK/VDX, AUTOSAR (*Automotive Open System ARchitecture*).

Abstract

This thesis introduces a hardware and software platform for the future development of control applications from MIEEC (Master in Electrical and Computer Engineering) students in FEUP (Faculty of Engineering, University of Porto) VEC's (Electric Vehicle Competition) . It presents the development of a low complexity and fault-tolerant communication system as it is possible to exchange reliable data in the network.

It's used a TDMA media access mechanism, using redundant bus with two modules CAN in a microcontroller.

The data are scheduled to the messages in the design phase and these are exchanged between nodes inside of TDMA cycle. This fault tolerant media access mechanism is completed with the implementation of a safety communication layer in order to increase the reliability of the messages exchanged in the network.

The implementation of communication solutions is performed in OSEK / VDX RTOS *Erika Enterprise & RT Druid* brand by Evidence. It is possible to exchange messages using the TDMA redundant media access mechanism with the application implemented on the RTOS.

Keywords - Comunication, vehicles, CAN (Controller Area Network), TDMA (Time Division Multiple Access), safety layer, Real Time Operating System, OSEK/VDX, AUTOSAR (Automotive Open System ARchitecture).

Agradecimentos

Gostaria de agradecer,
Ao Professor Paulo Portugal, pela motivação, a enorme ajuda no desenvolvimento desta dissertação.

Ao meus pais João e Judite e ao meu irmão Alexandre, pelo carinho, motivação e a mais que preciosa ajuda por esses vinte e quatro anos de vida ao vosso lado.

Ao meu avô Joaquim, dos Homens a quem eu quero dedicar este documento.

À Élia, pelos momentos ao meu lado, pela grande paciência, pelo carinho e pela grande ajuda nos momentos menos bons.

À todos os meus amigos, em especial ao André Gonçalves, ao Diogo Varajão e ao Tiago Bastos, pelas ideias partilhadas e pelas coisas que não se enquadram no contexto académico.

Bruno Laranjo dos Santos

"If we all did the things we are really capable of doing,
we would literally astound ourselves"

Thomas Alva Edison

Conteúdo

1 Introdução **1**
- 1.1 Motivação . . . 1
- 1.2 Objetivos . . . 2
- 1.3 Estrutura do Documento . . . 2

2 Enquadramento **5**
- 2.1 Introdução . . . 5
- 2.2 *X by Wire*(XBW) . . . 6
  - 2.2.1 *Brake by Wire* (BBW) . . . 8
  - 2.2.2 Arquiteturas *Brake by Wire* . . . 9
- 2.3 Rede de Comunicação . . . 12
  - 2.3.1 Introdução . . . 13
  - 2.3.2 CAN . . . 14
  - 2.3.3 TT-CAN . . . 17
  - 2.3.4 FTT-CAN . . . 19
- 2.4 Gestão das aplicações . . . 21
  - 2.4.1 Introdução . . . 21
  - 2.4.2 Sistema Operativo Tempo Real . . . 21
  - 2.4.3 OSEK/VDX . . . 22
  - 2.4.4 AUTOSAR . . . 27

3 Proposta **31**
- 3.1 Introdução . . . 31
- 3.2 Resumo . . . 31
- 3.3 Mecanismo de envio de mensagens tolerante a falhas . . . 32
  - 3.3.1 Limitação do protocolo CAN . . . 33
  - 3.3.2 Solução do *Time Triggered* Redundante CAN . . . 35
  - 3.3.3 *Safety-Layer* . . . 37
  - 3.3.4 Formato da mensagem . . . 42
- 3.4 Sistema Operativo Tempo Real . . . 44
  - 3.4.1 Implementações de Sistemas Operativos Tempo Real . . . 44
  - 3.4.2 RTOS escolhido . . . 45
- 3.5 Plataforma de *hardware* . . . 46

4 Implementação **49**
- 4.1 A estrutura do RTOS . . . 49
  - 4.1.1 O Ficheiro OIL . . . 49
- 4.2 Descrição da Aplicação . . . 52

4.3 Estrutura da Aplicação 53
4.3.1 Tarefas de Comunicação e Supervisão 53
4.3.2 Hierarquia de Tarefas da Aplicação 54
4.3.3 Subaplicação de comunicação 54
4.3.4 Subaplicação de supervisão 59
4.4 Teste da Implementação 60

5 Conclusões **63**
5.1 Conclusões da Dissertação 63
5.2 Trabalho futuro 64

A Código **65**
A.1 Exemplo ficheiro tipo OIL 65

Referências **69**

Lista de Figuras

2.1 Diferenças entre um sistema tradicional (esquerda) e um sistema *drive by wire* (direita) [1] 7
2.2 Esquema básico de um sistema *drive by wire* [2] 8
2.3 Esquema da arquitetura proposta em [3] 10
2.4 Esquema básico de um sistema *Fail Operational Unit* com duas *fail-silent units* [3] 10
2.5 Esquema da arquitetura proposta em [2] 11
2.6 Esquema da arquitetura proposta em [4] 12
2.7 Formato da trama básica CAN [5] 15
2.8 Esquema do acesso ao meio em barramentos CAN [6] 16
2.9 Maquina de estados de erros em CAN [6] 17
2.10 Descrição de uma *Transmition Column* [7] 18
2.11 Descrição do Ciclo Principal no TT-CAN [7] 19
2.12 Exemplo de funcionamento do TM [8] 20
2.13 Exemplo de um ciclo elementar; [8] 20
2.14 Diagrama de uma aplicação OSEK num microcontrolador [9] 22
2.15 Diagrama de estado das Tarefas em OSEK (adaptado de [9]) 24
2.16 Esquema de gestão das Tarefas em OSEK [9] 26
2.17 Divisão por camadas do software do AUTOSAR [10] 28

3.1 Esquema geral da proposta da solução 32
3.2 Inconsistência na recepção dos dados no CAN [11] 34
3.3 Principio do ciclo TDMA (adaptado de [12]) 36
3.4 Esquema geral do mecanismo de Transmissão da solução (adaptado de [12]) . . . 36
3.5 Esquema da Transmissão proposto (adaptado de [12]) 38
3.6 Esquema da Recepção proposto (adaptado de [12]) 39
3.7 Formato da trama do protocolo de segurança critica 42
3.8 Gráfico comparativo de diversos algoritmos de CRC de 8 bits [13] 43
3.9 Esquema do microcontrolador proposto [14] 48

4.1 Estrutura geral dos elementos necessários para uma aplicação (adaptado de [15]) 50
4.2 Mecanismo de envio e recepção de mensagens 52
4.3 Disposição dos SUBAPP na implementação 53
4.4 Hierarquia de Tarefas da Aplicação 55
4.5 Disposição da componente de transmissão da SUBAPP de Comunicação 56
4.6 Disposição da componente de recepção da SUBAPP de Comunicação 57
4.7 Tarefa de recepção das mensagens CAN 58
4.8 Disposição da componente da SUBAPP de Supervisão 59
4.9 Esquema simplificado de ligação dos dois microcontroladores 60

4.10 Fotografia do circuito de desenvolvimento 61

Lista de Tabelas

2.1 Formato das tramas básicas no CAN 14
2.2 Transição entre tarefas no OSEK/VDX [9] 25

3.1 Relação entre os possíveis riscos e as ameaças para a rede [16] 40
3.2 Relação entre as ameaças cobertas com as medidas propostas pela norma EN 50159 41

4.1 Tarefas utilizadas na aplicação de comunicação 54
4.2 Atribuição dos identificadores aos tipos de mensagem 58
4.3 Testes da implementação e seus resultados 61

Abreviaturas e Símbolos

ACC	Automatic Cruise Control
ABS	Anti-lock Braking System
A/D	Analógico/Digital
AUTOSAR	Automotive Open System Architecture
API	Application Programming Interface
BA	Brake Assistant
BC	Basic Cycle
BBW	Brake by Wire
BPIU	Brake Pedal Input Unit
CAN	Controller Area Network
CRC	Cyclic Redundant Code
CSMA	Carrier Sense Multiple Access
CSMA/BA	Carrier Sense Multiple Access/Bus Arbitration
CPU	Central Processing Unit
D/A	Digital/Analógico
DLC	Data Lenght Code
EC	Elementar Cycle
EN	European Standards
ECU	Electronic Control Unit
EOF	End Of Frame
ESP	Electronic Stability Program
FEUP	Faculdade de Engenharia da Universidade do Porto
FCS	Frame Sequence Check
FOU	Fail Operational Unit
FTT-CAN	Flexible Time-Trigerred Controller Area Network
GPL	General Public License
IDE	Integrated Development Environment
IDE	IDentifier Extension
ISO	International Organization for Standardization
ISR	Interrupt Service Routine
MC	Master Cycle
MIEEC	Mestrado Integrado em Engenharia Electrotécnica e de Computadores
OSEK/VDX	Open Systems and their Interfaces for the Electronics in Motor Vehicles
PCB	Printed Circuit Board
PSA	Peugeot Société Anonyme

RM	Remote Frame
RTOS	Real Time Operating System
RTR	Remote Transmition Request
SOF	Start of Frame
SRT	Synchronous Requirement Table
SW	Synchronous Window
TC	Transmition Column
TCS	Traction Control System
TDMA	Time Division Multiple Access
TM	Triggered Message
TMR	Triple Modular Redundancy
TT-CAN	Time-Triggered Controler Area Network
TTP/C	Time-Triggered Protocol
VEC	Veículo Elétrico de Competição
XBW	x by wire

Capítulo 1

Introdução

Neste capítulo faz-se a introdução ao trabalho desenvolvido no âmbito desta dissertação. É apresentada a motivação para o desenvolvimento deste trabalho. Em seguida, listam-se os objetivos que se pretende atingir com o trabalho da dissertação. Por fim, é exposto a estrutura deste documento.

1.1 Motivação

A eletrónica automóvel tem tido um grande crescimento nos últimos anos, e como tal, tem sido cada vez mais incluída nos sistemas de controlo. Alguns exemplos resultantes dessa evolução são: o sistema de *airbag*, os sensores de auxílio ao estacionamento, os vidros elétricos, o sistema de limpa-vidros automático, o *powertrain* e o *brake by wire*.

A FEUP atenta ao desenvolvimento da electrónica automóvel, criou o laboratório de electrónica automóvel numa parceria entre o departamento de engenharia eletrotécnica e o departamento de engenharia mecânica. Esse laboratório tem diversos projetos associados a conversão de veículos térmicos convencionais em veículos elétricos. O VEC (Veículo Electrico de Competição) corresponde a um desses projetos, trata-se de um veículo elétrico para uso em competições. O veículo baseia-se no chassi do Fiat Uno 45 S, onde está a ser desenvolvido diversas aplicações pelos alunos do MIEEC.

No desenvolvimento das diversas aplicações de controlo rapidamente começaram a surgir problemas de compatibilidade entre os diversos módulos. Na altura dos sistemas interagir entre eles, surgiam problemas de compatibilidade ao nível do hardware, software, e dos meios de comunicação entre os diversos sistemas impossibilitando o pleno funcionamento do veículo.

À vista dos problemas referidas acima, a necessidade de criar uma plataforma unificada base para os diversos sistemas de controlo presentes no veiculo automóvel elétrico começou a tornar-se crucial para o desenvolvimento de aplicações robustas.

Os sistemas de controlo a implementar no veiculo, sendo sistemas que devem garantir um certo nível de integridade, devem ser fortemente tolerantes a falhas, de forma a evitar ocorrência

de avarias. As falhas, quando ocorrem, podem vir a prejudicar fortemente os sistemas, podendo-se propagar pelo sistema, originando uma perda total da funcionalidade do sistema. Os sistemas tolerantes a falhas suportam normalmente um certo números de falhas antes da ocorrência de avaria e consequente perda de funcionalidade. O número de falhas que um sistema suporta quantifica a tolerância a falhas do mesmo.

As comunicações dos sistemas de controlo devem obedecer às propriedades de tolerância a falhas e de segurança crítica na troca de mensagens. Esses dois mecanismos, em conjunto, devem garantir que a mensagem enviada pelo nodo chega ao destino dentro dos requisitos temporais do sistema e que as mensagens sejam validadas como confiáveis para o seu uso pelo sistema.

Outra componente de grande importância no desenvolvimento de sistemas distribuídos, consiste na gestão dos diversos componentes que constituem a aplicação de software. O uso de métodos padronizados de desenvolvimento de software para as unidades de controlo tornou-se inevitável, devido ao aumento crescente da complexidade das aplicações a desenvolver. Essa sistematização do desenvolvimento diminui o tempo de projeto das aplicações, pois são eliminados desenvolvimentos de interfaces inter-componentes usadas nas arquiteturas.

1.2 Objetivos

O objetivo da dissertação consiste na proposta e implementação de uma arquitetura de comunicação tolerante a falhas e de baixa complexidade, com a possibilidade da troca de mensagens seguras na rede. Esta plataforma servirá como sistema base para futuro suporte no desenvolvimento de aplicações de controlo por parte dos alunos do MIEEC (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) da FEUP (Faculdade de Engenharia da Universidade do Porto). As aplicações desenvolvidas pelos alunos do MIEEC têm como principal obejectivo a sua integração no VEC (Veículo Elétrico de Competição) do departamento de Engenharia Eletrotécnica da FEUP.

A arquitetura proposta oferece flexibilidade para ligar diversos nodos, que trocam dados entre si por meio de um mecanismo de acesso ao meio TDMA (*Time Division Multiple Access*) com base no protocolo CAN (*Controller Área Network*) e o uso de barramentos redundantes. A arquitetura de comunicação inclui ainda uma camada de segurança das comunicações, de forma a garantir a confiabilidade dos dados trocados .

Com o objetivo de se adequar ao perfil de conhecimento técnico dos alunos, o sistema de comunicação foi integrado num RTOS do tipo OSEK/VDX (*Open Systems and Their Interfaces for the Electronics in Motor Vehicles*). O OSEK/VDX é um sistema operativo de baixa complexidade adaptado às aplicações de controlo a desenvolver nos veículos.

1.3 Estrutura do Documento

O documento da dissertação encontra-se dividido em 5 capítulos, explicitados em seguida:

O primeiro e presente capítulo apresenta a motivação e enquadramento súmario do trabalho, seguindo-se os objetivos, sendo no fim efetuado um resumo da estrutura do documento.

O segundo capítulo é dedicado ao enquadramento mais promenorizado dos conceitos associados a este trabalho. Inicialmente é feita uma apresentação das arquiteturas de comunicação dos sistemas *x by wire*. Em seguida, são introduzidos os conceitos do protocolo de comunicação CAN, incluindo o TT-CAN (*Time-Triggered Controler Area Network*) e o FTT-CAN (*Flexible Time-Trigerred Controller Area Network*). No término do capítulo, são descritos dois *standards* de gestão de aplicações, o OSEK/VDX e o AUTOSAR (*Automotive Open System Architecture*).

O terceiro capítulo é dedicado à apresentação da proposta de implementação para o sistema de comunicação tolerante a falhas e à escolha do sistema de gestão de aplicação utilizado.

O quarto capítulo apresenta a implementação da aplicação de gestão de comunicação, focalizando-se na integração de todas as soluções propostas no capítulo anterior.

O quinto capítulo está reservado para as conclusões do trabalho, onde é feito um resumo crítico do trabalho desenvolvido, apontando sugestões para o trabalho futuro que possa melhorar a plataforma.

Capítulo 2

Enquadramento

Neste capítulo pretende-se fazer uma revisão dos sistemas de comunicação tolerante a falhas para aplicações de segurança crítica no veículo. Devido à existência de muitos sistemas de comunicações no veículo, foi utilizado como exemplo o sistema de travagem, sendo uma aplicação de segurança crítica relevante. Serão primeiramente apresentados neste capítulo os tipos de arquiteturas mais usados nos sistemas de comunicação tolerante a falhas. Posteriormente, serão descritos os protocolos usados na troca dos dados. No final do capítulo, serão descritas algumas metodologias de desenvolvimento de aplicações no veículo, descrevendo o conceito de sistema operativo tempo real para microcontroladores.

Todos os itens referidos acima são descritos com mais pormenor para que se possa ter conhecimento dos desenvolvimentos existentes nas diversas áreas, sendo este um ponto importante para o desenvolvimento de uma proposta de uma plataforma que esteja enquadrada às já desenvolvidas. No entanto, também é importante ter em conta as limitações existentes, pelo fato de a plataforma a desenvolver tem o objetivo de vir a ser base para futuros trabalhos desenvolvidos por alunos do MIEEC.

2.1 Introdução

Os avanços no ramo da Electrónica Automóvel têm vindo a ter uma ênfase cada vez maior na área da Engenharia Eletrotécnica, pois o seu uso democratizou-se devido ao aumento da capacidade de efetuar complexas operações de controlo em unidades de *hardware* de tamanho reduzido, tudo isso a preços competitivos.

Esses avanços têm como consequência o aumento das variáveis a controlar, a quantidade de sinais a adquirir pelo sistema e a sua transmissão para os ECUs, descentralizando cada vez mais as soluções criadas. A descentralização das arquiteturas, e o uso de componentes eletromecânicos levou ao desenvolvimento de sistemas *x by wire*.

A tecnologia *x by wire* consiste no desenvolvimento de soluções para diversas aplicações eletrónicas tradicionalmente mecânicas. Esta tecnologia tenta também melhorar as limitações dos sistemas representados pelo "x", recorrendo a unidades electrónicas com controlo mais refinado.

A necessidade de implementar sistemas *x by wire* advém do crescimento exponencial da capacidade da eletrónica desempenhar funções que antes eram impossíveis devido as inúmeras restrições de ordem mecânica. Outra área onde o *x by wire* tem revelado grande crescimento, reside na sua inclusão nos veículos híbridos e elétricos, trazendo funções específicas, inexistentes nos automóveis tradicionais. Dentro dos sistemas *drive by wire* existem três grandes categorias:

Brake by wire - Sistema de travagem;

Steer by wire - Controlo de direção;

Throthle by wire - Controlo de borboleta do motor;

Para os objetivos do trabalho, será nesta secção descrito o resultado da pesquisa bibliográfica feita às arquiteturas de sistemas *x by wire*, contextualizando para o sistema *brake by wire*. Do resultado dessa pesquisa resultará um conhecimento das arquiteturas de comunicação dos sistemas *brake by wire*, ao nível da tolerância a falhas.

2.2 *X by Wire*(XBW)

A letra X no ambiente automóvel engloba todas as ações do condutor, sendo elas acelerar, travar ou o controlo de direção (*steering*). Atualmente, muitos veículos já estão equipados com sistemas *drive by wire* combinados com um sistema de *backup* mecânico. O objetivo do *x by wire* atual é a transferência do controlo para componentes electrónicos.

Na figura 2.1 da direita temos um sistema dito tradicional, onde a ligação entre o actuador e os diversos componentes do sistema é feito por meio mecânico (i.e. por meio de cados, veios, óleos). Nesse mecanismo, a ordem dada pelo condutor é diretamente transmitida aos actuadores sem nenhum controlo da dinâmica do veículo e nenhuma interação entre os sistemas de direção, travagem e aceleração. Num sistema *by wire* (figura da esquerda), os sistemas podem estar todos ligados por uma rede. A ordem do utilizador é registada num dispositivo que converte o estímulo físico do condutor em informação para o barramento. A informação proveniente da ordem do condutor é complementada com a informação proveniente dos diversos sensores, sendo estas enviadas diretamente para um ou vários controladores que processam a informação recebida e enviam para os actuadores as ordens resultantes da ponderação entre o que o condutor deseja e a dinâmica do veículo.

Esta nova abordagem remove assim todos os componentes hidráulicos e puramente mecânicos do sistema. Esta substituição leva à perda da interação mecânica entre o utilizador e a zona de atuação. Como já referido acima, nesta topologia, em geral, a ordem do utilizador não é diretamente enviada para os atuadores, existindo uma fase de processamento dos sinais adquiridos pelos diversos módulos do sistema.

Na Figura 2.2 encontra-se esboçado um esquema geral de um sistema *drive by wire*. A unidade de interface com o utilizador tende a ser exatamente idêntica ao sistema tradicional onde, os

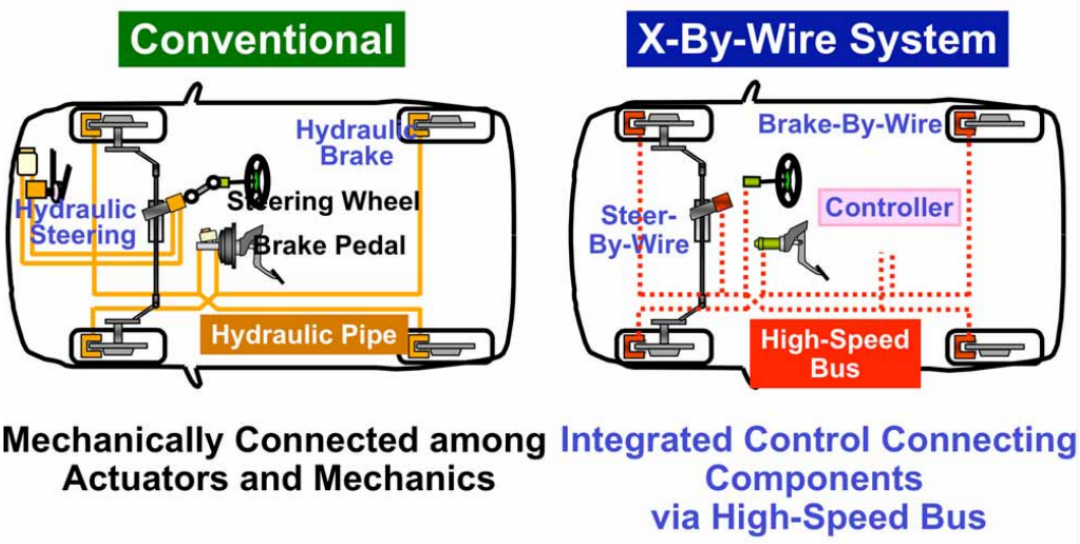

Figura 2.1: Diferenças entre um sistema tradicional (esquerda) e um sistema *drive by wire* (direita) [1]

comandos do utilizador são adquiridos através de sensores instalados nos diversos componentes da interface (i.e. volante, acelerador, travão).

O sistema, simula simultaneamente o feedback das sensações com o objetivo de serem idênticas às ressentidas nos sistemas mecanicos (i.e. sensação do pedal de travão, sensação de endurecimento do braço de direção), com o auxílio de simuladores de *feedback* do comportamento do veículo [2]. Este sistema, já amplamente utilizado em engenharia aeronaval, tem vindo a ser implementado aos poucos em automóveis comerciais modernos. [17]

As vantagens destes sistemas são:

1. Peso reduzido do sistema;
2. Controlo mais preciso;
3. Manutenção mais simples;
4. Possibilidade de interação entre vários sistemas;
5. Custos reduzidos;
6. Remoção de componentes hidráulicos que podem ser nocivos ao ambiente.

Em sistemas desta natureza, é necessário ter em atenção os aspetos de funcionamento pleno e seguro. Normas europeias e internacionais têm sido propostas para atender aos aspectos de desenvolvimento de aplicações XBW, em busca de mecanismos de diminuição da taxa de falhas no sistema e a diminuição dos efeitos das falhas.[18] [19]

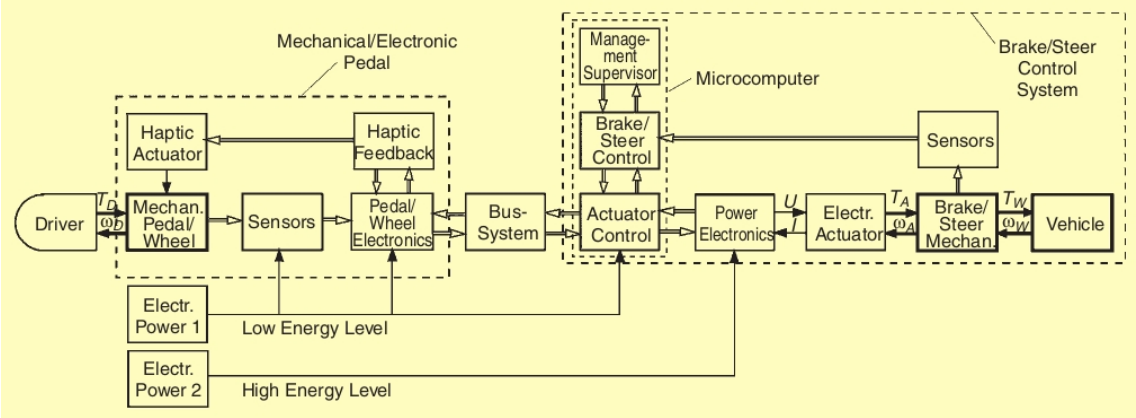

Figura 2.2: Esquema básico de um sistema *drive by wire* [2]

2.2.1 *Brake by Wire* (BBW)

O sistema de travagem *by wire* enquadra-se nos sistemas referidos acima e tem sido uma aposta crescente das construtoras automóveis. Existem já automóveis que englobam tecnologia BBW, sobretudo os carros de alta gama e veículos híbridos e eléctricos.

Num sistema de travagem BBW, a energia realizada pelo condutor no pedal de travão no ato da travagem não é transmitida para as pinças situadas nas rodas. Essa força de travagem, nas rodas, é aplicada por meio de atuadores elétricos, ou na atuação de circuitos hidráulicos. Com esta topologia, este sistema oferece diversas vantagens, tais como [4]:

1. Remoção de componentes mecânicos e complexos como o servo-freio;

2. Melhorias na eficiência e estabilidade de travagem, graças a um controlo mais rápido e preciso do binário de travagem;

3. Rapidez e simplicidade em estabelecer listas de diagnóstico exaustivas do sistema;

4. Facilidade em implementar funcionalidades tais como o ESP (*Electronic Stability Program*), o ABS (*Anti-lock Braking System*) e o BA (*Brake Assistant*), sem adição de componentes mecânicos ou hidráulicos;

5. Ganhos de espaço e diminuição do peso do sistema;

6. Redução do impacto ambiental associado aos sistemas tradicionais (i.e desaparecimento de circuitos hidráulicos);

7. Possibilidade de maior interação entre sistemas (i.e direção, controlo de potencia transmitida para as rodas).

2.2.2 Arquiteturas *Brake by Wire*

Nesta subsecção serão apresentadas algumas arquiteturas ao nível dos sensores, das redes de comunicação, da plataforma de processamento e dos atuadores frequentemente usados nos sistemas *brake by wire*, para um enquadramento do que existe na área dos sistemas *brake by wire*.

Arquitetura Multi Barramento

Descrição: A arquitetura proposta em [3] (Figura 2.3) consiste em quatro atuadores *wheeel brake control unit* situados nas rodas, controlados por: uma unidade de controlo de travagem eletrónica; um módulo de emulação do pedal de travão; e duas fontes de alimentação geridas por uma unidade de *hardware* composta por díodos. A comunicação dos módulos situados nas rodas e o FOU (*Fail Operational Unit*) é feita através de uma rede Flexray duplicada [20].

A central de controlo de travagem é composta por duas *fail silent unit*. Com esta configuração, no caso de uma das duas unidades de controlo entrar em falha, essa passa para o modo silencioso e a unidade de supervisão toma o seu lugar como unidade de controlo. Esta arquitetura incorpora também redundância ao nível dos dados, devido à existência de três barramentos CAN (*Controler Area Network*). O primeiro barramento envia informação para a unidade de controlo de travagem com a informação da pressão exercida no pedal de travão e recebe a informação de *feedback* para o emulador de pressão situado no BPIU(*Brake Pedal Input Unit*). Os barramentos CAN 2 e 3 têm como principal objetivo manter a funcionalidade de desaceleração, caso venha a ocorrer falha do FOU. Os atuadores situados nas rodas da esquerda recebem a informação de pressão do pedal proveniente do barramento 2 sendo a informação proveniente do barramento 3 enviada aos atuadores do lado direito. Essa configuração garante a função de desaceleração, mesmo que ocorra uma falha do FOU, os atuadores de travagem irão receber a ordem de desaceleração.

O módulo do pedal é composto por um emulador de sensação (BPFE) e uma interface eletrónica do pedal (BPIU), sendo o módulo a interface entre o utilizador e todo o sistema. O emulador de sensação deve ter uma arquitetura com grau de confiabilidade elevado[21], para poder oferecer ao utilizador a informação da dinâmica de travagem. O BPIU consiste numa configuração TMR (Triple Modular Redundancy) Dual [22], com três pares de sensores e módulos de processamento de sinal. A informação dos sensores chega a três votadores que enviam o resultado da votação para os barramentos CAN, de forma a evitar a situação de falha do sistema por falta de informação.

O módulo *fail silent unit* de travagem das rodas consiste em dois sensores que medem a velocidade das rodas e num atuador elétrico de travagem. O controlo do módulo de travagem avalia se a força pedida na travagem está dentro dos valores corretos, se a gama de valores for válida, então, o módulo envia para a rede a informação da força de travagem aplicada na roda.

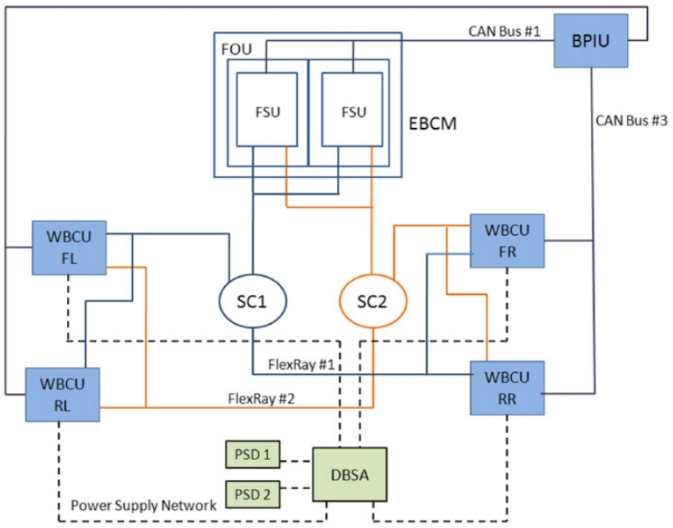

Figura 2.3: Esquema da arquitetura proposta em [3]

Fail-Operational-Unit: O bloco de *Fail-Operational-Unit* é composto por duas unidades de processamento *fail silent unit*, em que uma é o controlador principal e a outra funciona como supervisora. As unidades comunicam uma com a outra por meio de comunicação série com o objetivo de sincronizar os valores obtidos. Cada unidade tem a sua própria fonte de alimentação, o que faz com que o sistema tolere uma falha de uma das fontes de alimentação sem perder a funcionalidade de controlo.

Rede de Comunicação: A rede de comunicação entre unidades representada na figura 2.4, ilustra a forma como os microcontroladores estão ligados à rede de sensores e atuadores. A ligação entre módulos é feita através de uma rede de comunicação CAN redundante.

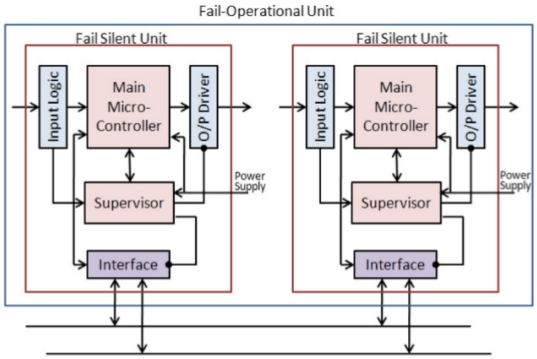

Figura 2.4: Esquema básico de um sistema *Fail Operational Unit* com duas *fail-silent units* [3]

Arquitetura Duplicada

Descrição: A arquitetura *duo-duplex* proposto em [2] (Figura 2.5) apresenta um controlador central que recebe a informação proveniente das unidades dos pedais, enviada por meio de barramentos CAN distintos.

A informação da força de travagem é enviada para as unidades de travagem das rodas por meio de um barramento partilhado. A informação enviada tem como destino as unidades de controlo do pedal. O barramento de comunicação partilhado confere ao sistema a capacidade de desempenhar as funções de desaceleração, mesmo que a existência de uma falha pontual retire o controlador central do sistema.

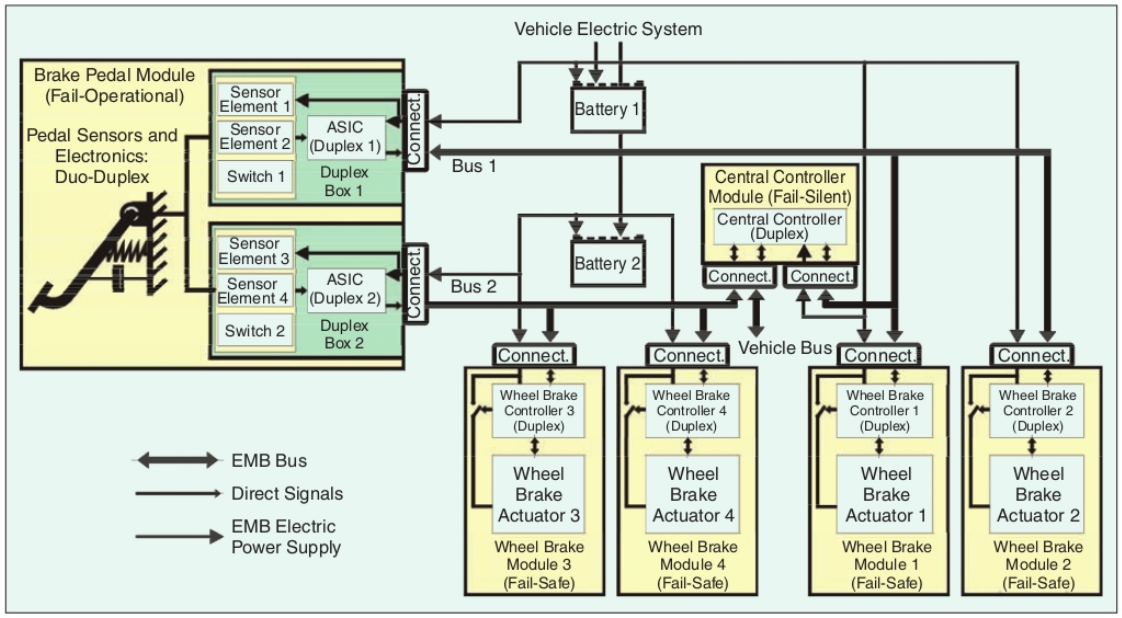

Figura 2.5: Esquema da arquitetura proposta em [2]

O módulo do pedal apresentado é composto por quatro sensores dispostos em duas unidades distintas. O número de sensores responde à necessidade de redundância num módulo crítico em termos de segurança. Depois dos sinais serem processados pelas unidades de processamento, esses sinais são enviados para um votador redundante que compara os valores obtidos em cada módulo. O valores comparados são enviados para o barramento CAN. Caso o resultado do votador seja discordante, o sistema é desligado, para não induzir falha no sistema.

Unidade de Processamento: O controlador central desta arquitetura é idêntico ao proposto em [3]. De forma análoga ao referido acima, esta arquitetura dispõe de uma unidade FOU ligada a uma rede de comunicação redundante.

Arquitetura Simplificada

Descrição A arquitetura proposta representada na [4] figura 2.6, de forma análoga às arquiteturas anteriores inclui atuadores eletromecânicos nas quatro rodas do veículo. Os módulos situados nas rodas são compostos por: dois sensores de velocidade; uma unidade atuadora eletromecânica de travagem e dois controladores de interface para o barramento.

O módulo do pedal de travão inclui três sensores de posição e velocidade de pressão e um par de microcontroladores que envia o resultado da votação a dados referentes ao pedal para a rede. Esta configuração acrescenta dois sensores de velocidade e de ângulo do braço de direção à aplicação de controlo.

A alimentação de todo o sistema é feita com o auxilio de três fontes de tensão, colocadas em cada extremidade da configuração. Com essa disposição, a funcionalidade de travagem é preservada mesmo que duas das três fontes entrem em modo de falha.

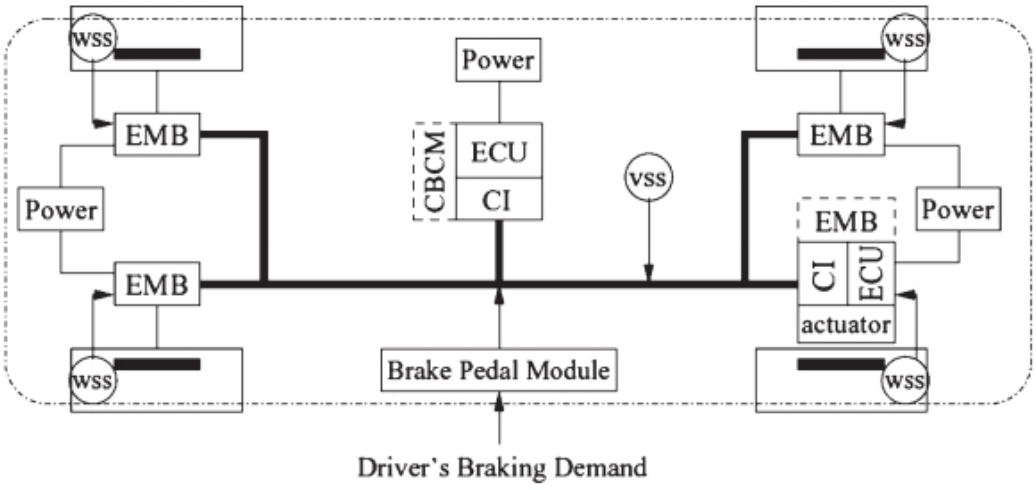

Figura 2.6: Esquema da arquitetura proposta em [4]

2.3 Rede de Comunicação

Após apresentar sumariamente algumas arquiteturas de sistemas de comunicação tolerante a falhas, serão abordadas neste documento os sistemas de comunicação utilizados nessas arquiteturas.

Da análise feita à secção anterior, conclui-se que as redes mais usadas em comunicações tolerante a falhas para veículos são o *Flexray* e o CAN. No entanto, a rede de comunicação *Flexray* não será alvo de análise na secção das redes de comunicação, pois não se enquadra no contexto académico. O *Flexray* corresponde a uma solução com um nível de tolerância a falhas muito superior ao CAN, onde o mecanismo de acesso ao meio baseia-se num modelo TDMA com precisão

de microssegundos na troca de dados. No entanto, sendo uma rede proprietária, os custos de desenvolvimento são enormes e desenquadrados com o perfil académico. Outra vertente que coloca um grande entrave ao uso do *Flexray* reside no nível de conhecimento do protocolo, é uma rede com elevado nível de dificuldade no desenvolvimento.

2.3.1 Introdução

Nos sistemas de comunicação tolerante a falhas, a componente do protocolo de comunicação é crucial. A escolha do protocolo de comunicação adequado deve ter em conta os seguintes parâmetros: [23]

1. Flexibilidade;
2. Custo;
3. Largura de Banda;
4. Determinismo Temporal;
5. Confiabilidade;
6. Requisitos Temporais (*Hard/Soft Real Time*);
7. Modularidade;
8. Facilidade na expansão da rede.

Dentro das redes de comunicação existentes o mercado, o CAN é uma das mais utilizadas. É uma rede de elevada tolerância a falhas e de elevada integridade dos dados. A qualidade do CAN reside na facilidade de incluir estações, pois é protocolo orientado as mensagens, onde somente existe identificador para distinguir o conteúdo das mensagens trocadas, tornando o número de estações completamente transparente para a rede.

De referir que na escolha de uma rede adaptada para o uso em aplicações tolerante a falhas, o *Flexray* também seria uma rede viável para o desenvolvimento, não será no entanto abordado nesta secção pelos motivos referidos no início da secção

Existem também implementações *time-triggered* do CAN. Para aplicações com requisitos de segurança críticos, o fator de utilização de rede com o envio de mensagens orientado a eventos ultrapassa os 30%. Essa limitação é necessária para evitar uma sobrecarga da rede em casos onde venha a ocorrer troca de mensagens de erro e potenciais retransmissões de mensagens anteriormente assinaladas como corrompidas. As implementações *time-triggered*, graças à distribuição temporal das mensagens é possível obter um melhor rendimento da rede. Serão agora descrito dois dos mais relevantes protocolos *time triggered* com base no protocolo CAN, sendo eles o TT-CAN (*Time-Triggered CAN*) e o FTT-CAN (*Flexible Time-Triggered communication on CAN*).

Tabela 2.1: Formato das tramas básicas no CAN

Campo	Tamanho (bits)	Função
SOF - Start of Frame	1	Representa o início de uma nova trama
ID - Indentifier	11	Identifica a mensagem e a prioridade da mesma
RTR - Remote Transmition Request	1	Distingue o envio de uma mensagem de um pedido de mensagem
IDE - Identifier Extension	1	Bit que distingue uma trama Base com o identificador de 11 bits de uma trama *extended* de 29 bits.
r0	1	Bit reservado.
DLC - Data Lenght Code	4	Numero de bytes de dados que serão enviados (0-8).
DATA - dados	[0-64]	Dados a enviar.
CRC - Cyclic Redundant Code	15 + bit delimitador	Codigo CRC enviado para posterior confirmação da veracidade dos dados.
ACK - Acknologment	1 + bit delimitador	O transmissor coloca o bit no estado recessivo. É colocado no estado dominante pelo receptor da mensagem em caso de sucesso na recepção.
EOF - End of Frame	7	Delimita o fim da trama.

2.3.2 CAN

Introdução

CAN é das redes mais utilizadas na indústria automóvel. No CAN, as estações não têm identificador, sendo um protocolo orientado às mensagens. No CAN são as mensagens que contêm um identificador único. O protocolo CAN é não determinístico, pois são as estações que decidem quando querem transmitir para a rede, podendo existir várias estações a tentar transmitir em simultâneo.

Formato das mensagens

As tramas do CAN podem ser do tipo básico ou *extend*, sendo que a diferença entre os dois tipos de tramas reside no fato das tramas *extended* poderem conter identificadores de 29 bits, sendo que o tamanho do identificador das tramas básica de 11 bits. O formato de mensagens *extended* surgiu no CAN 2.0 B para se adequar à elevada quantidade de mensagens trocadas nos sistemas distribuídos modernos. A tabela 2.1, juntamente com a figura 2.7, descreve o formato das tramas CAN de forma sucinta, descrevendo a função de cada um dos campos.[6]

Topologia do barramento

O barramento físico tem uma topologia *wired-and*. Neste caso, o valor correspondente ao barramento corresponde à operação do tipo *and* de todos os nodos. O barramento CAN tem dois estados: o recessivo (valor lógico "um") e o dominante (valor lógico "zero"). Caso nenhum nodo

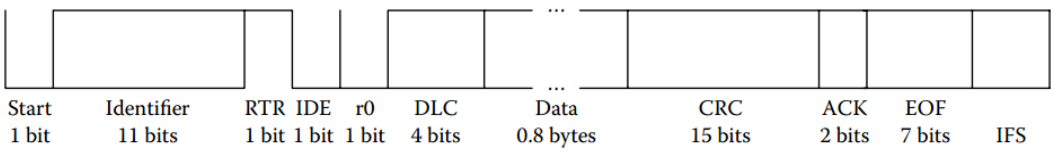

Figura 2.7: Formato da trama básica CAN [5]

da rede envie o bit dominante, a rede continua no estado recessivo, sendo que, no caso de um dos nodos transmitir um bit dominante, a rede encontra-se no estado dominante. Se um nodo envia um bit recessivo para a rede e recebe um bit dominante, interrompe imediatamente a transmissão e passa a receber a trama que está a ser transmitida. Graças a esse mecanismo de acesso ao meio, não existe alteração do bit da mensagem enviada, sendo por isso chamado de mecanismo de acesso ao meio não destrutivo (CSMA/BA Carrier Sense Multiple Access whith Bus arbritration). [5]

Quando os nodos iniciam a transmissão, a mensagem com prioridade maior ficará com o acesso ao barramento para poder enviar a mensagem e os outros nodos com mensagens de prioridade inferior passam a ouvir o barramento na espera da mensagem enviada. Na figura 2.8 podemos ver um exemplo de acesso ao meio por parte de três nodos que desejam comunicar ao mesmo tempo para o barramento. A codificação das mensagens é feita para a mensagem com identificador mais baixo (maior quantidade de zeros) ganhe o acesso ao meio, em detrimento de outro nodo que tenha uma mensagem com um identificador superior na transição. No exemplo da figura 2.8, o nodo 1 transmite o valor recessivo no quinto bit do identificador, quando o barramento está no estado dominante, por consequente, o nodo 1 para imediatamente de transmitir, recebendo a mensagem proveniente do barramento. No final da transmissão, as estações concorrem novamente pelo acesso ao meio de forma análoga à transmissão anterior.[6]

Tratamento de erros

Ao contrário de outras redes, o protocolo CAN não usa confirmação direta da recepção das mensagens, no entanto, em alternativa assinala os erros logo que esses ocorrem. Para detetar erros, o CAN implementa 2 mecanismos na camada 2 do modelo OSI (Ligação de Dados), sendo esses:[6]

- *Cyclic Redundancy Check* (CRC): A estação que transmite a mensagem calcula a *Frame Check Sequence* (FCS) a partir do campo de dados e envia no campo CRC da mensagem. A estação que recebe a mensagem calcula o seu próprio FCS e compara com valor do campo CRC da mensagem recebida.

  A verificação do CRC consiste num campo de 15 bits, correspondente à aplicação do polinómio $0x62CC$ à trama desde o SOF até ao fim do campo de dados. Este mecanismo de deteção de erros permite alta integridade das mensagens, adquirida pela relação do tamanho do código com o tamanho total da mensagem.

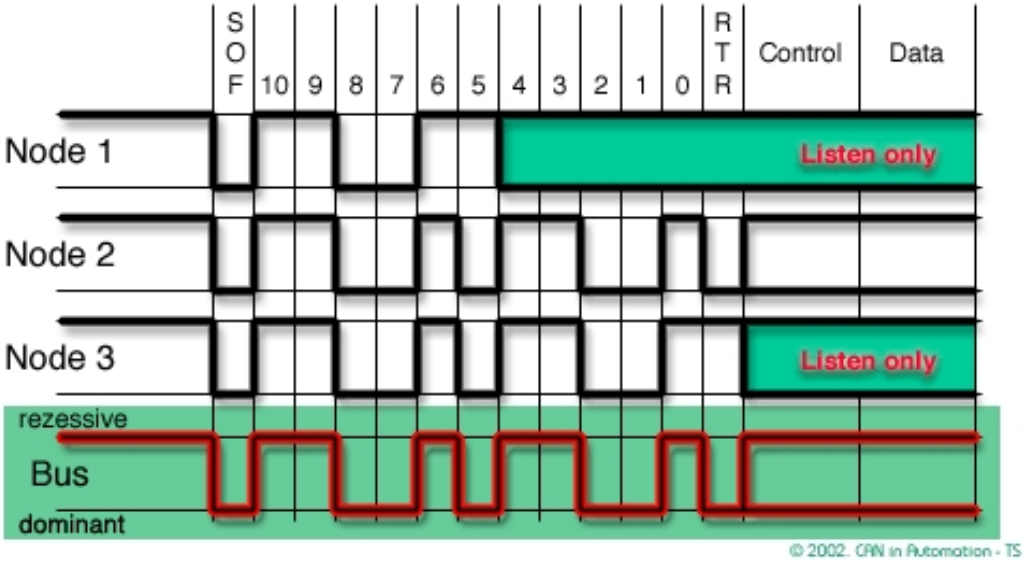

Figura 2.8: Esquema do acesso ao meio em barramentos CAN [6]

- Erros de ACK: Esse mecanismo indireto de confirmação de mensagens, permite à estação que transmite, saber se a mensagem chegou a ser recebida. No momento da transmissão, o campo de ACK na trama de envio é colocado no estado recessivo pelo emissor. Os receptores, ao receber a mensagem, colocam o bit de ACK no estado dominante para indicar a recepção da mensagem.

Confinamento de erros

O protocolo CAN tem um mecanismo de isolamento dos nós com falhas, através de uma máquina de estados de erros (Figura 2.9). Existem dois contadores de erros, um para a transmissão (*Transmition Error Counter*) e outro para a recepção (*Reception Error Counter*). A máquina de estado dos erros inicia no estado *Error Active*. Com os contadores a zero, o barramento continua em modo *Error Active* e os contadores não se alteram se não receberem nenhuma mensagem de erro.

Quando acontece um erro de transmissão o contador do módulo em questão incrementa 8 unidades. No caso da recepção, o incremento do contador de erros depende do tipo de erro em questão[24]. O contador de recepção decresce de 1 unidade por cada mensagem recebida com sucesso até ao valor 0. De forma análoga ao contador de recepção, o contador das mensagens transmitidas descresse de uma unidade por cada mensagem recebida.[24]

Se o valor de qualquer um dos dois contadores ultrapassar o valor 127, a máquina de estados entra em modo *Error Passive.* No modo *Error Passive* o nodo deixa de transmitir, mas continua a receber mensagens. Quando o barramento ultrapassa o valor de 255, esse entra

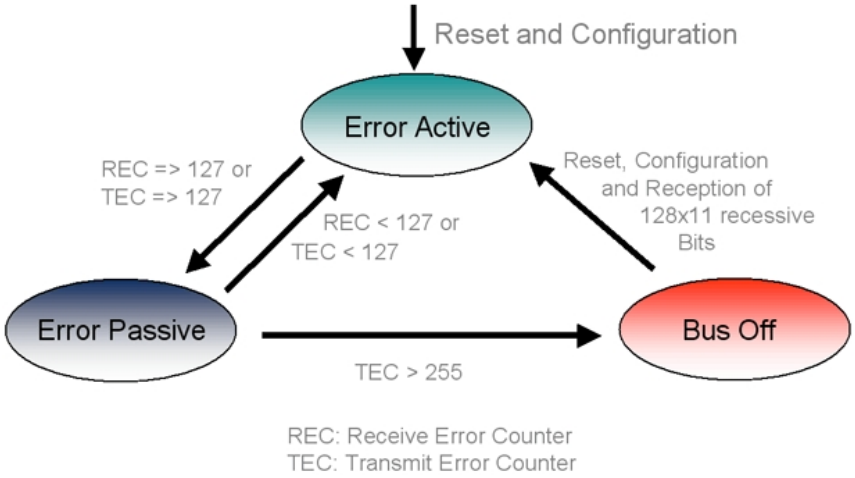

Figura 2.9: Maquina de estados de erros em CAN [6]

em BUS OFF, desligando as comunicações no nodo até a reconfiguração ou até a recepção de 128x11 bits recessivos. Nesse caso, o nodo entra novamente no estado *Error Active*.[24]

2.3.3 TT-CAN

O protocolo TT-CAN (*Time-Triggered CAN*) é uma implementação de um protocolo *time triggered* tendo como base o protocolo CAN. O seu aparecimento deve-se da necessidade de acrescentar um tipo de acesso ao meio *time-triggered* (orientado ao tempo) num protocolo essencialmente orientado a eventos.

O TT-CAN implementa um mecanismo de acesso ao meio composto por ciclos TDMA, baseada na divisão temporal das mensagens a transmitir, através do CAN ISO 11898-1 e com o suporte CSMA (*Carrier Sense Multiple Access*) para resolver conflitos entre mensagens que desejam ser enviadas na mesma janela de arbitragem.

O escalonamento de mensagens do TT-CAN é feito através de divisão temporal e é realizado *offline* durante a fase de desenvolvimento. O TT-CAN usa o conceito de tempo global. O tempo global é dado por um relógio *master* que difunde, a todos os *slaves*, o valor do relógio. O protocolo disponibiliza dois níveis de sincronização dos relógios, o nível 1 e o nível 2. No nível 1 são implementados os mecanismos básicos necessários para a difusão do tempo global para todos os nós da rede. No nível 2, em adição ao implementado no nível inferior, é feita uma sincronização de alta precisão do tempo global para um escalonamento de mensagens de alta precisão.[25]

O *master* designado envia periodicamente para todos os nodos, a referência temporal numa mensagem designada de *Reference Message* (RM). A RM marca o inicio do ciclo básico (*Basic Cycle* (BC)). O BC correspondente a uma janela temporal de tamanho fixo onde os diversos tipos de *slots* alocados na fase desenvolvimento. Os *slots* temporais ou *Transmission Column* (TC) (Figura 2.10), intervalo de tempos, que podem ser alocados para: [7]

- Mensagens síncronas específicas, o que implica que mais nenhuma mensagem possa aceder ao meio dentro da *slot*;
- Mensagens aperiódicas. Nesse intervalo de tempo definido, os nodos com mensagens assíncronas, competem pelo acesso ao meio de forma idêntica ao CAN;
- Alocar futuras mensagens, que podem vir a ser usadas no futuro no caso de uma possível expansão da rede.

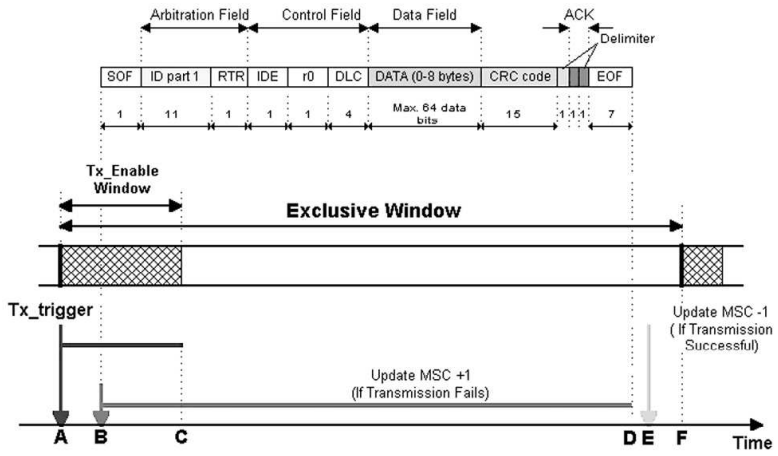

Figura 2.10: Descrição de uma *Transmition Column* [7]

Depois de todos os nodos da rede (incluído o próprio *master*) receberem a mensagem de RM, começa o *basic cycle*. Nas TC reservadas às mensagens periódicas, existe uma janela (Tx Enable Window) onde o escravo produtor da mensagem deve começar a enviar a mensagem. No final de um ciclo básico i (BC_i), o *master* envia outro RM para o início do próximo ciclo básico ($BC_i + 1$), até ao número de ciclo básicos programados no ciclo principal ($i \in [0; N-1]$).

O ciclo principal (Matrix cycle (MC)) é composto por um número N (entre 0 e 64) fixo de BC. Na figura 2.11, o exemplo apresenta um MC com 4 BC. Todos os BC têm todos o mesmo tamanho. A igualdade no comprimento dos BC's implica que o tamanho total do ciclo principal seja um múltiplo inteiro do tempo do ciclo básico. No final de um MC, recomeça novamente o mesmo ciclo principal, sendo esse repetido indefinidamente.[7]

Na fase de desenvolvimento é necessário ter em atenção que o tamanho do *slot* deve incluir a possibilidade de corrupção dos dados e, por consequente, o surgimento de tramas de erros. As tramas de erro no CAN são mensagens de tamanho máximo de 23 bits. Se todas as mensagens recebidas gerarem uma mensagem de erro, então a sobrecarga da rede levará a perdas ao nível do acesso ao *slot* seguinte. De assinalar que o TT-CAN não implementa retransmissões de mensagens

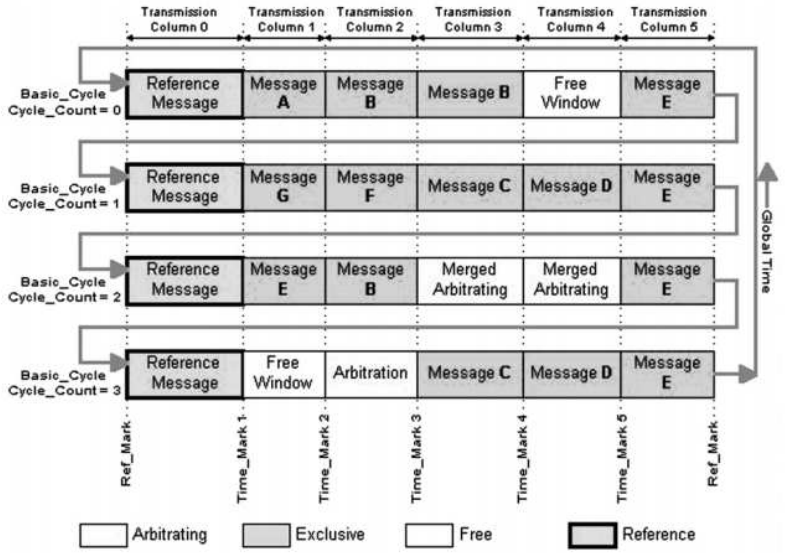

Figura 2.11: Descrição do Ciclo Principal no TT-CAN [7]

perdidas (com exceção das mensagens de referência). Pode ser ainda implementada uma janela de arbitragem de mensagens aperiódicas com mecanismo de acesso ao meio CSMA para o envio das mensagens perdidas no ciclo periódico.[7]

2.3.4 FTT-CAN

O protocolo FTT-CAN (*Flexible Time-Triggered communication on CAN*) é um protocolo que combina as componentes *time triggered* e *event triggered*.

Principio de Funcionamento

O FTT-CAN tem um mecanismo de escalonamento de mensagens centralizado no *master*. Nele recai a tarefa de enviar a informação com o escalonamento temporal das mensagens.

Periodicamente, o *master* da rede envia para todos os nodos uma mensagem denominada de *Trigger Message* (TM), que marca o início do ciclo elementar de tamanho fixo (EC). O TM contém a identificação das mensagens síncronas que serão transmitidas pelos nodos dentro do EC.

As mensagens síncronas são enviadas dentro da *Synchronous Windows* (SW), uma janela de tamanho variável. O tamanho da janela é ajustado para que, garantidamente, todas as mensagens escalonadas pelo *master* no TM sejam transmitidas dentro da SW. [8]

Na TM, é enviada a todos os nodos a informação codificada do tamanho da janela síncrona e quais as mensagens a enviar dentro da janela. Na figura 2.12 é possível ver que os bits 1, 2, 4 e 13 da mensagem estão a um. Isso indica que a SW terá o tamanho predefinido para quatro

mensagens e os *slaves* produtores das mensagens vão enviar as mensagens dentro da SW, usando o mecanismo de acesso ao meio do CAN. [8]

Entre a recepção da TM e o início da SW, existe um intervalo para o envio de mensagens orientadas a eventos chamada *Asychronous Window*. Nesta janela temporal, todos os nodos com mensagens aperiódicas competem pelo acesso ao meio de forma análoga ao protocolo CAN. O tamanho da janela assíncrona está diretamente relacionado com o tamanho da SW, sendo o tamanho de SW variável e o tamanho da EC fixa.

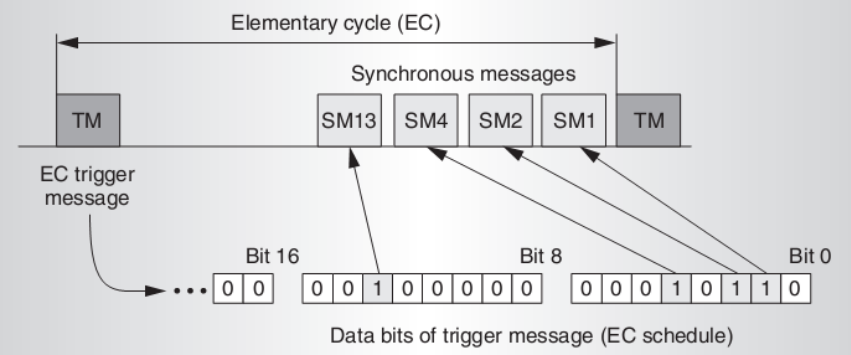

Figura 2.12: Exemplo de funcionamento do TM [8]

O protocolo também garante isolamento entre mensagens síncronas e assíncronas através do acréscimo por *software* de uma janela *idle* (α na figura 2.13) de tamanho fixo, previne que qualquer tipo de transmissão de mensagem assíncrona se sobreponha à janela de mensagens síncronas.

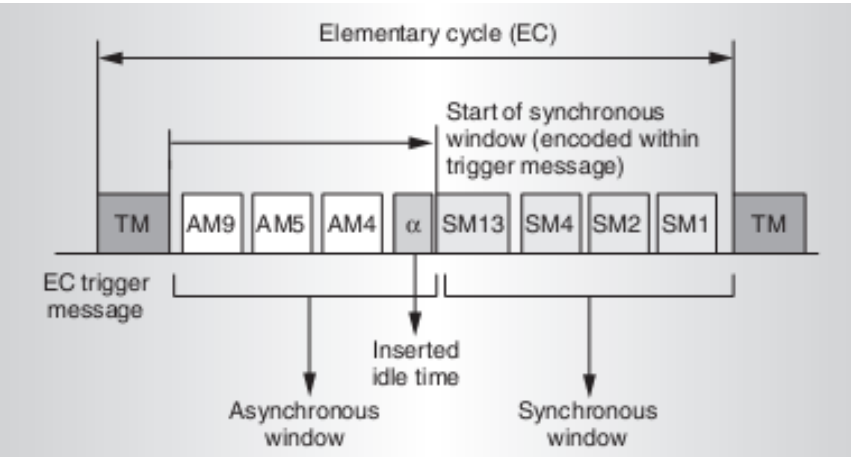

Figura 2.13: Exemplo de um ciclo elementar; [8]

Escalonamento das mensagens

A escolha das mensagens síncronas para cada EC é realizado *online* pelo *master* que é auxiliado por uma tabela de requisitos de mensagens síncronas (*Synchronous Requirements Ta-*

ble(SRT)). Nessa tabela, para cada mensagem, estão descritas o período da mensagem, a sua *deadline* e o seu desfazamento inicial, codificados em numero de EC. A SRT também descreve a prioridade e o tamanho em bytes da mensagem. O protocolo FTT-CAN prevê um escalonamento dinâmico, com suporte para mudanças da função de escalonamento, mesmo em pleno funcionamento do sistema. Esse escalonamento centralizado e dinâmico torna as comunicações totalmente transparentes para os *slaves*, pois esses limitam-se a disponibilizar as mensagens *síncronas* na SW aquando do pedido do *master*. No entanto, o escalonamento dinâmico pode levar a um processamento complexo e pesado e induzir a um *overhead* durante a execução, provocando assim um uso ineficiente da largura de banda da rede. Para contrariar esse problema, pode ser usado um plano de escalonamento. A ideia do plano de escalonamento é a de produzir o despacho das mensagens síncronas para um número N fixo de EC, não sendo idêntico a cada ciclo de planeamento. Outra solução é o uso de *hardware* suplementar para o escalonamento. Nesse caso usa-se um co-processador que comunica com o processador principal e os dois em conjunto geram o TM.[8]

2.4 Gestão das aplicações

2.4.1 Introdução

Devido à vontade de se disponibilizar uma plataforma para o desenvolvimento de aplicações de controlo no veículo, com o objetivo de facilitar a sua concepção, irão introduzir-se algumas soluções de gestão de software para microcontroladores que estão enquadradas na indústria automóvel. Em primeira instância, esta introdução remete para a ideologia do uso de um RTOS (*Real Time Operating System*) , com a descrição do OSEK/VDX (*Open Systems and their Interfaces for the Electronics in Motor Vehicles*). Após ser feita a descrição do OSEK, será explicado o conceito do consórcio AUTOSAR e da sua ideologia no desenvolvimento de *software* para os controladores.

2.4.2 Sistema Operativo Tempo Real

O uso de um sistema operativo tem como objetivo acrescentar propriedades interessantes ao desenvolvimento de software, tais como:

- Escalonamento de tarefas, sendo possível gerir o tempo de execução de cada tarefa;
- Poder escolher níveis de prioridades para as diversas tarefas;
- Respeitar requisitos de *soft/hard real time* impostos;
- Integração de *Drivers* para periféricos;
- O programador somente se preocupa com o desenvolvimento de aplicações.
- Portabilidade do código desenvolvido para outros microcontroladores.

Essas propriedades têm particular interesse no desenvolvimento de aplicações de controlo complexas, pois reduz o tempo de desenvolvimento e acrescenta confiabilidade às aplicações.

Se seguida será apresentado o RTOS (*Real Time Operating System*) OSEK/VDX(*Open Systems and the Corresponding Interfaces for Automotive Electronics*), um dos RTOS usado nos microcontroladores que equipam os automóveis.

2.4.3 OSEK/VDX

A diretiva OSEK foi fundada em 1993 por um consórcio de construtoras alemãs, tendo como principal meta padronizar as arquiteturas de controlo distribuídos dos veículos. O termo OSEK provem do alemão *Offene Systeme und für die Deren Schnittstellen Elektronik im Kraftfahrzeug*, (*Open Systems and the Corresponding Interfaces for Automotive Electronics*). A abordagem VDX (Vehicle Distributed eXecutive) foi trazida pelos fabricantes franceses do grupo PSA(*Peugeot Societé Anonime*) e *Renault*, que se juntaram ao projeto em 1994, criando o grupo conhecido atualmente por OSEK/VDX. A uniformização da arquitetura de gestão de aplicações com o uso do OSEK/VDX leva ao desaparecimento de problemas de integração de plataformas e de sistemas, pois a norma engloba procedimentos padronizados na interação entre módulos de controlo e na execução de aplicações, tais como[26]:

- Sistema Operativo (execução em tempo real do *software* dentro da ECU (figura 2.14));
- Comunicação (troca de dados entre unidades de controlo como na troca de mensagens internas);
- Gestão da Rede (configuração, determinação e monitorização dos barramentos de dados existentes).

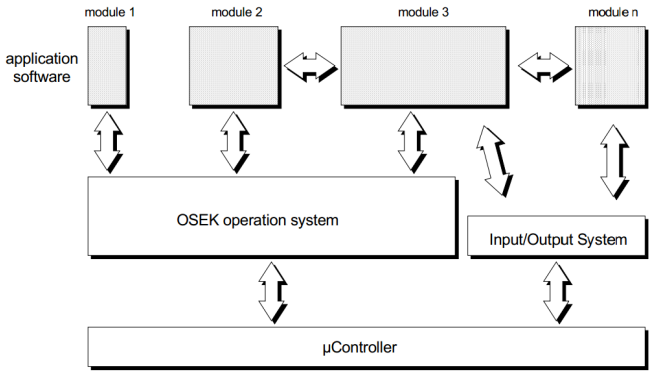

Figura 2.14: Diagrama de uma aplicação OSEK num microcontrolador [9]

Benefícios

Com a uniformização das interfaces entre as diversas unidades de controlo e com uma gestão única das comunicações, consegue-se, em toda a cadeia de desenvolvimento de uma aplicação de controlo, atingir melhorias significativas no resultado global, nomeadamente[26]:

- Diminuição do tempo e custos de desenvolvimento;
- Melhoria da qualidade das aplicações de controlo;
- Normalização das interfaces entre unidades de controlo e dentro das próprias unidades;
- Gestão global do sistema não necessita de *hardware* adicional;
- Perfeita liberdade no desenvolvimento de aplicações.

Conceito de Tarefa

Podemos entender uma tarefa como uma subdivisão em pequenas partes da resolução do nosso problema. As tarefas englobam um conjunto de funções, sendo o sistema operativo responsável por organizar a execução das tarefas de forma assíncrona e concorrente com outras tarefas, sendo assim possível criar a abstração da execução multi-tarefas.

A abstração do multi-tarefas é muito útil no ambiente das aplicações *automotive*, pois em microcontroladores de um único *core* é possível criar tarefas de controlo, tendo em paralelo, a execução de tarefas de monitorização de variáveis, a execução de tarefas de diagnóstico e a execução de tarefas de comunicação com outros módulos.

Diagrama de estados das Tarefas: As tarefas em OSEK / VDX podem ser de dois tipos: básicas e estendidas. As tarefas básicas podem ser entendidas como funções que somente libertem o acesso ao processador quando terminam ou no caso do sistema operativo entregar o processador a uma tarefa de prioridade maior, mesmo que a tarefa esteja em execução. As tarefas estendidas são baseadas nas básicas, mas têm a particularidade de poder entrar num estado de espera de eventos, com a possibilidade de disponibilizar o recurso CPU a outra tarefa[9].

As tarefas do tipo básico podem ter os seguintes estados dentro do sistema operativo(figura 2.15):

***Running*:** Neste estado, a tarefa em questão tem o acesso ao CPU, sendo possível executar as suas instruções. Só pode existir uma tarefa em estado *running* de cada vez.

***Ready*:** Tarefa em espera que a chamada do escalonador coloque a tarefa no estado *running*.

***Suspended*:** Uma tarefa suspensa está à espera de ser ativada, logo não compete pelo acesso ao CPU.

As tarefas do tipo estendidas têm os mesmos três estados, acrescentando um quarto, o estado **Wait**. Este estado consiste numa tarefa que não pode ser executada, pois está a espera de um evento para despertar. [9].

Transições entre tarefas: As transições dos diversos estados são feitas através de chamadas de funções da API do OSEK e de ações do gestor de tarefas, através de eventos ou alarmes. A tabela 2.2 descreve as transições entre estados tanto para as tarefas simples como as tarefas estendidas.

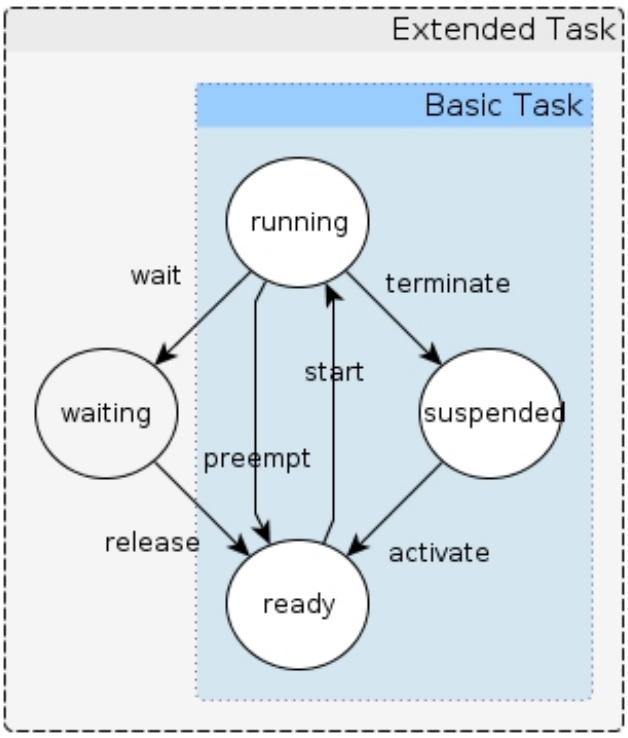

Figura 2.15: Diagrama de estado das Tarefas em OSEK (adaptado de [9])

Prioridade entre tarefas: As tarefas em OSEK, são principalmente diferenciadas pela ordem de prioridade. A tarefa com ordem de prioridade 0, corresponde à tarefa com menor prioridade. As prioridades seguintes estão ligadas a tarefas de prioridades superiores. A atribuição das prioridades na norma OSEK são estáticas, isto é, a hierarquia de prioridades entre tarefas é feita antes da compilação e não pode ser alterada na execução do programa.

Gestor de Tarefas: A gestão de tarefas no OSEK é feita com base nas várias propriedades das tarefas tais como:

- As tarefas serem básicas ou *extend*;
- A preemptividade das tarefas;
- A sua prioridade;
- A gestão das interrupções;
- Alarmes associados.

O escalonador das tarefas é quem escolhe, pelos critérios impostos na fase de configuração qual a tarefa que deve estar em execução naquele momento. O escalonador é invocado sempre que:

Tabela 2.2: Transição entre tarefas no OSEK/VDX [9]

Transição	Estado inicial	Estado Final	Descrição
activate	*suspended*	*ready*	Uma tarefa do sistema operativo é posta no estado *ready* através de uma chamada da API. Quando a tarefa inciar, será pela primeira instrução;
start	*ready*	*running*	A tarefa de maior prioridade da lista de tarefas no estado *ready* vai passar a ser a próxima a executar;
wait	*running*	*wait*	Uma tarefa que esteja a espera de um evento coloca-se no estado *wait* através de uma chamada da API. Permace nesse estado até o depoletar do evento em espera;
release	*wait*	ready	Quando o evento ocorre, a tarefa em espera volta para a lista de tarefas disponíveis para o acesso ao CPU.
preempt	*running*	*ready*	O escalonador decide retirar a execução da tarefa, ainda antes dela terminar.
terminate	*running*	*suspended*	A tarefa em questão invocou o serviço de terminar a tarefa entrando em modo suspenso e não voltando a competir pelo processador.

- Uma tarefa preventiva está ativa e uma tarefa de maior prioridade passa para o estado *ready*;
- Na chamada de funções da API OSEK;
- Depois de uma tarefa terminar;
- Quando uma tarefa do tipo *extend* passa para o estado *wait*;
- Depois da ativação de um alarme;
- No despoletar de um evento.

O escalonador, quando invocando, escolhe sempre a tarefa de maior prioridade em espera para o acesso ao processador. A escolha de uma tarefa de prioridade superior em detrimento de outra com prioridade inferior pode vir a suceder mesmo que a tarefa ainda não tenha terminado a sua execução. O único caso onde isso não sucede é no caso de a tarefa não ser preemptiva.

Uma tarefa não preemptiva não pode ser interrompida no seu decorrer de forma a deixar o acesso ao processador para uma tarefa mais prioritária. Uma tarefa preemptiva, pelo contrário pode ver a sua execução interrompida em qualquer ponto, pois existe uma tarefa de maior prioridade em espera para executar.

Gestão de Interrupções

As interrupções em OSEK são idênticas às interrupções do próprio microcontrolador, sendo possível, unicamente no ISR (*Interrupt Service Routine*) do tipo 2, chamar funções da API do

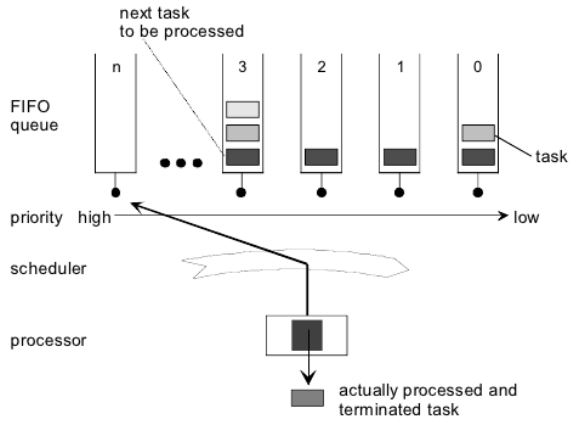

Figura 2.16: Esquema de gestão das Tarefas em OSEK [9]

OSEK. Existe no entanto uma interface própia do RTOS, para a gestão das interrupções. No final da rotina de interrupção, a tarefa que estava a ser executada antes da chamada do ISR continua a sua execução. Pode no entanto suceder que a tarefa em execução antes da chamada do ISR ser colocada no estado *wait* devida a uma chamada de funções da API do OSEK que coloque outra tarefa em execução.

Em secções críticas, é possível desabilitar todas as interrupções ou somente as interrupções do sistema operativo, para que essas não despoletem a meio dessas secções críticas[9].

Eventos

Os eventos são mecanismos associados a tarefas do tipo *extend* do sistema operativo OSEK. Através dos eventos, é possível colocar tarefas à espera de eventos específicos para prosseguir a execução da tarefa, que cede o CPU para outras tarefas serem executadas. A tarefa é colocada no estado *waiting*, onde não concorre para aceder ao CPU. Essas tarefas ficam no estado *waiting* até ao momento em que um objeto gere um evento que altera o estado da tarefa de *waiting* para *ready*, estando assim novamente disponível no gestor de tarefas.

Alarmes

Os alarmes disponibilizam serviços adequados à gestão de acontecimentos periódicos. Os alarmes OSEK/VDX contêm diversos serviços sendo eles:

- **Ativar tarefas** – Podem ser associados a transição das tarefas do estado *suspended* para o estado *wait*;
- **Despoletar eventos** – Associados ao objeto alarme está um evento que por sua vez está associado a uma tarefa;
- **Chamar funções do tipo (ALLARMCALLBACK()) –** As funções de ALLARMCALLBACK são funções tratadas como chamadas de ISR tipo 2. São rotinas de tamanho reduzido;

Os alarmes têm associados contadores que são configurados a partir dos *timers* disponíveis pelo microcontrolador. Os alarmes podem ser chamados de forma periódica ou única, sendo somente possível alterar os valores cíclicos do alarme quando esses não estiverem em execução. [9].

2.4.4 AUTOSAR

A parceria AUTOSAR nasce da associação dos principais fabricantes de veículos e dos fabricantes de componentes. Os objetivos dessa aliança são a "de desenvolver e estabelecer, na prática, um *standard* para aplicar na indústria automóvel " que sirva as " infraestruturas básicas para gestão, o desenvolvimento de funções para aplicações futuras e a criação de standards de *software*."

A necessidade crescente de uniformização aparece com o aumento da complexidade dos sistemas embarcados no automóvel. Essa complexidade levou ao aumento do uso de componentes de diversos fabricantes.

O uso de componentes de diversas marcas implica o desenvolvimento de interfaces para a comunicação entre módulos. Os acréscimos de tempo e de recursos para a criação de interfaces ligados ao tempo de aprendizagem inerentes a conhecer as particularidades de cada fabricante, diminuem o foco de desenvolvimento de aplicações. O AUTOSAR criou um princípio em que todos os fabricantes tinham de adaptar os seus produtos a um modelo comum definido por uma norma.

Aproveitando vários trabalhos já feitos na componente do desenvolvimento de software, tais como o OSEK e o HIS (Hersteller Initiative Software), o AUTOSAR criou uma unificação da metodologia de criação de software para os veículos.

O AUTOSAR divide a sua ação em três componentes principais, sendo elas a estandardização das camadas de *software* para os ECU, a estandardização no desenvolvimento do *software* e a estandardização dos métodos de desenvolvimento de aplicações.

2.4.4.1 Estandardização das Camadas de software para os ECU

O AUTOSAR propõe uma divisão por camadas da componente de *software* da ECU(figura 2.17). A divisão em camadas simplifica o desenvolvimento global do software. Existem três camadas principais: a camada básica de *software*, mais acima a camada de *Middleware*, que separa a camada básica da terceira camada, a camada de aplicação. [10]

Camada Básica de *software* - A camada Básica de *software* implementa interfaces para os recursos disponíveis no microcontrolador, para o acesso às portas de Entrada e Saída, bem como a interfaces de acesso a memória. Esta abstração torna completamente transparente o tipo de *hardware* usado. A camada é dividida em serviços para aumentar o grau de abstração do *hardware*. A presente camada é dependente do *hardware* usado, mas disponibiliza uma interface tipo AUTOSAR, tornando a variável "*hardware*" irrelevante para os desenvolvedores das camadas superiores. [10]

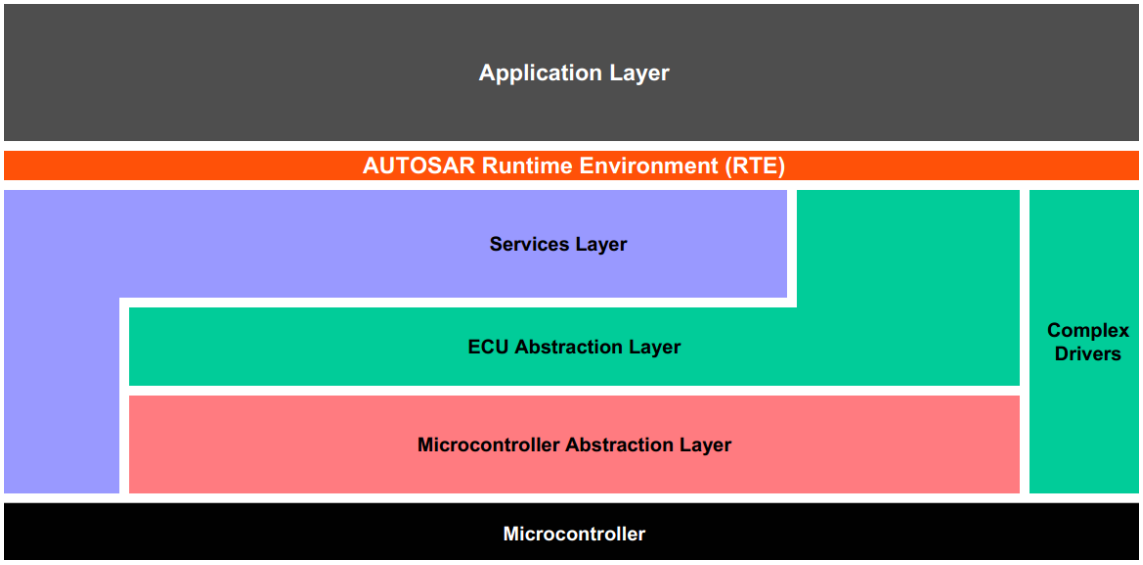

Figura 2.17: Divisão por camadas do software do AUTOSAR [10]

Runtime Environment - O *Midellware* ou RTE (*Runtime Environment*) é uma camada que separa a aplicação das infraestruturas básicas do microcontrolador. Acrescenta a isso, funções de comunicação inter-componentes. Essa é a camada que confere total transparência no desenvolvimento de *software*. [10]

Camada de Aplicação - A camada de aplicação é a camada de topo da arquitetura do ECU proposta pelo AUTOSAR. Contém o *software* da aplicação com potencial de comunicação com módulos internos e externos. Nessa camada também é executado o programa ou programas para desempenhar a função ou funcionalidade requerida. [10]

O ponte forte da arquitetura em camadas reside na possibilidade de transportar a aplicação para outro microcontrolador, modificando somente as duas primeiras camadas.[10]

2.4.4.2 Unificação da componente de interface

A unificação da componente aparece na continuidade do descrito no ponto acima. O objetivo desse ponto é a unificação das funções desenvolvidas em cada uma das camadas. As interfaces desenvolvidas em cada camada devem reger-se ao principio do AUTOSAR. Existe assim uma portalidade de código entre plataformas e entre aplicações, poupando muito tempo no desenvolvimento de funcionalidades.[27]

2.4.4.3 Estandardização no Desenvolvimento

É necessário que todos os desenvolvedores de soluções para a indústria automóvel apliquem métodos de desenvolvimento semelhantes de forma a que continue a existir a completa portabili-

dade de código de modo a facilitar a compreensão do trabalho desenvolvido por todos os intervenientes na elaboração de aplicações *automotive*.[27]

2.4.4.4 Relação entre o AUTOSAR e o OSEK

A estrutura do OSEK/VDX, foi um dos princípios seguidos pelos desenvolvedores do AUTOSAR para o desenvolvimento das camadas básicas de *software* e de *Midellware*. Pode ser referido ainda que a implementação do AUTOSAR OS [28] é em maioria baseada no OSEK/VDX.

Capítulo 3

Proposta

Neste capítulo introduz-se a proposta de implementação do trabalho a desenvolver, sendo feito um resumo da mesma. De seguida, é explicado as razões que levam à escolha do sistema proposto. Após explicar as razões, é detalhada a proposta de comunicação, explicando o seu princípio de funcionamento. Posteriormente são mostrados em que consistem os mecanismos implementados na troca de mensagens seguras na rede. Na conclusão da proposta, é proposto o sistema operativo tempo real e o microcontrolador utilizado na implementação.

3.1 Introdução

O objetivo do trabalho consiste na criação de uma plataforma, que suporte comunicações tolerante a falhas, com a finalidade de ser utilizada nos futuros veículos elétricos da FEUP(Faculdade de Engenharia da Universidade do Porto), servindo de base para futuros projetos académicos dos alunos do MIEEC (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) da FEUP
.

A proposta será dividida em duas grandes partes:

- Serviço de comunicações Tolerante a Falhas;
- Serviço de comunicações de segurança-crítica;

3.2 Resumo

A proposta apresentada (figura 3.1) consiste na implementação do sistema de comunicação tolerante a falhas. Tem suporte para um protocolo TDMA redundante com base no CAN proposto por [12]. Essa proposta inclui um suporte TDMA para envio de mensagens e uma sequência de mecanismos para a troca de mensagens por meio de dois barramentos. Para o aumento da segurança das mensagens, foi incluída na proposta uma camada de segurança de mensagens, implementando os mecanismos descritos na norma EN 50129 [16].

Para o desenvolvimento do *software* de controlo servindo de suporte ao mecanismo de comunicações tolerante a falhas proposto, será utilizado um sistema operativo para microcontroladores baseado no OSEK/VDX da empresa *Evidence*, o *Erika Enterprise & RT Druid* .

O *hardware* escolhido para a implementação da comunicação, será o dsPIC30F6014A da marca *Microchip*.

Será abaixo descrita com mais pormenor a proposta de implementação, bem como os aspectos de escolha de cada uma das soluções apresentadas.

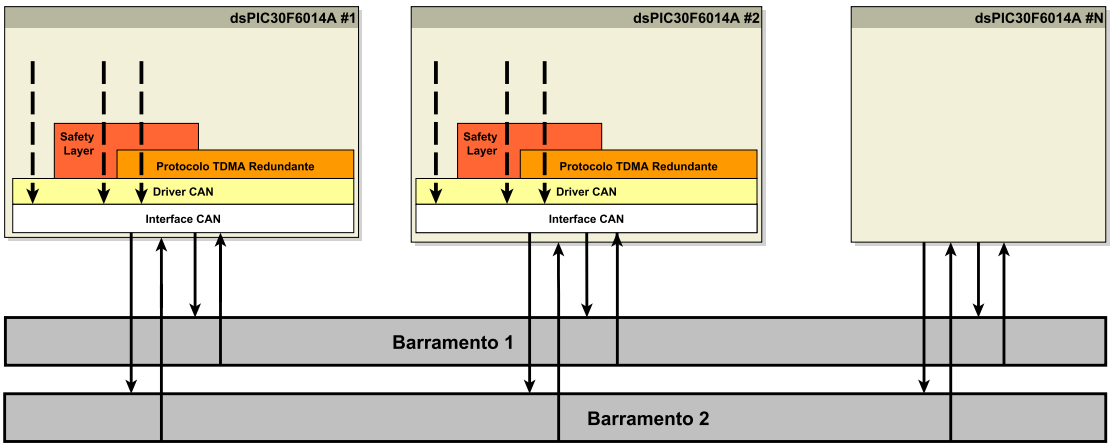

Figura 3.1: Esquema geral da proposta da solução

3.3 Mecanismo de envio de mensagens tolerante a falhas

Na indústria automóvel é necessário transmitir informação entre nodos, tendo sempre em atenção a confiabilidade das informações e a robustez da rede às diversas anomalias, tais como o ruído eletromagnético ou o corte da cablagem. As anomalias na transmissão de dados podem levar a diversos acontecimentos indesejados, que se resumem a:

- A informação transmitida não é recebida;
- A informação transmitida é recebida, mas não corresponde à informação enviada.

Em primeira instância, deve ser escolhido o protocolo de comunicação base adequado para o projeto. A escolha feita incide no CAN, sendo um protocolo orientado à mensagem com alta tolerância a falhas e alta integridade nos dados trocados numa rede onde existe facilidade em incluir novas estações sem grandes alterações na rede.

Como suporte da rede CAN mencionada acima, será escolhida uma arquitetura de comunicação redundante proposta em [12], com o uso de 2 barramentos CAN. A escolha do número de barramentos coincide com a maioria das propostas de arquiteturas presentes na literatura, correspondendo também ao número de módulos disponíveis nos microcontroladores de média/baixa complexidade.

O tipo de acesso ao meio TDMA proposto, enquadra-se as mensagens que serão trocadas na plataforma, sendo maioritariamente mensagens contendo dados relacionados com controlo de sistemas.

O mecanismo de transmissão também atende ao principio de baixa complexidade do trabalho, sendo possível implementar em microcontroladores de baixa complexidade. O princípio de troca de dados proposto, tenta atenuar os efeitos negativos de algumas limitações do CAN, sendo seguidamente feita uma descrição dessas limitações.

3.3.1 Limitação do protocolo CAN

As limitações existentes no CAN segundo [12] e [29], são identificadas e listadas abaixo:

1. Suporte para comunicações *time-triggered*;
2. Confiabilidade nas comunicações;
3. Suporte para arquiteturas com barramentos redundantes;
4. Mecanismo para resolver erros do tipo *babbling idiot*.

3.3.1.1 Suporte para comunicações *time-triggered*

Com a comunicação *time-triggered* é possível detetar a omissão de mensagem que deviam ter sido recebidas dentro tempo definido para tal. Com o uso de protocolos *time triggered* é possível otimizar o fator de utilização da rede, com o escalonamento temporal dos dados a enviar. O CAN é por natureza *event triggered*, ou seja não tem suporte para comunicações *time triggered*.

Existem no entanto diversas implementações de protocolos *time triggered* implementadas a partir do protocolo CAN, tais como o TTCAN e o FTT-CAN, referidos no capítulo do estado da arte.

3.3.1.2 Confiabilidade nas comunicações

O protocolo CAN, teoricamente, tem suporte para comunicações em bloco, isto é, quando um nodo envia uma mensagem, todos os nodos recebem corretamente a mensagem enviada, utilizando um mecanismo de *atomic broadcast*. Um dos problemas do protocolo CAN é que essa condição nem sempre é satisfeita. A exceção acontece quando os nodos recebem tramas "pseudo-erradas", devido a erros detetados no campo EOF (*end-of-frame*).

O receptor valida uma mensagem quando a sequência está correta até ao sexto bit do campo do EOF. O transmissor avalia as mensagens enviadas de forma diferente, isto é, uma mensagem só é validada se a sequência do EOF está correta no sétimo e último bit do EOF. Esses dois critérios levam a potenciais problemas, pois este critério de validação diferente das mensagens gera reações contrárias. Ou seja, pode um subconjunto de receptores detetar um erro no sexto bit do EOF e consequentemente enviar uma *flag* para sinalizar o erro na trama recebida quando

os restantes receptores já aceitaram a trama como correta, gerando uma inconsistência nos dados recebidos por todos os receptores. Esses erros originam um dos dois efeitos:

- Mensagens repetidas por retransmissão, devido à deteção de erro de um dos receptores, quando outros determinaram que a mensagem era válida;
- Erros de omissão, pois a mensagem pode não ser recebida por todos os nodos.

Na figura 3.2, [11] é possível entender os casos em que ocorre inconsistência. Na imagem A, o receptor 1 recebeu corretamente a mensagem e o transmissor não detetou erro no penúltimo bit, no entanto, o receptor 2 detetou um erro no último bit, sendo obrigado a aceitar a trama como verdadeira. Na imagem B, o receptor 2 encontrou um erro no penúltimo bit, enviando uma trama de erro, no entanto, o receptor 1 aceitou a trama como verdadeira. O transmissor, ao receber a trama de erro vai retransmitir a mensagem. O receptor 2 recebe a mesma trama duas vezes. Na imagem C acontece o mesmo que na imagem B, a menos que não exista retransmissão da imagem, afetando somente o receptor C. Esses erros são muitos dependentes do bit rate, do trafego da rede e do número de nodos presente na rede.

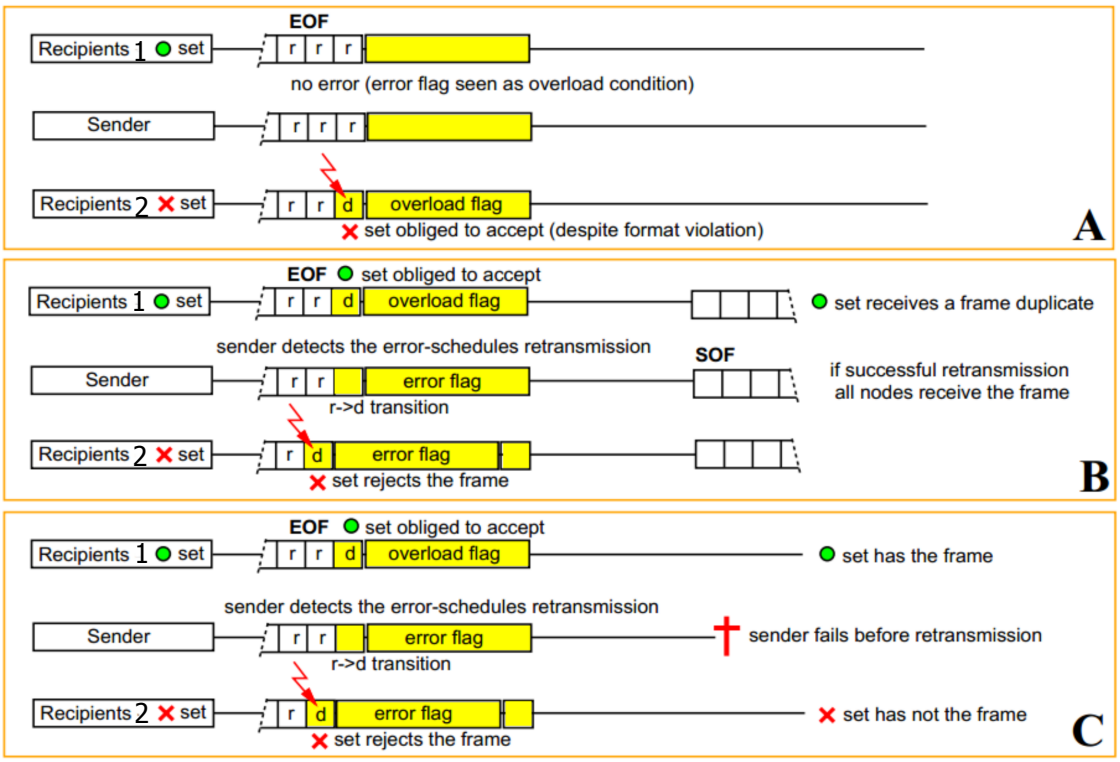

Figura 3.2: Inconsistência na recepção dos dados no CAN [11]

3.3.1.3 Suporte para barramentos redundantes

A rede CAN é baseada num barramento partilhado por todos os nodos. Essa situação leva a que qualquer dano no barramento, seja ele nos cabos, nos conectores ou nas interfaces elétricas,

faz com que o barramento não seja capaz de transmitir dados. O uso de barramentos redundantes acrescenta tolerância a falhas de vários barramentos, desde que exista sempre um barramento operacional para troca de dados.

3.3.1.4 Erros de *babbling idiot*

Os erros *babbling idiot* acontecem quando os transmissores, por erros de *hardware* ou *software*, enviam mensagens de forma descontrolada e repetida, impedindo os outros nodos de transmitir. Os erros de *babbling idiot* são dos erros mais indesejáveis, pois impedem o requisito de *fail silent* das aplicações *safety critical*. Para prevenir esses erros, diversos autores propuseram implementações de diversos *bus guardian* baseados em *hardware* com complexidades diferentes. No entanto, é possível também implementar *bus guardian* em *software*, inibindo a retransmissão em caso de erro unicamente alterando os bits dos registos CAN em certos microcontroladores.

Conclusão

As limitações acima apresentadas são as principais da rede CAN, podendo ser atenuadas com a proposta de comunicação tolerante a falhas. As limitações mais importantes a serem atenuadas são as de *babbling idiot*, de forma a garantir a característica *fail silent* da rede. Outras propriedades já referidas acima e importantes em relação ao CAN são o suporte de barramentos redundantes e o suporte para comunicações *time-triggered* .

No ponto seguinte será explicado com mais pormenor a solução retirada de [12] apontando os princípios da solução.

3.3.2 Solução do *Time Triggered* Redundante CAN

A solução apresentada por [12] consiste na implementação de um protocolo TDMA Duplo com base no CAN. A informação é enviada de forma cíclica para dois barramentos CAN com ligeiro desfasamento temporal. A informação está ligeiramente desfasada para prevenir erros de comunicação que possam afetar em simultâneo os dois barramentos.

Esta solução pressupõe que sejam usados microcontroladores com mais de um módulo CAN. Outro aspecto necessário para que seja possível utilizar o método proposto em [12] reside na possibilidade de conseguir transmissões em *single shot*. No *single shot*, o controlador CAN não retransmite a mensagem em caso de erro. Com isso, é possível garantir que o sistema será *fail-silent*, isolando assim a possibilidade de erros de *babling idiot* que possam vir a propagar-se na rede.

Outro aspecto a ter em conta neste sistema é a existência de um ciclo TDMA(figura 3.3). Dentro da janela temporal predefinida (*TDMA Cycle*) de tamanho fixo, são alocados *slots* temporais para o envio das mensagens, sendo o escalonamento das mensagens realizado *offline* ou *online* utilizando o método descrito em [30]. O ciclo TDMA deve incluir um mecanismo de sincronização do relógio como o descrito em [31] garantindo a precisão do relógio global do sistema.

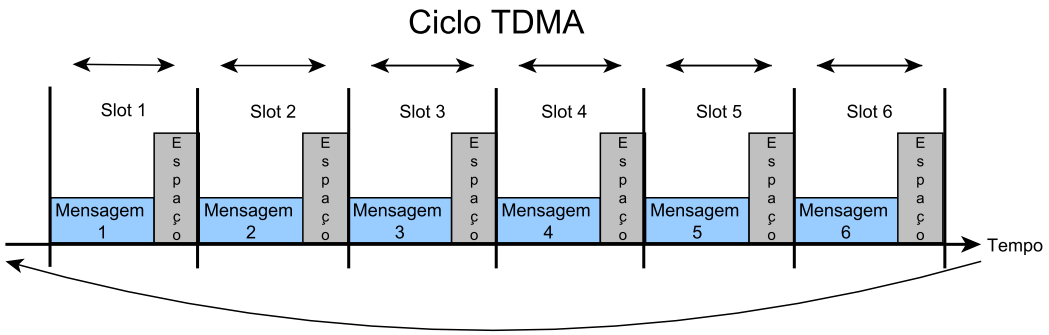

Figura 3.3: Principio do ciclo TDMA (adaptado de [12])

Funcionamento geral do sistema Dentro do *slot* de cada mensagem, a implementação proposta apoia-se numa transmissão em dois barramentos com um ligeiro desfazamento entre transmissões (figura 3.4). O mecanismo de transmissão inicia-se com o envio da mensagem em *single shot* por parte do controlador para o primeiro barramento, seguindo-se de um tempo de espera fixo (D) para o envio da mesma mensagem para o segundo barramento.

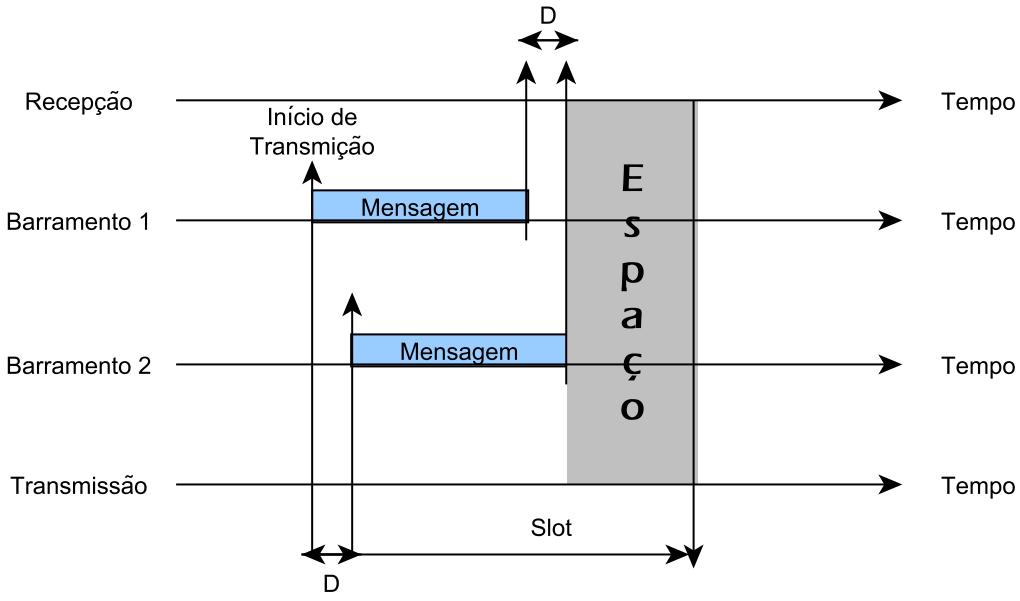

Figura 3.4: Esquema geral do mecanismo de Transmissão da solução (adaptado de [12])

No mecanismo de recepção, o receptor espera pela interrupção de um dos dois barramentos, sendo recebidas as duas mensagens provenientes dos dois barramentos com um espaçamento temporal de dimensão D. No final da recepção, é colocado um tempo de espera entre mensagens para garantir isolamento entre *slots*, sendo assinalados possíveis erros de recepção ocorridos.

Mecanismo de Transmissão Dentro do *slot* definido no ciclo TDMA, o autor propõe uma sequência de instruções esquematizada na figura 3.5. Para um número fixo de módulos CAN (superior a um), devemos garantir que a sequência seja atómica, ou seja sem interrupções externas que possam induzir atrasos e perturbar a transmissão. De forma a garantir o isolamento, ao nível do microprocessador, desabilita-se as interrupções do microcontrolador. Dentro da secção atómica é colocado um *timer* supervisor a contar o tempo decorrido dentro da secção de transmissão.

O procedimento de transmissão entra num ciclo cobrindo os dois módulos CAN. Para cada módulo, depois de preparar a mensagem para ser enviada é ligado o bit de transmissão, esperando por um de dois acontecimentos: a transmissão da mensagem é efetuada com sucesso ou o *timer* associado a supervisão ultrapassa $T(T \geq 2 \times \text{tempo de bit})$. Caso o *timer* ultrapasse o valor de T e não for enviada a mensagem com sucesso, é assinalada uma *flag* de erro correspondente ao módulo I em questão. O sistema deve esperar um tempo $D(D \geq 5 \times \text{tempo de bit})$ entre cada transmissão para ser possível uma separação temporal dos canais. Quando o sistema termina as transmissões de todos os módulos, volta a habilitar as interrupções externas, terminando o procedimento de transmissão.

Mecanismo de Recepção O ciclo de recepção (figura 3.6) é iniciado com a interrupção da recepção de um dos dois módulos CAN. É necessário ter em atenção a ordem de prioridade das interrupções da recepção das mensagens. O módulo 1 é o módulo de maior prioridade, sendo a prioridade decrescente para os módulos seguintes. O ciclo de recepção deve ser incluído, tal como acontece no ciclo de envio dentro da zona crítica, desabilitando as interrupções externas. Ao receber uma mensagem do módulo 2 (K=2), o módulo 1 é marcado com um bit de erro. Caso a primeira interrupção provenha do primeiro módulo, nenhum bit de erro é assinalado pelo sistema. Aquando da interrupção do módulo e do desabilitar das interrupções, é posto um *timer* para supervisão temporal do processo de recepção.

Depois de recebida a mensagem do módulo k, a aplicação deve fazer o *clear* do bit da interrupção correspondente e esperar pela mensagem no módulo seguinte. Estando as interrupções desabilitadas, é necessário vigiar o estado dos módulos de recepção, fazendo o teste da variável correspondente. Se depois do tempo de espera de $D(D \geq \times\text{tempo de bit})$ da recepção da mensagem anterior não existir nenhuma alteração dos bits de controlo da recepção o módulo corrente é marcado com um bit de erro. Esse processo é repetido para todos os módulos desde do módulo k até ao módulo 2. No final do ciclo de recepção, o sistema espera pelo fim do tempo de recepção para sair da rotina e voltar a habilitar as interrupções.

3.3.3 *Safety-Layer*

A solução apresentada acima acrescenta uma elevada tolerância a falhas nas comunicações, não sendo um sistema de troca de mensagens *safety critical*. Para atender aos requisitos de segurança das mensagens, a proposta engloba uma camada suplementar para a validação das mensa-

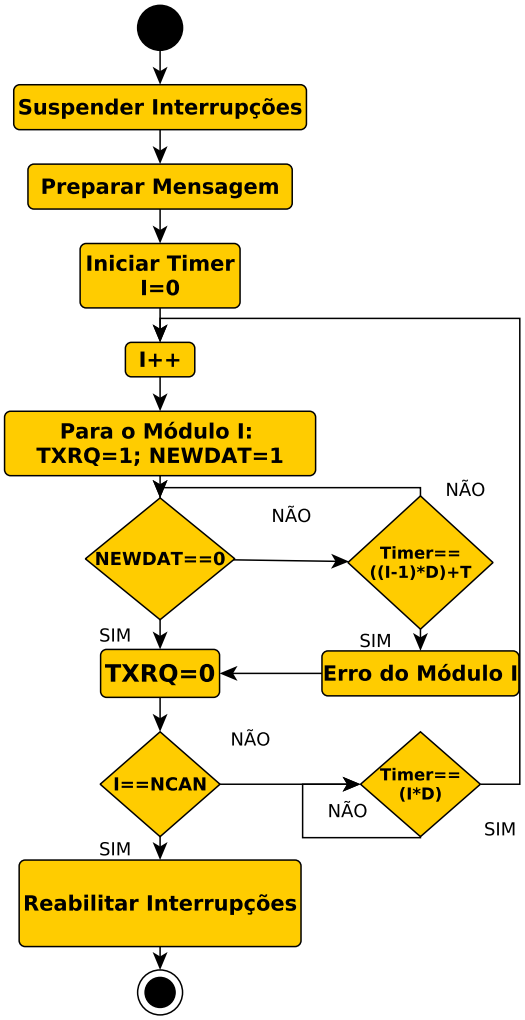

Figura 3.5: Esquema da Transmissão proposto (adaptado de [12])

gens trocadas. Esse mecanismo permitirá a troca de mensagens com um elevado nível de segurança.

A segurança nas mensagens é importante no controlo de aplicações no veículo, pois devido à ocorrência de uma anomalia nas comunicações que altere a confiabilidade das mensagens, o sistema deve detetar essas anomalias e impor medidas para atenuar os efeitos prejudiciais que possam vir a ocorrer.

Serão abaixo apresentados os mecanismo que contribuem para o aumento da confiabilidade das mensagens, introduzindo o conceito de ameaça, e mecanismos de aumento da segurança às ameaças.

Ameaças à segurança das mensagens

As ameaças podem ser entendidas como formas de alteração da confiabilidade da mensagem recebidas pela estação. As ameaças a serem consideradas em comunicações *safety-critical* são de

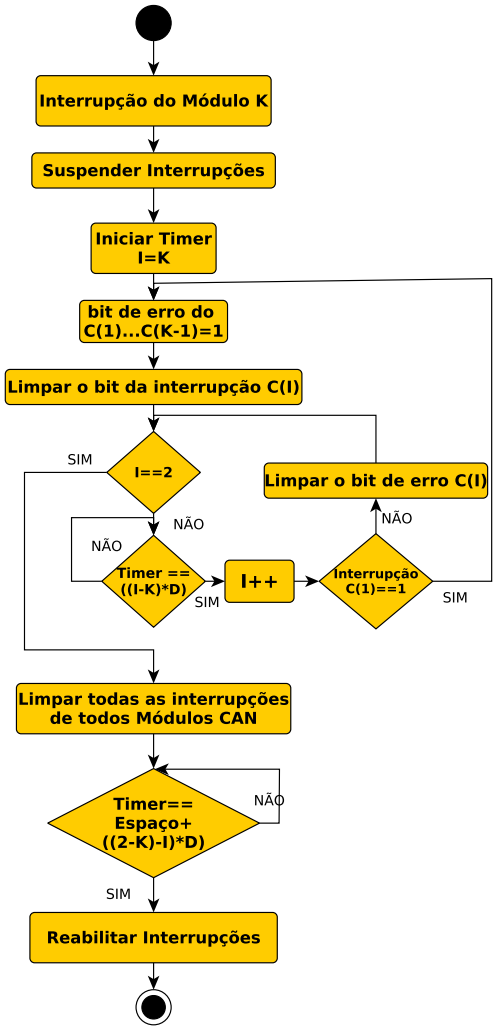

Figura 3.6: Esquema da Recepção proposto (adaptado de [12])

várias naturezas e podem induzir a aplicação em erro de diversas formas. A norma EN 50129 [32] é uma norma europeia que trata os aspetos da *safety* nas comunicações utilizadas em aplicações ferroviárias. Na parte 2 [16], são identificadas algumas ameaças que podem vir a suceder em comunicações entre nodos, sendo elas enumeradas em seguida:

Repetição de mensagens: Mensagem enviada uma vez e recebida mais do que uma vez;

Omissão de mensagens: A mensagem enviada pelo emissor é apagada da rede;

Inserção de mensagens: Uma mensagem fora do sistema ou não, que não corresponde a nenhuma informação válida;

Mensagem fora de sequência: A ordem de recepção das mensagens difere da ordem de envio;

Mensagem Corrompida: A mensagem de envio não corresponde à mensagem entregue;

Atraso na recepção: A mensagem é recebida fora do tempo;

Mensagem proveniente de fora do sistema: Mensagem exterior ao sistema, que tenta aceder ao mesmo.

A tabela 3.1 apresenta a relação entre eventos de risco e ameaças para a troca de dados seguras, sendo também enumeradas, tanto quais podem ser os eventos como as ameaças. No entanto, os eventos de risco podem resumir-se a descuidos das várias partes intervenientes, acidentes ou a intenções criminosas.

Para prevenir essas ameaças, é necessário acrescentar ao mecanismo de mensagens não seguras, uma camada de gestão do envio de mensagens seguras.

Tabela 3.1: Relação entre os possíveis riscos e as ameaças para a rede [16]

	Ameaças						
Riscos	**Repetição**	**Omissão**	**Inserção**	**Fora de Sequência**	**Corrupção**	**Atraso**	**Fora do Sistema**
Erro Sistemático	X	X	X	X	X	X	X
Corte nos cabos		X			X	X	X
Cablagem errada		X	X		X	X	X
Erro Aleatório	X	X	X	X	X	X	X
Hardware usado	X	X	X	X	X	X	X
Manutenção incorreta	X	X	X	X	X	X	X
Ruido eletromagnético		X			X		
Erro Humano	X	X	X	X	X	X	X
Problemas Térmicos		X			X	X	
Trovoada		X			X	X	
Fogo		X			X	X	
Hardware danificado		X			X	X	
Congestionamento da rede		X				X	
Hardware avariado		X			X	X	
Modificações não autorizadas do software	X	X	X	X	X	X	

A norma EN 50159 [32] disponibiliza mecanismos de deteção da ocorrência de ameaças para melhor conter os seus efeitos na aplicação. Essa implementa uma biblioteca de métodos para o desenvolvimento de mecanismos de deteção e contenção. A biblioteca *safety layer* contém 8 mecanismos de defesas sendo eles:

Número de sequência: O número de sequência consiste num valor numérico, enviado em conjunto com a mensagem e incrementado de cada vez que o emissor envia uma nova mensagem. Com este método, do lado do receptor é possível verificar se esse recebe a sequência correta de mensagens.

Referência temporal da mensagem: Através da referência temporal da mensagem, ou *timestamp* em inglês, o receptor é capaz de saber o tempo em que foi enviada a mensagem por parte do emissor. O *timestamp* consiste num valor numérico inserido dentro da trama com o valor temporal de acordo como relógio global. Esse método é muito útil para aplicações de controlo contínuo, onde a localização temporal das mensagens é importante.

***Timeout*:** O *timeout* é uma medida preventiva para controlar o envio correto de mensagens. O *timeout* é o tempo máximo que o receptor espera entre duas mensagens consecutivas do emissor. Pode ser entendido também como o tempo de resposta do receptor à mensagem recebida.

Identificação da fonte e do destinatário: A identificação da fonte e do destinatário nas mensagens, através de números de ID, é uma forma de conhecer a proveniência da informação tal como o(s) destinatário(s) da mensagem.

Resposta: A resposta consiste numa mensagem do receptor a confirmar a recepção da mensagem. Essa mensagem pode conter variada informação para confirmar o receptor de que a mensagem foi recebida. A resposta pode no entanto servir para informar o emissor que a mensagem não foi enviada corretamente e por consequente ativar os mecanismos necessários para corrigir o erro.

Identificação de comando: A identificação de comando consiste num parâmetro a incluir na mensagem, que identifica a razão pela qual o emissor envia a mensagem. Os comandos de identificação podem situar-se no pedido de informação a um nodo, a mensagens de sincronização de relógios, as mensagens de sinal de *heartbeat* dos nodos da rede, entre outros.

Código de segurança: Os códigos de segurança têm como principal função detetar alterações na mensagem enviada. Eles são calculados pelo emissor a partir dos bits constituintes da mensagem e são enviados juntamente com esses na trama. O receptor recebe a mensagem que contém o código de segurança, calcula do seu lado o valor de código de igual forma à feita pelo emissor e compara os dois valores. Esses códigos são relevantes para descobrir erros de corrupção de mensagens. Existem diversos códigos de segurança como o CRC, o MAC, o *hash code* e o *digital signature*.

Encriptação dos dados: Esse método é bastante usado em redes de comunicação públicas, e transmissões por ondas de radiofrequência, onde a exposição a ataques maliciosos é maior. A encriptação dos dados tem como princípio a alteração da representação dos dados com a ajuda de mecanismo de codificação da mensagem a enviar. O receptor com o mecanismo de descodificação reconstrói a mensagem enviada ao estado, antes de encriptar.

Todos os mecanismos referidos são individualmente, medida de deteção de várias ameaças, sendo no conjunto uma solução com um certo grau de redundância para as diversas ameaças que possam vir a suceder na rede. A tabela 3.2 contém um resumo das ameaças tratadas por cada medida da biblioteca.

Tabela 3.2: Relação entre as ameaças cobertas com as medidas propostas pela norma EN 50159

	Ameaças						
Mecanismos de defesa	**Repetição**	**Omissão**	**Inserção**	**Fora de sequencia**	**Corrupção**	**Atraso**	**Fora do sistema**
Número de Sequência	X	X	X	X			
Referência Temporal	X			X		X	
Timeout						X	
ID fonte/destinatário			X				
Resposta			X				X
Identificação do comando			X				X
Código de segurança					X		
Encriptação					X		X

Os mecanismos de defesa das ameaças, ao serem implementados na rede retiram desempenho na troca de dados. Estes mecanismos de defesa devem ser escolhidos de forma cuidada para se obter um equilíbrio entre confiabilidade e desempenho da rede.

A proposta de comunicação tolerante a falhas com base no CAN proposto na secção anterior tem mecanismos para detetar atrasos na recepção, devido ao mecanismo de acesso ao meio TDMA, em conjunto com o mecanismo indireto de confirmação das mensagens trocadas, fornecido pelo CAN.

Mediante as 5 primeiras ameaças presentes na tabela 3.2, escolheu-se implementar os mecanismos de defesa: número de sequência e código de segurança.

O número de sequência garante defesa das ameaças de repetição, omissão, inserção e de troca de sequência da mensagens da rede.

O código de segurança garante, em conjunto com o código CRC do CAN, a deteção da corrupção dos dados ao nível dos bits.

3.3.4 Formato da mensagem

Os mecanismos de troca de dados do protocolo CAN são complementados com o proposto nos mecanismos do número de sequência e do código CRC da *safety layer*. A trama CAN contém espaço reservado para 8 bytes de dados por isso deve ser bem escolhido o espaço a alocar para cada um dos campos de forma a obter uma boa confiabilidade dos mecanismos, sem perda de muita capacidade de transmissão de dados.

A divisão da trama apresentada na figura 3.7 mostra o formato da trama implementada no sistema de comunicação. Esta aloca dois bytes para o campo do número de sequência e um byte para o resultado do algoritmo CRC. O utilizador pode escolher habilitar individualmente cada um dos dois mecanismos de segurança.

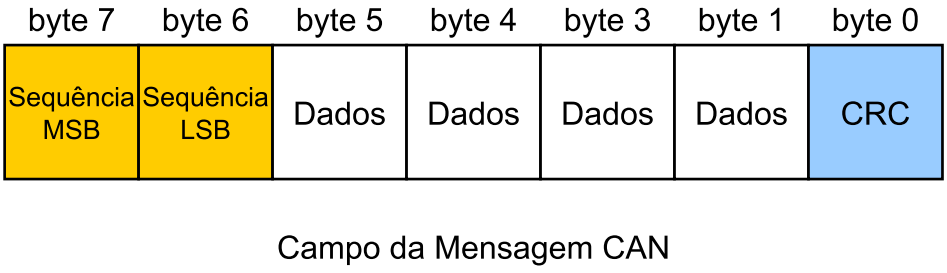

Figura 3.7: Formato da trama do protocolo de segurança critica

Número de Sequência

O número de sequência será enviado no campo da mensagem e terá o tamanho de 2 bytes. Com o número de sequência, será possível detetar uma sequência de mensagens com sequências de [0,

65535]. A tarefa de supervisionamento das mensagens recebidas, com o número de sequência recebido, pode retirar a propriedade *trusted* das mensagens recebidas fora da sequência. Quando o valor associado ao número de sequência atingir o valor 65535, o próximo valor da sequência a enviar será o valor 1.

Código de segurança CRC

O CRC é utilizado para codificar os 7 bytes de dados da trama CAN, mesmo que estejam incluídos os dois bytes do número da sequência (figura 3.7). Para tal, é necessário escolher um bom algoritmo de codificação CRC, capaz de encontrar erros de corrupção em 56 bits de dados (16 bits de número de sequência + 40 bits de dados). O CRC implementado consiste num código de 8 bits gerado a partir de uma equação. O código escolhido para a implementação é o CRC-8, sendo esta a escolha mais adequada para o número de bits a implementar, como é possível verificar no gráfico abaixo. A probabilidade de um erro não ser detetado (P) situa-se nos 10^{-26}, com um *Hamming Distance* (HD) de 4 (figura 3.8) [13] [33].

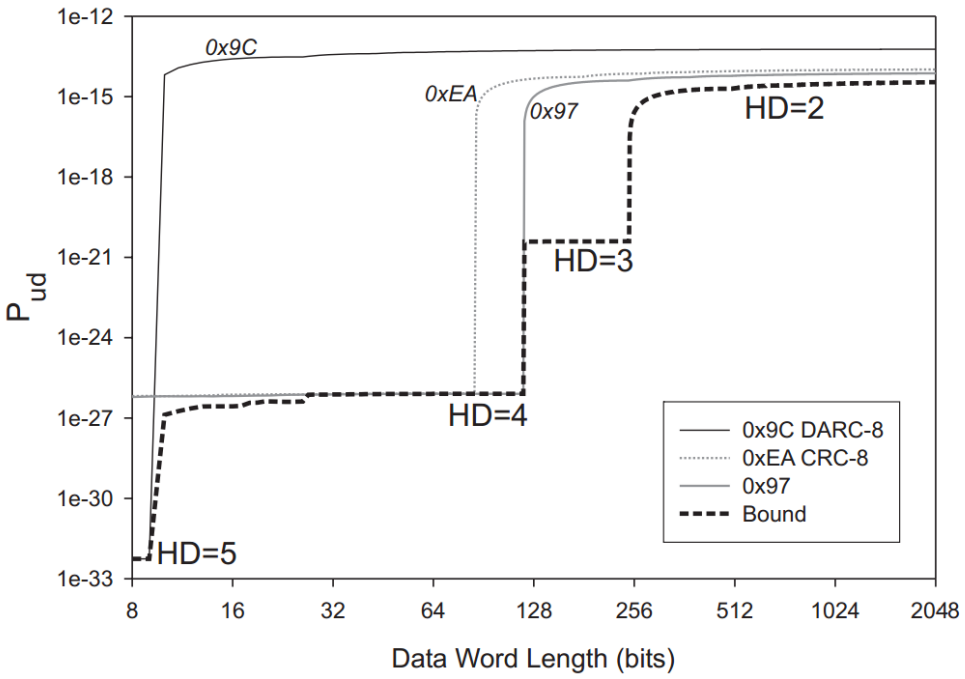

Figura 3.8: Gráfico comparativo de diversos algoritmos de CRC de 8 bits [13]

O CRC utilizado baseia-se num polinómio do oitavo grau $(0xEA = x^8 + x^7 + x^6 + x^4 + x^2 + 1)$, sendo que para diminuir a carga de operações a efetuar no microcontrolador, escolheu-se gerar uma *lookup-table* em *offline* e transferir essa tabela para o microcontrolador, sendo somente necessários 256 bytes de memória para armazenar a tabela [34].

3.4 Sistema Operativo Tempo Real

O uso de um RTOS em aplicações de controlo facilita o seu desenvolvimento, pois as ferramentas disponibilizadas facilitam o desenvolvimento das tarefas de controlo . O RTOS proposto teve o intuito de proporcionar uma fácil concepção de aplicações de controlo por parte dos alunos, adequada ao seus níveis de conhecimento. No entanto, procurou-se encontrar um sistema operativo baseado em implementações adequadas aos veículos.

Pelos itens requeridos para a solução, o uso de um RTOS responde a todas as solicitações de gestão de tarefas e de controlo de prioridades entre elas. Para cumprir os *standards* do AUTOSAR, pode ser escolhida uma de duas soluções, sendo elas o uso de uma implementação do OSEK ou uma implementação do RTOS AUTOSAR OS.

3.4.1 Implementações de Sistemas Operativos Tempo Real

Existem diversas implementações de sistemas operativos tipo real, tanto do RTOS OSEK/VDX como do AUTOSAROS. Os dois sistemas adaptam-se na perfeição ao objetivo da plataforma, sendo necessário escolher qual das implementações escolher baseado com base nas especificações de cada uma. Nesta análise, somente foram tomadas em conta as implementações livres de custo, o que é importante para a vertente académica da solução.

ARC CORE - A empresa ARC CORE desenvolveu uma série de produtos em linha com a política do AUTOSAR, constituída nos seguintes pacotes para o desenvolvimento de aplicações:[35]

***Artic Core*:** Este pacote contém uma plataforma de desenvolvimento de aplicações do *standard* AUTOSAR: inclui também um RTOS e um BSW baseado em AUTOSAR 3.1, serviços para comunicações CAN, LIN (*Local Interconnect Network*) e drivers para diversos componentes. O software Artic Core é distribuído segunda a licença GPL.

***RTE Builder*:** Ferramenta para geração automática do RTE AUTOSAR, com a possibilidade de configurar com base na aplicação desejada, em conjunto com uma ferramenta de validação do RTE. (licença Comercial)

***SWC Builder*:** Ferramenta para a criação de interfaces para os periféricos, portas de I/O, gestão do tipo de dados e gestão dos objetos de dados. (licença Comercial)

Trampoline - *Trampoline* é um RTOS desenvolvido pelo "*real time Systems group*" do "*Institut de Recherche en Communications et Cybernétique de Nantes* "(Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou and Yvon Trinquet), baseado em OSEK/VDX. Não sendo um RTOS certificado OSEK, foi criado a partir da diretiva OSEK, sendo classificado como OSEK *ready*. Utiliza o *goil*, ferramenta que compila os ficheiros do tipo OIL juntamente

com as *source file* do projeto e gera os ficheiros necessários para a compilação do conjunto RTOS + aplicação para a sua compilação através dos compiladores padrão de cada plataforma. O trampoline está disponível numa licença GNU *Lesser General Public* License V2.1. [36]

PICOS 18 - O PICOS18 é desenvolvido pela *Pragmatec* e é um RTOS baseado OSEK/VDX desenvolvido para a família de 8 bits dos microcontroladores da marca Microship (PIC18). Este sistema operativo não é um sistema operativo certificado pelo consórcio OSEK, pois o sistema operativo não implementa o conceito do ficheiro OIL, sendo o transporte para o PICOS18 feito através do uso de ficheiros ".c" e ".h". Todos os restantes mecanismos correspondem às propostas pela norma OSEK. No entanto, sendo o PICOS18 um RTOS para microcontroladores de arquitetura de 8 bits, existe uma limitação no uso de certos objetos como os eventos (limitados a 8 objetos evento). [37]

Erika Enterprise & RT Druid - O *Erika Enterprise & RT Druid* consiste num conjunto de *plugins* para o ambiente de desenvolvimento *Eclipse Indigo*, sendo uma das mais completas implementações *open source* de um sistema operativo do tipo OSEK. Implementa a norma ISO 17356. No entanto, como as implementações vistas acima, *Erika Enterprise & RT Druid* , também não é uma implementação certificada da norma OSEK, sendo de forma análoga ao Trampoline uma norma OSEK ready. O RTOS implementa as classes de conformidade de classes BCC1, BCC2, ECC1 e ECC2 presentes na norma OSEK, importante para a compatibilidade com a norma OSEK. O Erika é um sistema operativo para uma grande maioria dos microcontroladores da marca Microchip (arquiteturas 16-bits, dsPIC e 32-bits), arquiteturas ARM e powerPC entre outras[38]

3.4.2 RTOS escolhido

A escolha do RTOS apoiou-se em parâmetros não comparáveis entre si, pois as soluções diferem em vários aspetos, sendo impossível a comparação direta entre eles. No entanto, sendo necessário a escolha do sistema operativo tempo real deve ter em conta o seu uso em aplicações automóveis, a compatibilidade com vários microcontroladores de diversas marcas e modelos. A solução adoptada deve respeitar o carácter académico da solução, sendo que esta deve ter uma interface amigável para o desenvolvedor das aplicações.

De acordo com os critérios acima referidos, o RTOS apoia-se numa implementação do OSEK, com o uso do *Erika Enterprise & RT Druid* . A solução escolhida é a implementação mais próxima da norma OESK/VDX, pelo que se adequa às aplicações futuras. A sua grande gama de microcontroladores compatíveis com a implementação foi também determinante para a escolha da plataforma. Por fim, é de realçar que o *Erika Enterprise & RT Druid* têm diversas parcerias com

vários fabricantes de renome da industria automóvel e motociclo, tais como a FAAM, a EnSilica, a Magneti Mareli, a Aprilia Racing e a Cobra.

Especificações do *Erika Enterprise & RT Druid*

Essas são as principais especificações do *Erika Enterprise & RT Druid* disponíveis no website: [38]

- Implementação de um *kernel* com suporte tempo real, baseado em prioridades, com otimização da RAM;
- Interface minimalista para aplicações multi-tarefas;
- Interface semelhante ao proposto pelo consórcio OSEK/VDX com a inclusão de ficheiros de configuração OSEK(o *kernel* não foi totalmente certificado pelo OSEK/VDX);
- API (*Application Programming Interface*) semelhante ao OSEK/VDX com: tarefas, eventos, alarmes, modos de aplicação, semáforos e tratamento de erros;
- Implementação da seguintes classes de conformidade de tarefas (FP, BCC1, BCC2, ECC1, ECC2, EDF, FRSH);
- Suporte para tarefas preemptivas e não preemptivas;
- Suporte para programação de tarefas de prioridade fixa;
- Suporte para escalonamento *Earliest Deadline First* (EDF);
- Suporte para partilha de pilha de dados e de pilha de dados tarefas de única execução;
- Suporte para partilha de recursos;
- Suporte para alarmes periódicos;
- Suporte para tratamento de erros;
- Suporte para *Hook Functions*.

3.5 Plataforma de *hardware*

Para o desenvolvimento da aplicação OSEK, é necessário uma plataforma que preencha os requisitos impostos pelas diversas soluções propostas nas secções anteriores. O microcontrolador a utilizar deve preencher os requisitos listados abaixo:

- Limitações impostas na solução TT redundante:
    - o Existir mais do que um módulo CAN;

  - Possibilidade de desabilitar as retransmissões;

- Limitações impostas pelo *Erika Enterprise & RT Druid* :

  - Família de microcontroladores especificada pela empresa: Altera Nios II; ARM7TDMI; Atmel AVR5; ARM Cortex MX; EnSilica eSi-RISC; Freescale PPC e200; Freescale S12; Infineon Tricore; Lattice Mico32; Microchip dsPIC; Microchip PIC32; TI MSP430; Renesas RX200;

De acordo com as limitações presentes e o facto do sistema a desenvolver servir a aplicações de média complexidade, a escolha do microcontrolador incidia somente em dois microcontroladores semelhantes, sendo eles o dsPIC30F6012A e o dsPIC30F6014A. Sendo a diferença em termos de custo dos dois microcontroladores muito baixa, foi escolhido o microcontrolador da Microchip, o dsPIC30F6014A(figura 3.9). É um microcontrolador com uma arquitetura de 16 bits com frequência máxima de 30MIPS (Millions Instructions Per Second) e com diversas entradas para periféricos. Abaixo encontra-se descritas as principais especificações do microcontrolador, presentes no seu *website*. [14]

Arquitetura - 16-bit;

Velocidade do CPU - 30 MIPS;

Memória do Programa - 144 KBytes;

Memória RAM 8.192 Bytes;

Pinos de Entrada/Saída Digitais - 68;

Protocolos de Comunicação - 2xUART, 2xSPI, 1xI2C, 2xCAN;

Timers - 5x16 bits, 2x32 bits;

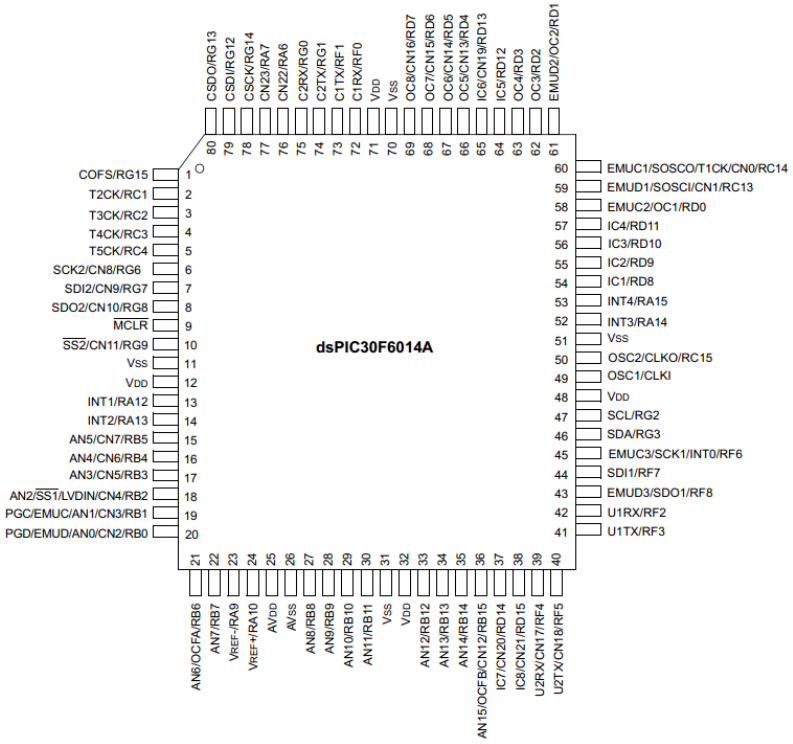

Figura 3.9: Esquema do microcontrolador proposto [14]

Capítulo 4

Implementação

Neste capítulo será apresentada a implementação da proposta descrita no capítulo anterior. Será em primeira instância descrita a configuração do RTOS, mostrando os modos de configuração dos principais objetos necessários.

Numa segunda parte será descrita a implementação da proposta do mecanismo de comunicação tolerante a falhas no ambiente RTOS.

4.1 A estrutura do RTOS

A aplicação implementada no *Erika Enterprise & RT Druid* consiste na compilação dos ficheiros de código da aplicação, do *Kernel* do RTOS e dos ficheiros de configuração dos objetos. Devido à complexidade das configurações do sistema operativo, o ambiente de desenvolvimento dispõe de uma interface para as configurações dos objetos através de um ficheiro OIL (*OSEK Implementation Language*). Nesse ficheiro são declarados os objetos com a possibilidade de configuração, sendo posteriormente transformado, pela ferramenta de geração, nos ficheiros de configuração necessários. Na figura 4.1, é possível ver em que zonas o utilizador tem liberdade de desenvolvimento. As zonas do "Código do utilizador" e do "Ficheiro OIL" são no seu conjunto as zona de desenvolvimento de aplicações.

Antes de apresentar a estrutura da aplicação a desenvolver, deve ser feita uma pequena introdução ao ficheiro OIL. Será feita uma descrição sumária dos principais aspetos, focando nas possíveis configurações existentes para cada objeto.

4.1.1 O Ficheiro OIL

Um ficheiro OIL constitui a interface onde é possível definir os diversos objetos constituintes do OSEK necessários à aplicação. Os objetos têm diversos parâmetros configuráveis, acrescentando flexibilidade no desenvolvimento das aplicações. Nesta secção serão detalhados os objetos OIL mais importantes, omitindo os objetos não presentes no *Erika Enterprise & RT Druid* ou que não são necessários [15].

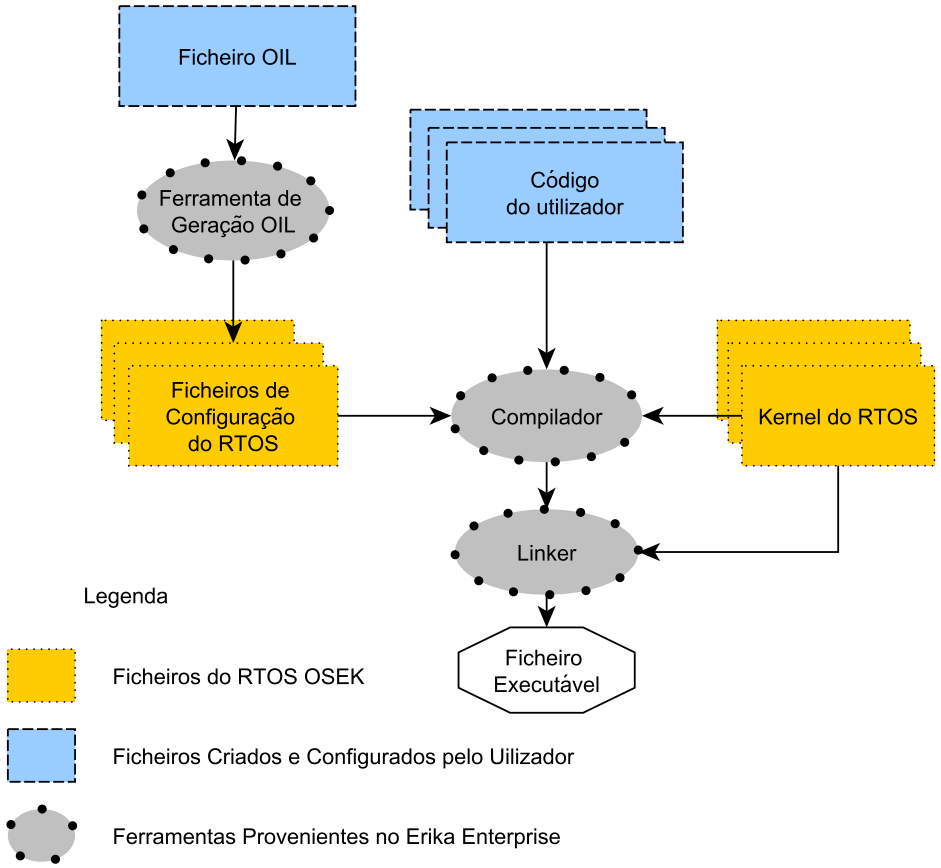

Figura 4.1: Estrutura geral dos elementos necessários para uma aplicação (adaptado de [15])

CPU

Todos os objetos estão incluídos dentro um macro-objeto chamado **CPU**. O **CPU**, como o nome indica, é o objeto que simboliza o processador do microcontrolador. Poderá existir mais do que uma instância do objeto **CPU**, caso a arquitetura do processador seja *multicore*. Dentro de cada **CPU**, existe a definição do objeto **OS**. Esse objeto define as propriedades do sistema operativo OSEK para uma determinada aplicação. O objeto **OS** tem diversos atributos associados, sendo os mais relevantes:

- **STATUS**: Define um de dois tipos de tarefas a implementar na aplicação. O atributo pode estar associado a tarefas básicas (**STATUS = STANDARD**) ou a tarefas *extend* (**STATUS = EXTENDED**);

- **APPMODE**: Pode-se entender a entidade **APPMODE** como um agrupamento de objetos que pode isolar sub-aplicações entre si. Dentro de um **CPU** deve estar definido pelo menos um **APPMODE**, sendo apenas possível um APPMODE estar ativo no decorrer da aplicação.

Tarefas

O objeto **TASK** corresponde à definição de uma tarefa no OSEK. Para cada tarefa existente na aplicação, é necessário definir um objeto do tipo **TASK**, associando uma prioridade à mesma. O atributo **PRIORITY** define a prioridade como sendo um valor inteiro sem sinal de 32 bits entre $[1; 2^{32} - 1]$. De forma análoga ao descrito na norma OSEK, o valor de menor prioridade "1"corresponde à tarefa de maior prioridade, sendo a prioridade decrescente em função do crescimento do valor do atributo **PRIORITY**.

O atributo **SCHEDULE** define a preemptividade das tarefas, ou seja, uma tarefa pode interromper a sua execução para dar o acesso ao recurso **CPU**. O atributo pode ter um de dois valores: **NON**, caso as tarefas não sejam preemptivas ou **FULL** caso sejam preemptivas.

O atributo **AUTOSTART** pode ativar as tarefas aquando do início de uma APPMODE. Se o atributo **AUTOSTART** é configurado com o valor **TRUE** é necessário indicar em qual ou quais **APPMODE** o início automático da tarefa ocorre. Caso não seja necessário o início automático da tarefa, basta colocar o atributo a **FALSE**.

Na configuração **EXTENDED**, é necessário indicar se exitem eventos associados à tarefa, enumerando-os.

Eventos

O objeto **EVENT** é representado por uma máscara de 64 bits. O objeto evento pode ser configurado automaticamente, atribuindo ao valor **MASK** o valor **AUTO**. Os eventos devem ser declarados em todas as tarefas onde serão usados.

Alarmes

O objeto **ALARM** pode ser usado para ativar um de três acontecimentos:

ACTIVATETASK - Ativar uma determinada tarefa;

SETEVENT - Ativar um evento associado a uma tarefa;

ALARMCALLBACK - Ativar uma pequena função do tipo ISR.

É possivel configurar no ficheiro OIL para o início automático de um alarme, aquando o início de um determinado **APPMODE**, através do atributo **AUTOSTART**, sendo possível configurar um alarme para vários **APPMODE**.

No objeto **ALARM** pode ser configurado o tempo do primeiro ciclo e o tempo dos restantes ciclos, com um valor correspondente ao número de impulsos do contador do microcontrolador.

No anexo A.1 podemos ver com mais detalhe a configuração de um ficheiro OIL, descrito acima.

4.2 Descrição da Aplicação

A implementação tem como objetivo a criação de um mecanismo que possibilite a troca de mensagens de três modos distintos, sendo eles:

Troca de mensagens dentro do Ciclo TDMA -Possibilita trocar mensagens dentro do ciclo TDMA;

Troca de mensagem seguras - Possibilita o envio de mensagens replicadas no dois barramentos, utilizando o mecanismo de transmissão e recepção descritos em [12];

Troca de mensagens não seguras - Envia mensagens pelo barramento 1 e 2 não utilizando mecanismos de tolerância a falhas.

O mecanismo de transmissão(figura 4.2) recebe a informação do tipo de transmissão juntamente com os dados a enviar. O gestor de transmissão prepara a mensagem a enviar e gere o seu envio. Caso exista falha no envio de mensagens, a aplicação envia ao utilizador um aviso de falha de um dos canais de transmissão.

Sempre que chega uma mensagem por um dos canais de recepção, o mecanismo de gestão recebe a informação proveniente desse canais, fazendo uma triagem do tipo de mensagem. Para cada tipo de mensagem recebida, o gestor de mensagens recebidas, verifica se a mensagem é valida e disponibiliza-a para o utilizador.

É necessário agora definir a estrutura da aplicação para a transmissão e supervisão da troca de mensagem.

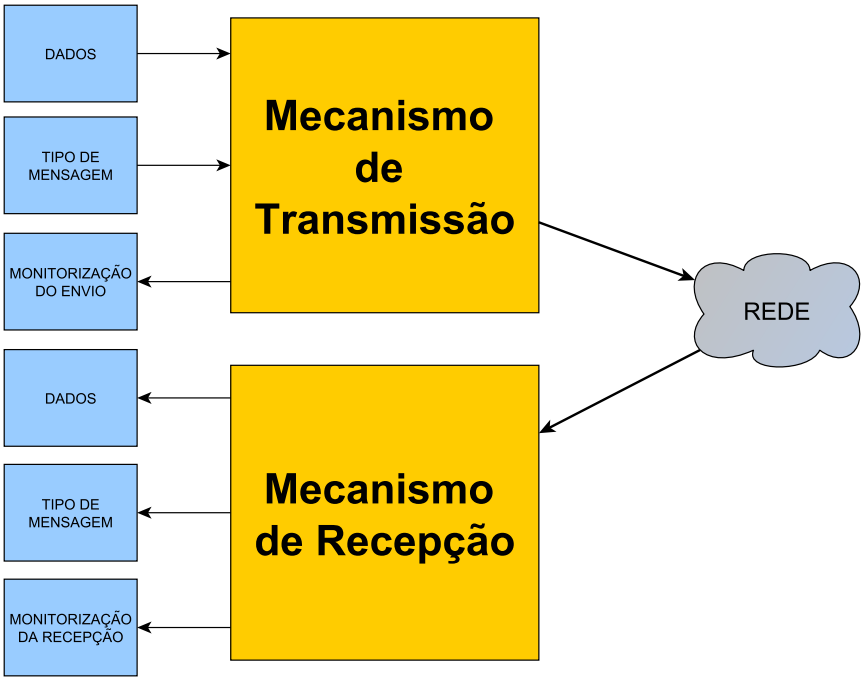

Figura 4.2: Mecanismo de envio e recepção de mensagens

4.3 Estrutura da Aplicação

A aplicação implementada pode ser descrita como um conjunto de sub aplicações. Como está descrito na figura 4.3, a aplicação central é composta por diversas subaplicações com o nome de SUBAPP, sendo possível com esses grupos genéricos, separar as componentes umas das outras, diminuindo assim a complexidade de compreensão. A divisão apresentada é uma divisão que procura ser típica das aplicações de controlo no veículo e está dividida em:

Aquisição de sinal - Aplicação de aquisição e tratamento básico dos sinais proveniente dos periféricos (Conversor A/D, portas digitais). Esta componente não será implementada, e servindo somente de exemplo para a apresentação da estrutura;

Comunicação - Aplicação do envio e recepção de mensagens seguras e não seguras;

Controlo - Aplicação de controlo para o desempenho das funcionalidades posteriormente implementadas pelos alunos do MIEEC;

Supervisão - Na aplicação de supervisão é possível detetar possíveis falhas nas diversas subaplicações descritas anteriormente.

Abaixo será descrita a subaplicação de comunicação, onde estão incluídas as implementações descritas no capítulo anterior.

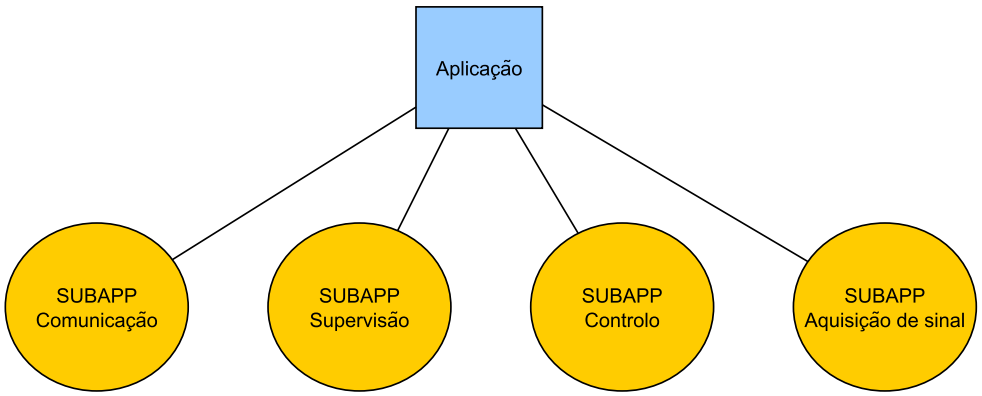

Figura 4.3: Disposição dos SUBAPP na implementação

4.3.1 Tarefas de Comunicação e Supervisão

Para desempenhar as funcionalidades pedidas, as SUBAPP incluem uma série de tarefas que implementem as funções propostas. Nesta subseção serão descritas as principais tarefas que coordenam o envio e recepção de dados nos três modos (tabela 4.1), em que SUBAPP se incluem e quais a suas funcionalidades.

Tabela 4.1: Tarefas utilizadas na aplicação de comunicação

Nome da Tarefa	SUBAPP	Descrição
Enviar_CAN1	Comunicação	Gestão do envio da mensagem pelo módulo 1;
Enviar_CAN2	Comunicação	Gestão do envio da mensagem pelo módulo 2;
Recepção_CAN	Comunicação	Gestão da recepção dos dois módulos;
S_Mensagem_S	Comunicação	Tarefa que coordena o envio de mensagens seguindo o proposto;
Triagem_CAN	Comunicação	Tarefa de separação do tipo de mensagem pelo Identificador;
Ciclo_TDMA	Comunicação	Gestão do ciclo *time-triggered* de envio de mensagens;
S_Mensagem_R	Comunicação	Tarefa de avaliação de mensagens seguras;
Ciclo_TDMA_R	Comunicação	Gestão das mensagens correspondentes ao ciclo TDMA;
Supervisão_CAN_S	Supervisão	Tarefa de supervisão do envio das mensagens pelos dois módulos;

4.3.2 Hierarquia de Tarefas da Aplicação

Sendo o aceso ao processador feito com base em prioridades, é necessário definir as prioridades de cada uma das tarefas constituinte da aplicação. Foi escolhido o modo de total preemptividade das tarefas dentro da aplicação, pois a total preemptividade acrescenta flexibilidade às aplicações. Com a preemptividade, as tarefas de baixa prioridade ficam com tempos de execução mais baixos, colocando tarefas de maior prioridade em execução.

Sendo as tarefas totalmente preemptivas, é necessário atribuir valores de prioridade para que as tarefas de gestão que necessitem de chamar tarefas de envio/recepção pelo meio da função *ActivateTask()* não deixem as tarefas de envio/recepção demasiado tempo no estado READY à espera do acesso ao processador.

Foi adotada uma configuração (figura 4.4), onde as tarefas de "baixo nível" de envio/recepção de mensagens CAN têm atribuídas as prioridades maiores. As tarefas de interface entre a gestão e as tarefas de envio-recepção são colocadas com prioridade logo abaixo da prioridade das tarefas de "baixo nível". Por fim, as funções de gestão de protocolos de codificação/descodificação de mensagens ficam com os valores de prioridade imediatamente abaixo dos valores das tarefas de interface. Todas as outras tarefas de controlo e aquisição de dados devem ter valores de *PRIORITY* inferiores aos das tarefas de comunicação, de forma a não perturbar os tempos de funcionamento da transmissão e recepção de mensagens. Os níveis de prioridade superiores ficam livres para que seja possível incluir tarefas de prioridades mais elevadas.

Depois da exposição das tarefas de gestão da troca de dados, será explicada a forma como essas tarefas comunicam entre elas para desempenhar as funcionalidades pretendidas.

4.3.3 Subaplicação de comunicação

Transmissão

A SUBAPP de comunicação disponibiliza uma interface para os diversos serviços de envio de mensagens existentes. O sistema pode enviar tanto mensagens seguras, como mensagens não seguras, podendo também ativar o ciclo TDMA. Todos esses serviços são implementados nas tarefas descritas na figura 4.5.

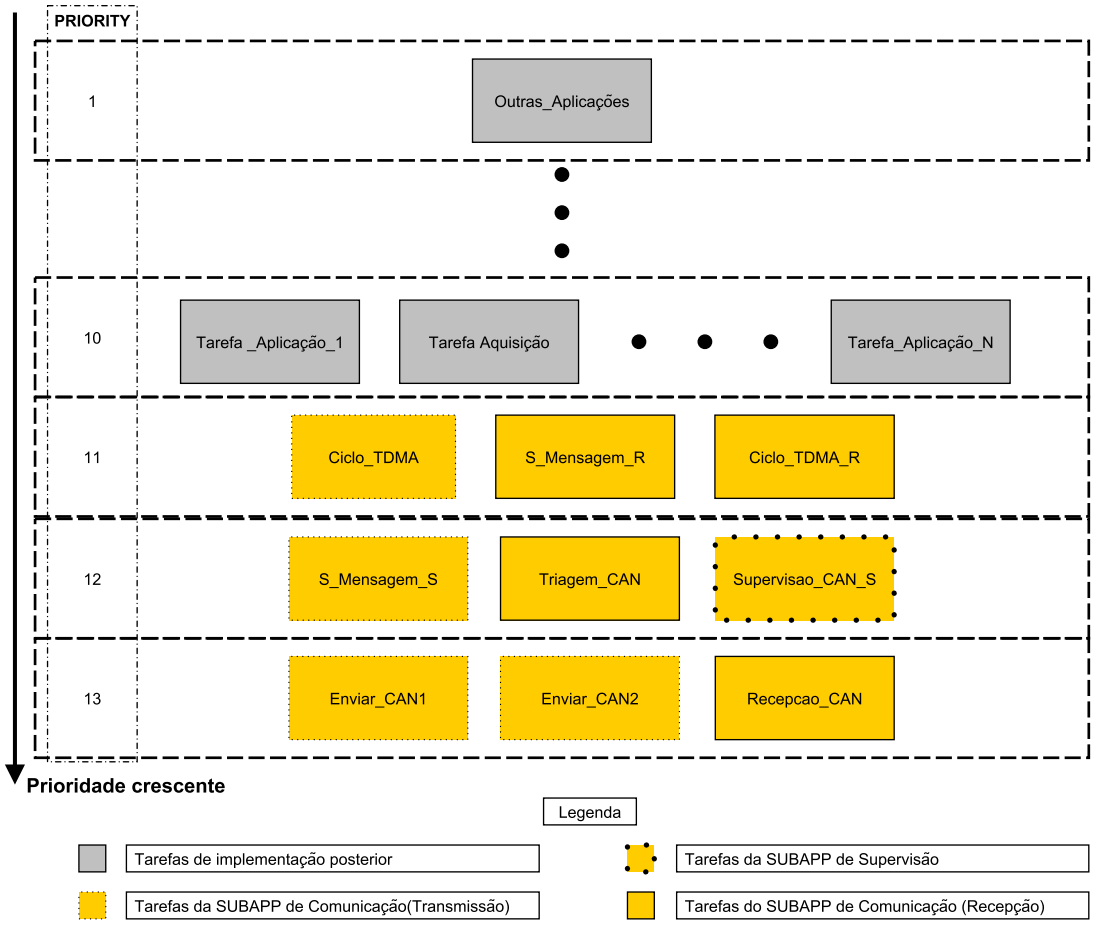

Figura 4.4: Hierarquia de Tarefas da Aplicação

A SUBAPP é composta pela interface que permite ao utilizador chamar individualmente cada serviço através de uma chamada de função da API do *Erika Enterprise & RT Druid* . A Chamada *ActivateTask()* possibilita a ativação das tarefas passando do estado *suspended* para o estado *ready*, estando prontas para executar, mal tenham acesso ao CPU. Em paralelo à ativação do serviço de envio da mensagem deve ser colocada a mensagem que se pretende enviar usando a função *ColocarMensagem()*, que será enviada por cada um dos serviços.

Mensagens não seguras - No modo das mensagens não seguras, as mensagens são enviadas utilizando os módulos CAN;

Mensagens seguras - Este modo possibilita o envio de mensagens seguras por parte do utilizador, seguindo o esquema proposto por [12] (figura 3.5) e recorrendo às tarefas de envio de mensagens não seguras;

Ciclo TDMA - O ciclo TDMA é definido pelo utilizador, sendo ele quem deve escalonar as tarefas conforme as suas necessidades. Dentro do ciclo TDMA, todas as mensagens enviadas

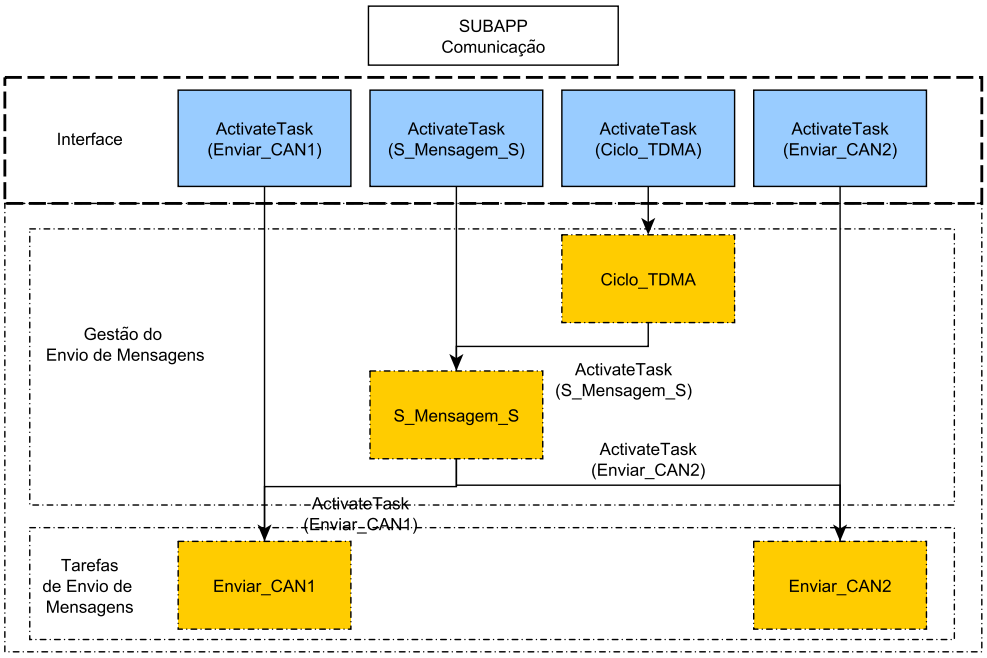

Figura 4.5: Disposição da componente de transmissão da SUBAPP de Comunicação

são mensagens seguras. Nesta implementação não foi considerada a implementação do mecanismo de sincronização do relógio.

A confirmação do envio das mensagens não é feita pela SUBAPP de comunicação, mas pela SUBAPP de supervisão, sendo explicado o funcionamento da tarefa na subseção do supervisionamento.

Recepção

O mecanismo de recepção(figura 4.6), como o mecanismo de transmissão, contém uma interface para o utilizador. A interface fornece a informação do momento de recepção de novas mensagens (TDMA, mensagens seguras, ou mensagens simples) ativando um evento para que as tarefas de tratamento de mensagens recolham e tratem as mensagens recebidas. Existe uma estrutura de dados que, ao receber uma nova mensagem de um certo identificador, sobrepõe os dados, sendo perdida a mensagem anterior.

A tarefa de recepção espera pela recepção de uma mensagem de qualquer um dos barramentos. A tarefa de Recepção, quando recebe uma mensagem do módulo 1, espera pela recepção do segundo módulo para o caso de existir recepção da segunda mensagem. A tarefa de recepção envia a informação recebida, que vai ser recebida e tratada pelo supervisor CAN. O supervisor CAN está encarregado de fazer a primeira triagem, onde as mensagens são divididas pelos três modos de envio de mensagens: o modo TDMA, o modo seguro e modo não seguro. A divisão pelos modos é feita por meio do identificador, em função do caráter *safety critical* da mensagem. As mensagens

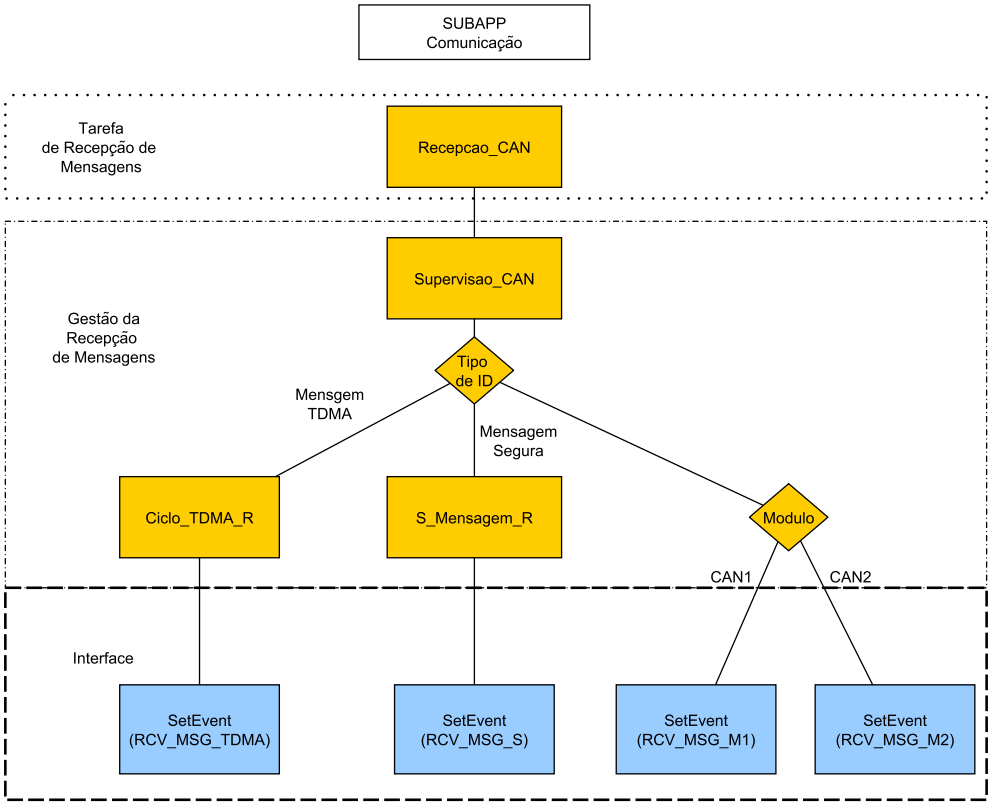

Figura 4.6: Disposição da componente de recepção da SUBAPP de Comunicação

de maior prioridade são as do ciclo TDMA, seguindo-se as mensagens seguras e acabando nas mensagens não seguras. Esta divisão está detalhada na tabela 4.2.

O mecanismo de recepção da tarefa de recepção, implementa um mecanismo semelhante ao da proposta de recepção das mensagens, mas com ligeiras alterações. Como é possível constatar na figura 4.7 o mecanismo de recepção não desabilita as interrupções do microcontrolador, deixando o sistema livre de receber qualquer tipo de interrupção. Essa simplificação acrescenta flexibilidade, sendo mais fácil receber as mensagens pelos diversos modos. Dentro de cada um dos módulos fazem os testes de confiabilidade da mensagem.

Mecanismos de segurança de mensagens

As tarefas de gestão das mensagens recebidas devem verificar se as mensagens recebidas são confiáveis. Essas tarefas implementam os mecanismos descritos na proposta de implementação da *safety layer*. Para avaliar se as mensagens são confiáveis, a proposta aponta três mecanismos de confirmação da sua segurança, sendo elas:

- No caso das mensagens serem enviadas pelos dois barramentos, é necessário verificar se as mensagens recebidas por cada módulo são idênticas(ID e mensagem);
- Não existem falhas na sequência das mensagens;

Tabela 4.2: Atribuição dos identificadores aos tipos de mensagem

ID	[0x001;0x2AA]	[0x2AB;0x554]	[0x555;0x7EF]
Modo de envio	Mens. TDMA	Mens. Seguras	Mens. Não Seguras

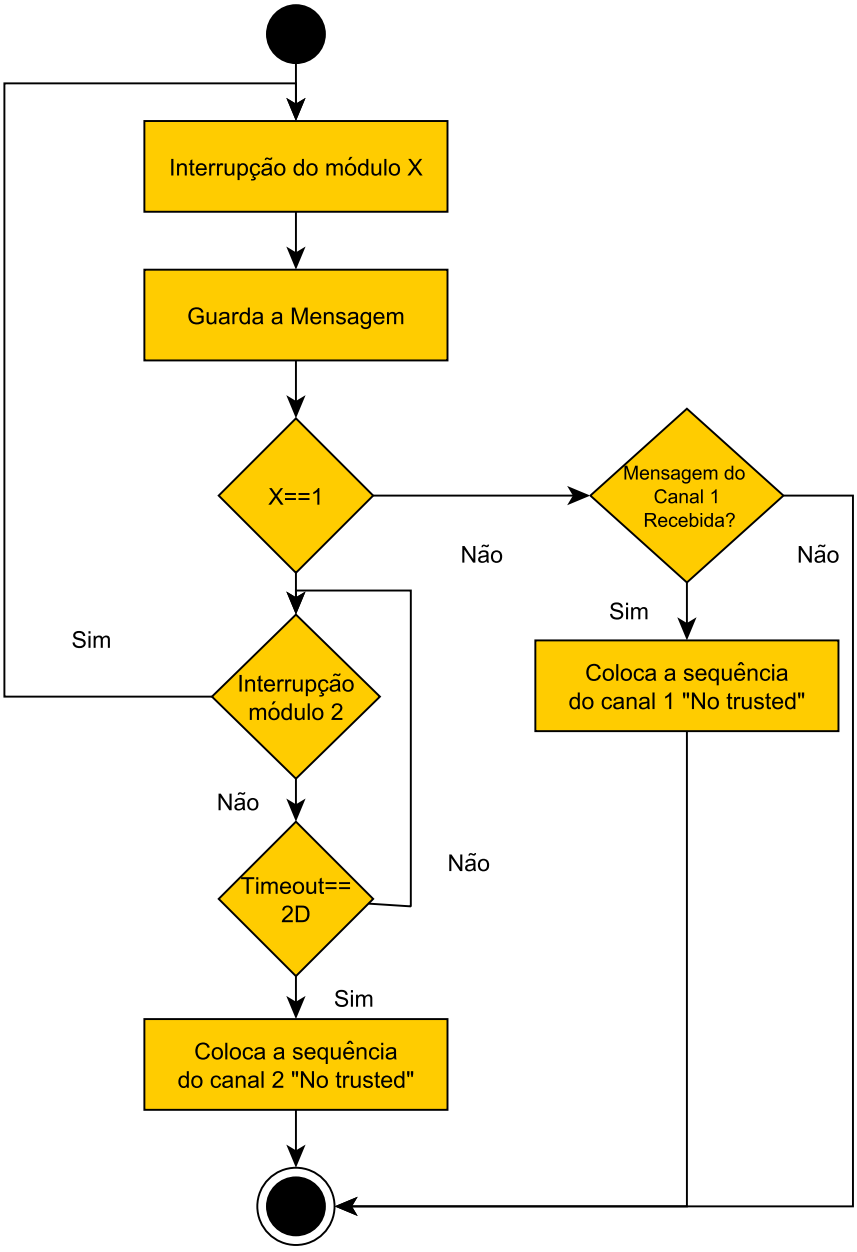

Figura 4.7: Tarefa de recepção das mensagens CAN

- O mecanismo de codificação e descodificação de dados não detetou corrupção dos dados.

Erro de Sequência

Um erro de sequência é detetado quando uma mensagem for recebida fora de sequência. O microcontrolador contém um registo dos valores de sequência para os identificadores das mensagens recebidas. Quando é recebida uma mensagem com número de sequência na tarefa de supervisão das mensagens recebidas, o valor recebido tem de corresponder à sequência de mensagens. No caso de existir falha de sequência, a mensagem é considerada *NO TRUTED SEQUENCE*, estando na mesma disponível para o utilizador, através da interface.

Erro de CRC

No mecanismo de avaliação de mensagens recebidas, o valor de CRC recebido é calculado por intermédio do mesmo mecanismo de cálculo usado pelo emissor. De forma análoga ao mecanismo de deteção de erros de CRC, caso os valores de CRC diferirem, a mensagem é avaliada como *NO TRUSTED CRC*, estando na mesma disponível para o utilizador.

4.3.4 Subaplicação de supervisão

A subaplicação de supervisão tem como principal funcionalidade encarregar-se de detetar anomalias nos diversos processos. Nessa SUBAPP, são colocadas todas as tarefas de supervisionamento dos diversos processos a controlar.

Na SUBAPP(figura 4.8) existe uma tarefa de supervisionamento do envio das mensagens. A tarefa de supervisão monitoriza as variáveis associadas aos dois módulos CAN e dos bits de controlo provenientes das diversas interrupções. No caso de qualquer erro de transmissão detetado num dos dois barramentos, é colocado um evento ativo avisando as aplicações de um erro no barramento.

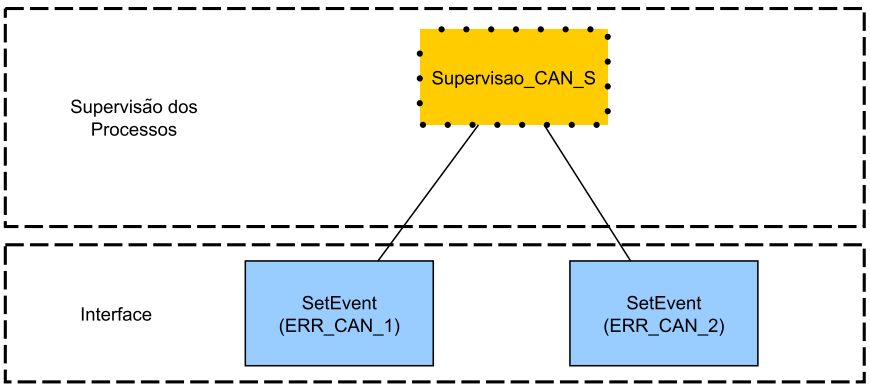

Figura 4.8: Disposição da componente da SUBAPP de Supervisão

4.4 Teste da Implementação

Com o objetivo de verificar o código implementado, foi necessário criar um circuito de testes, utilizando o microcontrolador escolhido no capítulo anterior (dsPIC30F6014A). A montagem do circuito corresponde ao esquema representado na figura 4.9. Na figura 4.10 é apresentada uma fotografia do circuito de desenvolvimento.

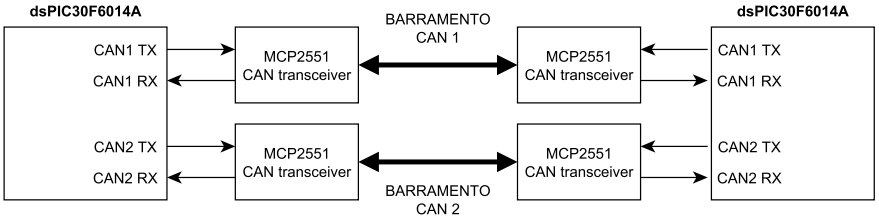

Figura 4.9: Esquema simplificado de ligação dos dois microcontroladores

Foram utilizados o *Pickit 2* e o ambiente de desenvolvimento *MPLAB X* da *Microchip* como ferramentas de verificação do funcionamento do microcontrolador. Estas ferramentas desempenham as funções de programador e de *debugger*, permitindo ao utilizador verificar o estado de execução do programa, bem como o estado das variáveis do microcontrolador. Esta verificação só pode ser feita aquando da interrupção do programa pelo utilizador.

Para a verificação da solução implementada, utilizam-se duas topologias de teste. Em primeiro lugar, os testes são efetuados em modo LOOPBACK. Neste modo, os módulos CAN do microcontrolador não estão ligados a nenhum barramento, sendo que as mensagens enviadas são recebidas pelo mesmo microcontrolador, através da simulação das sequências padrão do envio/-recepção, por parte do mesmo. A simulação do envio/recepção é possível através do habilitar do modo LOOPBACK no microcontrolador, ligando um bit de configuração dos módulos CAN.

Após a solução ser testada e verificada em modo LOOPBACK, fazem-se outros testes. Liga-se uma placa idêntica à anterior, recorrendo-se a um barramento CAN. A topologia deste teste encontra-se muito próxima da topologia de uso da plataforma, onde existe troca de dados entre os dois módulos.

Para verificar a implementação foram executados diversos testes, sendo somente apresentados na tabela 4.3 os resultados dos testes feitos com as duas placas:

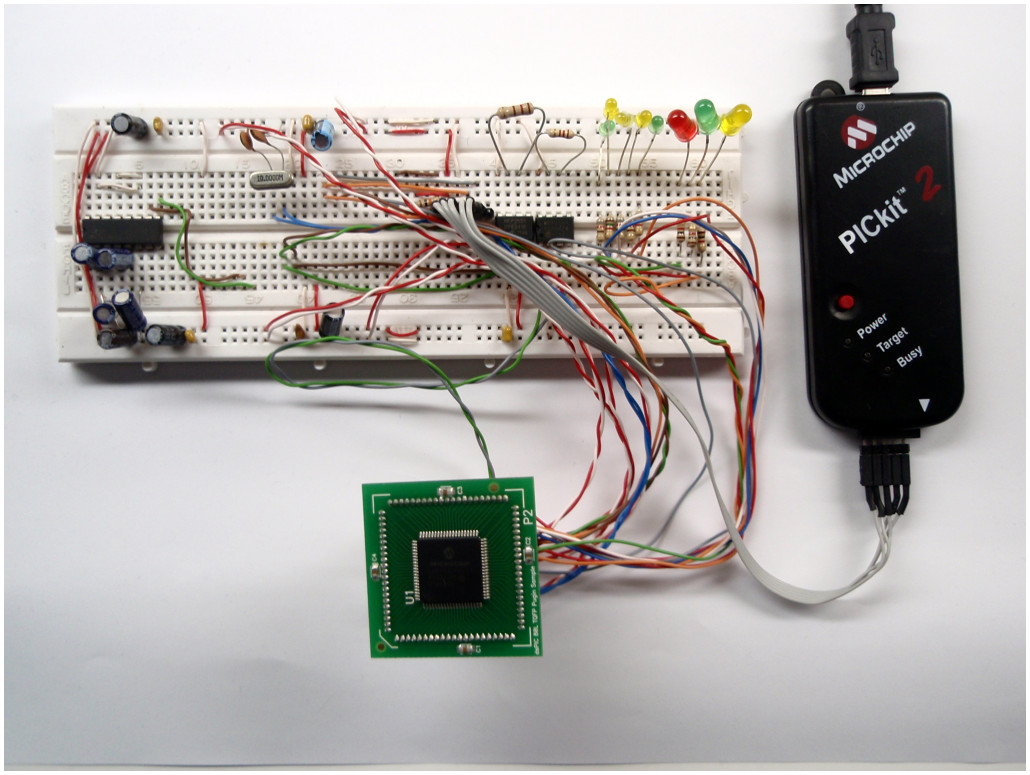

Figura 4.10: Fotografia do circuito de desenvolvimento

Tabela 4.3: Testes da implementação e seus resultados

Testes	**Resultados Esperados**	**Resultados Obtidos**
Corte da ligação de um dos barramentos (CAN 1 ou 2)	Deteção de falha no envio da mensagem;	A tarefa de supervisão envia um evento a assinalar o erro no barramento;
Corte da ligação dos dois barramentos (CAN 1 e 2)	Deteção de falha global no envio da mensagem;	A tarefa de supervisão envia um evento de falha global da comunicação;
Interrupção da ligação da rede (CAN 1 e/ou 2)	Deteção de falha na sequência de mensagens;	A tarefa de avaliação das mensagens indica que a mensagem é *no trusted* ao nível da sequência;
Corrupção dos dados em um dos barramentos	Assinala que as mensagens dos dois barramentos são diferentes; Assinala erro de CRC no barramento onde ocorreu a corrupção dos dados; Assinala a outra mensagem como segura.	A tarefa de avaliação das mensagens disponibiliza a informação de mensagem obtida, assinalando erro de mensagem *no trusted* do barramento com mensagem corrompida
Verificação da existência de erros *babbling idiot*, mesmo que os nodos estejam desligados	A propriedade de *single shot* deve ser mantida;	O controlador tenta enviar somente uma única vez cada mensagem;
Teste de *endurance* do sistema (Funcionamento continuo do sistema durante 6 horas)	Não deve ser detado nenhum erro de implementação	O microcontrolador não interrompiu a sua execução, apontando para um bom funcionamento do sistema;

Capítulo 5

Conclusões

Neste capítulo são apresentadas as principais conclusões desta dissertação bem como propostas para trabalho futuro com o intuito de melhorar o sistema.

5.1 Conclusões da Dissertação

Esta dissertação tinha como principal objectivo o desenvolvimento de um sistema de comunicação tolerante a falhas, onde fosse possível a troca de mensagens seguras, para futuras aplicações no VEC da FEUP.

Com vista a atingir os objectivos traçados foi desenvolvido um sistema de comunicação TDMA com uso de barramentos redundantes.

O mecanismo atómico de troca de dados acrescenta tolerância a falhas ao nível do barramento de dados. O seu perfil adequa-se a aplicações no automóvel.

O mecanismo de acesso ao meio TDMA tem como principal objectivo controlar o fluxo de dados no barramento, garantindo que somente uma estação transmitisse para o barramento.

Para troca de mensagens seguras na rede, os mecanismos do número de sequência e de CRC possibilitam a detecção da alteração dos dados trocados. O número de sequência possibilita a detecção de qualquer falha na sequência de dados recebidos por um determinado nó da rede. O mecanismo de CRC complementa o mecanismo de CRC do CAN, podendo assim detetar erros ao nível do bit nos dados da mensagem. Deve ser referido que os dois mecanismos referidos, utilizados na pratica, necessitam de 3 dos 8 bytes de dados da mensagem CAN, sendo dois para o número de sequência e um para o mecanismo de CRC.

O RTOS *Erika Enterprise & RT Druid* é outro componente importante para os requisitos relacionados com o futuro uso desta pelos alunos. Através do uso do RTOS, o desenvolvimento de aplicações encontra-se simplificado e o sistema de comunicação torna-se completamente transparente em relação às tarefas de controlo.

Na componente da implementação das soluções propostas, pode-se apontar os seguintes aspectos como motivos de realce:

O microcontrolador escolhido na implementação da proposta foi o microcontrolador da marca Microchip, modelo dsPIC30F6014A, compatível com o desenvolvimento no *Erika Enterprise & RT Druid* , que facilitou o desenvolvimento do mecanismo de troca de mensagens. Foram feitos testes em laboratório para verificar o pleno funcionamento da plataforma.

Os diversos testes para verificar o seu pleno funcionamento, tentaram ao máximo simular o seu modo de utilização na prática. No entanto, uma das componentes que não foi devidamente aprofundada foi a sua verificação no veículo com uma verdadeira aplicação de controlo. Essa componente será brevemente testada com as implementações dos alunos do MIEEC.

O uso do *Erika Enterprise & RT Druid* acrescenta flexibilidade, tanto para a solução de comunicação, como para o desenvolvimento de futuras aplicações, pois, esta componente introduz vários aspetos inerentes à gestão de tarefas, com a introdução dos conceitos de preemptividade e prioridades de tarefas.

Para concluir, é de referir que o trabalho desenvolvido introduziu os conceitos de comunicação tolerante a falhas, de segurança na troca de mensagens e do uso de RTOS em microcontroladores.

5.2 Trabalho futuro

Atendendo a que o trabalho desenvolvido servirá de plataforma base para futuros trabalhos a desenvolver por alunos da FEUP, nomeadamente à nível do desenvolvimento de controlo para o veiculo, enumera-se seguidamente algumas sugestões de trabalhos futuros que possam ser explorados:

- Bibliotecas de mecanismos de integridade dos dados do controlador;
- Desenvolvimento de tarefas de coordenação de outros periféricos (UART, SPI, I2C);
- Bibliotecas com as principais funções de controlo (controlador PWM, controlador *Fuzzy*);
- Estudo da facilidade de exportação do código para outro controladores (nomeadamente em arquiteturas ARM);

Anexo A

Código

A.1 Exemplo ficheiro tipo OIL

```
CPU mySystem {  //Objeto CPU

    APPMODE defaul;  //Definição de APPMODE
OS myOs {  //Objeto OS
EE_OPT = "DEBUG";
CFLAGS = "-Wall";
CPU_DATA = PIC30 {
APP_SRC = "code.c";
APP_SRC = "drv_can.c";
MULTI_STACK = TRUE {
IRQ_STACK = FALSE;
};
ICD2=TRUE;
};

MCU_DATA = PIC30 {
MODEL = PIC30F6014A;
};

STATUS = EXTENDED;
STARTUPHOOK = FALSE;
ERRORHOOK = FALSE;
SHUTDOWNHOOK = FALSE;
PRETASKHOOK = FALSE;
POSTTASKHOOK = FALSE;
USEGETSERVICEID = FALSE;
```

```
USEPARAMETERACCESS = FALSE;
USERESSCHEDULER = FALSE;
        KERNEL_TYPE = ECC1;

};

TASK CAN_s {  // Objeto Tarefa
PRIORITY = 8;
ACTIVATION = 1;
AUTOSTART = TRUE { APPMODE = defaul; };
STACK = PRIVATE {
SYS_SIZE = 128;
};
SCHEDULE = FULL;
EVENT = "CAN_NEW_MSG";
EVENT = "CAN_QUEUE_EMPTY";
EVENT = "CAN_S_EVENT";
APP_SRC = "code.c";
APP_SRC = "drv_can.c";

};

TASK CAN_r { // Objeto Tarefa
PRIORITY = 9;
ACTIVATION = 1;
AUTOSTART = TRUE { APPMODE = defaul; };
STACK = PRIVATE {
SYS_SIZE = 128;
};
SCHEDULE = FULL;
EVENT = "CAN_RCV_MSG";
EVENT = "CAN_QUEUE_FULL";
EVENT = "CAN_SAFETY_RCV_1";
APP_SRC = "code.c";
APP_SRC = "drv_can.c";

};

TASK Supervisao_CAN_S{ // Objeto Tarefa
PRIORITY = 8;
```

```
ACTIVATION = 1;
AUTOSTART = TRUE { APPMODE = defaul; };
STACK = PRIVATE {
SYS_SIZE = 128;
};
SCHEDULE = FULL;
EVENT = "CAN_SAFETY_SEND_OK";
EVENT = "CAN_SAFETY_SEND_ERR";
APP_SRC = "code.c";
APP_SRC = "drv_can.c";
};



EVENT  CAN_NEW_MSG{ // Objeto Evento
MASK = AUTO ;
};

EVENT  CAN_RCV_MSG{// Objeto Evento
MASK = AUTO ;
};

EVENT  CAN_ERR_MSG{// Objeto Evento
MASK = AUTO ;
};

EVENT  CAN_QUEUE_FULL{// Objeto Evento
MASK = AUTO ;
};

EVENT  CAN_QUEUE_EMPTY{// Objeto Evento
MASK = AUTO ;
};

EVENT  CAN_END_SEND{// Objeto Evento
MASK = AUTO ;
};

EVENT  CAN_S_EVENT{// Objeto Evento
MASK = AUTO ;
```

```
};


EVENT  SUPERV_R_EVENT{// Objeto Evento
MASK =AUTO;
    };

COUNTER myCounter{// Objeto Contador
    MINCYCLE = 1;
MAXALLOWEDVALUE = 511;
TICKSPERBASE = 1;
};

ALARM AlarmFlash {// Objeto Alarme
COUNTER = "myCounter";
ACTION  = ACTIVATETASK { TASK = "TaskFlash"; };
};

ALARM CAN_TASK_ALARM {// Objeto Alarme
        COUNTER = "myCounter";
        ACTION = SETEVENT { TASK = "CAN_s"; EVENT = "CAN_S_EVENT"; };
        AUTOSTART = TRUE { ALARMTIME = 100; CYCLETIME = 100; };
    };
};
```

Referências

[1] N. Kanekawa, "X-by-wire system," Hitachi Research Laboratory, Tech. Rep., 2004.

[2] R. Isermann, R. Schwarz, and S. Stolzl, "Fault-tolerant drive-by-wire systems," *Control Systems, IEEE*, vol. 22, no. 5, pp. 64–81, 2002.

[3] S. Purnendu, "Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives," *Reliability Engineering & System Safety*, vol. 96, no. 10, pp. 1349–1359, 2011.

[4] X. Weidong, P. C. Richardson, Z. Chenming, and S. Mohammad, "Automobile brake-by-wire control system design and analysis," *Vehicular Technology, IEEE Transactions on*, vol. 57, no. 1, pp. 138–145, 2008.

[5] G. Cena and A. Valenzano, "Controller area network," in *Industrial Information Technology*. CRC Press, 2005, pp. 13–1–13–21–, http://dx.doi.org/10.1201/9781420037821.ch13 data de acesso: 17/02/2012.

[6] C. in Automation, *CAN specification version 2.0 PART A*, CAN in Automation Std., http://www.can-cia.org/ data de acesso: 14/11/2011.

[7] G. Leen and D. Heffernan, "Ttcan: a new time-triggered controller area network," *Microprocessors and Microsystems*, vol. 26, no. 2, pp. 77 – 94, 2002, http://www.sciencedirect.com/science/article/pii/S014193310100148X data de acesso: 15/03/2012.

[8] L. Almeida, P. Pedreiras, and J. Fonseca, "The ftt-can protocol: why and how," *Industrial Electronics, IEEE Transactions on*, vol. 49, no. 6, pp. 1189 – 1201, dec 2002.

[9] *OSEK/VDX Operating System Version 2.2.3*, OSEK group Std., Rev. 1, February 2005, www.picos18.com/Docs/OS21R1.PDF data de acesso: 13/02/2012.

[10] AUTOSAR, "Layered sotware architecture," 2008, http://autosar.org/download/AUTOSAR_LayeredSoftwareArchitecture.pdf data de acesso 2/2/2012.

[11] L. Pinho and F. Vasques, "Improved fault tolerant broadcasts in can," in *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*, oct. 2001, pp. 305 –313 vol.1.

[12] M. Short and M. Pont, "Fault-tolerant time-triggered communication using can," *Industrial Informatics, IEEE Transactions on*, vol. 3, no. 2, pp. 131 –142, may 2007.

[13] P. Koopman and T. Chakravarty, "Cyclic redundancy code (crc) polynomial selection for embedded networks," in *Dependable Systems and Networks, 2004 International Conference on*, june-1 july 2004, pp. 145 – 154.

[14] Microchip. (2011, Fevereiro) dspic30f6014a. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en024766 data de acesso : 14/04/2012.

[15] *OSEK/VDX - System Generation OIL:OSEK Implementation Language Version 2.5*, Open Systems and the Corresponding Interfaces for Automotive Electronics Std., July 2004.

[16] E. C. for Electrotechnical Standardization, "En 50159-2 railway applications - communication, signalling and processing systems part 2: Safety-related communication in open transmission systems," 2001.

[17] F. Seidel, "Fault tolerant drive-by-wire systems," 2009.

[18] I. I. O. for Standardization, *ISO 6469-2:2009 Electrically propelled road vehicles – Safety specifications – Part 2: Vehicle operational safety means and protection against failures*, International Organization of Standarization Std., http://www.iso.org/ data de acesso: 12/3/2012.

[19] I. I. E. Commission, *Functional Safety and IEC 61508*, International Electrotechnical Commission Std., http://www.iec.ch/functionalsafety/, acedido 4/4/2012.

[20] F. Consortium, *FlexRay™ Protocol Specification Version 3.0.1*, FlexRay™ Consortium Std., disponivel em http://www.flexray.com/, acedido 4/03/2012.

[21] N. Storey, *Safety-critical computer systems*. Addison-Wesley, 1996.

[22] W. Dunn, *Practical design of safety-critical computer systems*. Reliability Press, 2002.

[23] U. Keskin, "In-vehicle communication networks: A literature survey," Julho 2009.

[24] C. in Automation, *CAN specification version 2.0 PART B*, CAN in Automation Std., http://www.can-cia.org/ data de acesso: 14/11/2011.

[25] A. S. R. Group, *Time-Triggered Controller Area Network Protocol*, Automotive Systems Research Group Std., http://www.can-cia.org/index.php?id=521 data de acesso: 6/03/2012.

[26] D. John, "Osek/vdx history and structure," in *OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), IEE Seminar*, nov 1998, pp. 2/1 –214.

[27] AUTOSAR, "Autosar technical overview," Automotive Open System Architecture, Tech. Rep., autosar.org/download/AUTOSAR_TechnicalOverview.pdf data de acesso: 23/02/2012.

[28] ——, *AUTOSAR, Specification of Operating System*, AUTOSAR GbR Std., Rev. 0003, June 2008.

[29] M. Bertoluzzo and G. Buja, "Application protocols for safety-critical can-networked systems," in *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*, sept. 2010, pp. T15–1 –T15–6.

[30] N. Kandasamy, J. P. Hayes, and B. T. Murray, "Dependable communication synthesis for distributed embedded systems," *Reliability Engineering & System Safety*, vol. 89, no. 1, pp. 81 – 92, 2005, http://www.sciencedirect.com/science/article/pii/S0951832004001784 data de acesso: 12/03/2012.

[31] M. Nahas, M. J. Pont, and M. Short, "Reducing message-length variations in resource-constrained embedded systems implemented using the controller area network (can) protocol," *Journal of Systems Architecture*, vol. 55, no. 5–6, pp. 344 – 354, 2009, http://www.sciencedirect.com/science/article/pii/S1383762109000368 data de acesso: 9/02/2012.

[32] E. C. for Electrotechnical Standardization, *EN50129. Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling*, Comite Europeen de Normalisation Electrotechnique Std., 2001.

[33] M. Rahmani, B. Muller-Rathgeber, and E. Steinbach, "Error detection capabilities of automotive network technologies and ethernet - a comparative study," in *Intelligent Vehicles Symposium, 2007 IEEE*, june 2007, pp. 674 –679.

[34] R. N. Williams, "A painless guide to crc error detection algorithms," *Internet publication August*, p. 34, 1993, http://www.epm.ornl.gov/~dunigan/crc.html data de acesso: 10/03/2012.

[35] ARCCORE, *ARR CORE AUTOSAR Software*, ARCCORE Std., http://www.arccore.com/products/ data de acesso: 6/03/2012.

[36] *Trampoline v1.1 OpenSource RTOS project*, Real-Time Systems group of IRCCyN Std., http://trampoline.rts-software.org/ data de acesso: 17/01/2012.

[37] Pragmatec, *PICOS18 RTOS*, Pragmatec Std., http://www.picos18.com/ data de acesso: 5/3/2012.

[38] *Erika Enterprise and RT-Druid 1.6.0 RTOS*, Erika Enterprise Std., http://erika.tuxfamily.org/ data de acesso: 17/01/2012.