INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Sistema de Monitorização e Comunicação de Dados para Aplicação em Pavement Energy Harvesting

LUIS PEDRO CADECO CUNHA LOBATO DE AZEVEDO
Outubro de 2021

POLITÉCNICO
DO PORTO

SISTEMA DE MONITORIZAÇÃO E COMUNICAÇÃO DE DADOS PARA APLICAÇÃO EM *PAVEMENT ENERGY HARVESTING*

Luís Pedro Cadeco Cunha Lobato de Azevedo

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

2021

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Luís Pedro Cadeco Cunha Lobato de Azevedo, Nº 1150635, 1150635@isep.ipp.pt

Orientação científica: Cecília Maria do Rio Fernandes Moreira Reis, cmr@isep.ipp.pt

Coorientação: Nuno Alexandre Neto Dias, nnd@isep.ipp.pt

Empresa: PAVNEXT – *Technological Pavements*, Lda

Supervisão: Francisco João Anastácio Duarte, fd.pavnext@gmail.com

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Automação e Sistemas

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

30 de outubro de 2021

Agradecimentos

Ao longo dos últimos 12 meses dedicados a este projeto, inúmeras foram as pessoas que, direta ou indiretamente, me ajudaram a concluir esta grande etapa da minha vida académica. Quero, desta forma, deixar os meus devidos agradecimentos.

A toda a minha família, pelo apoio, força e carinho que sempre me prestaram, com um especial agradecimento ao meu irmão, madrinha e padrinho.

Ao Engenheiro Francisco Duarte, da *Pavnext*, pela proposta, apoio e acompanhamento excecional que me prestou no decorrer deste projeto.

À minha orientadora, Engenheira Cecília Reis, pela disponibilidade e apoio na elaboração deste trabalho.

Ao meu coorientador, Engenheiro Nuno Dias, pela ajuda indispensável à realização deste projeto e pelo conhecimento transmitido na área.

A todos os colegas que tive oportunidade de conhecer ao longo desta jornada, e que, de alguma maneira, marcaram a minha vida. Um agradecimento especial aos meus colegas Rui Carvalho e José Silva pelo companheirismo, incentivo e ajuda prestada ao longo dos últimos meses.

Ao meu grupo de amigos, pelos momentos de lazer bem passados, que me ajudaram a abstrair dos problemas que foram surgindo.

À minha namorada, que embora não tenha estado presente desde o início desta aventura, foi um incentivo enorme nos momentos finais do projeto.

Finalmente, e o mais importante de todos os agradecimentos, à minha Mãe, que embora já não esteja presente, foi a pessoa que mais lutou por mim e pelos meus objetivos, e definiu quem sou. Por estes, e muitos outros motivos, dedico-lhe, não só este projeto, como os meus últimos 6 anos de carreira académica.

Resumo

Neste projeto foi desenvolvido um sistema de monitorização de comunicação de dados para aplicar nos módulos *NEXT-road*, da empresa *Pavnext*. Estes módulos são desenhados para serem instalados no pavimento e são capazes de gerar energia com a passagem de veículos. Além da quantificação da energia gerada, o sistema desenvolvido tem como objetivo monitorizar a temperatura e humidade internas, assim como quantificar o número de veículos que por ele passam, e a que velocidades.

O sistema é composto por três tipos de controladores, interligados através de dois barramentos, em formato de hierarquia. Os barramentos foram construídos utilizando como base o protocolo CANFD, uma variante mais recente do protocolo CAN, muito utilizado na indústria automóvel. Para o desenvolvimento do projeto, foram estudados vários tipos de microcontroladores e respetivos IDE, assim como diferentes protocolos de comunicação. Utilizaram-se microcontroladores da família G4, da empresa *STMicroelectronics*, e desenvolveram-se PCB para cada um dos controladores. Em termos de sensores, foram escolhidos um sensor de temperatura e humidade *HTS221*, um acelerómetro *ADXL345* e um sensor de monitorização de energia *PAC1934*, todos capazes de comunicar por I^2C. Devido à falta de componentes eletrónicos no mercado, numa fase mais avançada do projeto, foi necessário substituir o *PAC1934* por um sensor de corrente e utilizar o ADC do microcontrolador. Além do *hardware*, foi desenvolvido todo o *firmware* para o sistema através do *software STM32Cube*.

Durante o processo de implementação, foram realizados diversos testes: periféricos, protocolos, sensores e diferentes montagens. Numa fase mais avançada, o *hardware* desenvolvido foi implementado nos protótipos *NEXT-road* e foram realizados novos testes, simulando a passagem de veículos. Finalmente, o sistema foi montado em uma estrada no concelho de Matosinhos, permitindo realizar testes com veículos.

Palavras-Chave

Sistema de Monitorização, Barramento de dados, CANFD, I^2C, *Pavement Energy Harvesting*

Abstract

In this project, a data communication and monitoring system was developed to be applied in the NEXT-road modules, produced by Pavnext. These modules are designed to be installed on the pavement and can generate energy with the passage of vehicles. In addition to quantifying the energy generated, the developed system aims to monitor the internal temperature and humidity, as well as quantify the number of vehicles that pass through it, and at what speeds.

The system is composed of three types of controllers, connected through two buses, in a hierarchy format. The buses were built using CANFD protocol, a more recent variant of the CAN protocol, widely used in the automotive industry. For the development of the project, several types of microcontrollers and their IDE were studied, as well as different communication protocols. The chosen microcontrollers belong to the G4 family, produced by STMicroelectronics, were used with custom PCB for each controller. In terms of sensors, an HTS221 temperature and humidity sensor, an ADXL345 accelerometer and a PAC1934 energy monitor sensor were chosen, all capable of communicating via I^2C. Due to the lack of electronic components on the market, at a more advanced stage of the project, it was necessary to replace the PAC1934 with a current sensor and use the microcontroller ADC. In addition to the hardware, all firmware for the system was developed using the STM32Cube software.

During the implementation process, several tests were carried out: peripherals, protocols, sensors and different assemblies. At a more advanced stage, the developed hardware was implemented in the NEXT-road prototypes and new tests were carried out, simulating the passage of vehicles. Finally, the system was mounted on a road in Matosinhos, allowing tests to be carried out with vehicles.

Keywords

Monitoring System, Data Bus, CANFD, I^2C, Pavement Energy Harvesting

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ADC – *Analog-to-Digital Converter*

ACK – *Acknowledge*

ARM – *Advanced RISC Machines*

CAN – *Controlled Area Network*

CANFD – CAN *Flexible Data-rate*

CANH – CAN *High*

CANL – CAN *Low*

CIPO – *Controller Input Peripheral Output*

COPI – *Controller Output Peripheral Input*

CPU – *Central Processing Unit*

CS – *Chip Select*

CPHA – *Clock Phase*

CPOL – *Clock Polarity*

CRC – *Cyclic Redundancy Check*

DAC – *Digital-to-Analog Converter*

DFN – *Dual-Flat No-Lead*

DLC – *Data Length Code*

DMA – *Direct Memory Access*

EOF – *End of Frame*

FIFO – *First In First Out*

I^2C – *Inter-Integrated Circuit*

ICT – *Information and Communication Technologies*

ID – *Identification*

IDE – *Integrated Development Environment*

IEEE – *Institute of Electrical and Electronics Engineers*

LCD – *Liquid Crystal Display*

LED – *Light-emitting Diode*

LSB – *Least Significant Bit*

MOSFET – *Metal-Oxide-Semiconductor Field-effect Transistor*

MSB – *Most Significant Bit*

NACK – *Not Acknowledge*

PCB – *Printed Circuit Board*

PCIe – *Peripheral Component Interface – Express*

R/W – *Read and Write*

RTOS – *Real Time Operating System*

RTR – *Remote Transmission Request*

SAS – *Serial Attached Small Computer System Interface*

SATA – *Serial AT Interface*

SCL – *Serial Clock*

SCLK – *Serial Clock*

SDA – *Serial Data*

SE – Sistema Embebido

SOF – *Start of Frame*

SOIC – *Small Outline Integrated Circuit*

SPI – *Serial Peripheral Interface*

STM – *STMicroelectronics*

UART – *Universal Asynchronous Receiver Transmitter*

UQFN – *Ultra-thin Quad Flat-pack No-lead*

USART – *Universal Synchronous Asynchronous Receiver Transmitter*

USB – *Universal Serial Bus*

1. INTRODUÇÃO

1.1. ENQUADRAMENTO

Durante toda a história, grande parte da população mundial residiu em zonas rurais e pequenas comunidades. Nos últimos séculos, devido ao fenómeno da urbanização, verificou-se uma migração em massa das populações rurais para as grandes cidades. O ano de 2007 ficou marcado pela ultrapassagem da quantidade de população urbana *versus* a quantidade de população rural, a nível mundial [1].

Em 2018, um estudo conduzido pelas Nações Unidas, revelou que cerca de 55% da população mundial habita em área urbana. Além disso, estima-se que este valor atinja os 68% até 2050, sendo o crescimento mais notório em regiões como Ásia e África [2].

Com este aumento da população, as cidades começaram a tornar-se, aos poucos, mais vastas e densas, conduzindo a uma alteração no modo de vida dos seus habitantes. Não só aumentou a complexidade de providenciar diferentes recursos à população, tal como energia elétrica, água e serviços de comunicação, mas também a dificuldade de gerir problemas como congestionamento de trânsito, recolha de lixos e sistema de saneamento, nas diferentes artérias das cidades [3].

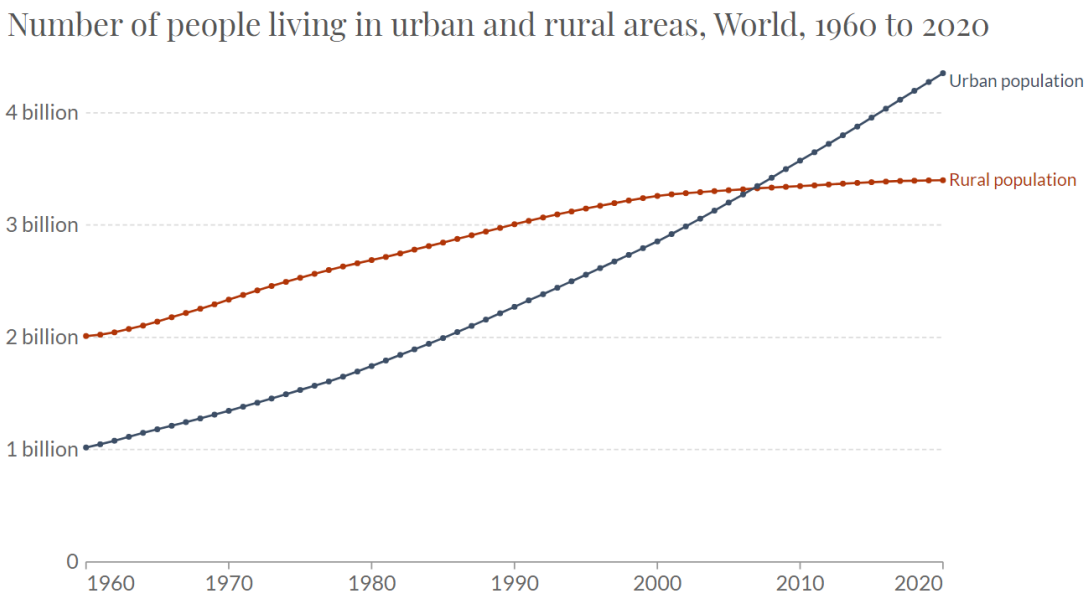


Figura 1 Número de habitantes em zonas urbanas e rurais [1]

De forma a combater tais dificuldades, começaram a ser desenvolvidos e instalados diversos sistemas autónomos, em diferentes pontos das cidades, com o intuito de recolher dados que pudessem ajudar a melhorar a qualidade de vida e segurança da população. Ao longo dos últimos anos, não só têm surgido novos equipamentos e sistemas direcionados para a aquisição de diferentes tipos de dados, mas também os anteriores têm sofrido alterações e aprimoramentos. Estes avanços a nível dos equipamentos são favorecidos pela evolução de *hardware* e de protocolos de comunicação, assim como pela facilidade de acesso aos mesmos. A aquisição e o tratamento destas grandes quantidades de dados é a base do conceito das *Smart Cities* [3], [4].

1.1.1. SMART CITIES

Não existe uma definição concreta para o conceito de *Smart Cities*, embora este possa ser definido como um conjunto de soluções inteligentes que permitem às cidades melhorar a vida dos seus habitantes, direta e indiretamente. O que torna estas cidades *smart* é a capacidade de criar uma rede de sistemas inteligentes, que conseguem tomar um conjunto de decisões através da análise dos dados obtidos do meio envolvente. Estes tipos de ações podem ser aplicados de modo a melhorar a eficiência, equidade, sustentabilidade e qualidade de vida nas cidades [4].

O uso de tecnologias de Informação e Comunicação (ICT) permite, às cidades, melhorar o uso dos seus recursos, de uma forma mais ecológica e menos poluente. Na prática, o resultado poderá conduzir à criação de redes de transporte inteligentes, locais públicos seguros, melhor fornecimento de serviços públicos e gestão otimizada da energia elétrica e iluminação na via pública. Para que tal seja possível, é necessário que exista um empenho acrescido por parte da administração destes centros urbanos [5].

A União Europeia tem impulsionado a transição para as *smart cities* através da criação de verbas e ajudas para todos os países constituintes. Não só criou mecanismos como o *Smart Cities Marketplace*, onde podem ser acompanhados projetos no âmbito do desenvolvimento das *smart cities*, como tenta motivar este desenvolvimento nas áreas metropolitanas europeias [5], [6].

De todos os setores presentes nas grandes cidades, o setor da mobilidade é um dos mais complexos e difíceis de lidar. Esta complexidade aumenta proporcionalmente com a área metropolitana, pois depende não só das tecnologias implementadas, como do comportamento dos seus habitantes. Por existir esta dificuldade, é uma área das *smart cities* que se tem ramificado e evoluído em várias componentes: mobilidade elétrica, transportes públicos, sinalização, iluminação das vias, controlo de tráfego, recursos para os peões, entre outros [7].

1.1.2. CONTROLO DE TRÁFEGO RODOVIÁRIO

O tráfego rodoviário é um dos maiores problemas da sociedade moderna, ainda sem uma solução à vista. Embora aconteça em diferentes locais, é mais comum nas grandes cidades e nos seus arredores. Todos os problemas de tráfego rodoviário, sejam eles acidentes, atrasos de viagem, poluição sonora e do ar, entre outros, devem ser analisados individualmente e deve ser procurada uma solução capaz de os resolver total ou parcialmente, para cada local e problema em específico. Atualmente, alguns dos problemas mais comuns nos grandes centros urbanos são a ineficiência de grande parte dos sinais de transito colocados nas vias, que muitas vezes não evitam acidentes, assim como os tempos fixos nos semáforos, que, em alguns casos, geram grandes filas de espera [8].

O planeamento das estradas de uma cidade ou região é um fator muito importante para que a qualidade do tráfego possa evoluir. O mais comum é dividir as vias de circulação por níveis de velocidade e níveis de acesso. Em primeiro lugar encontram-se as autoestradas

e vias rápidas que conduzem o tráfego de alta velocidade a grandes distâncias. De seguida inserem-se as principais avenidas, estradas nacionais e outras vias importantes, consideradas as artérias das cidades, que distribuem o tráfego a todas as zonas urbanas, a uma velocidade mais reduzida que o caso anterior. Por fim, as estradas locais que permitem o acesso a todos os restantes destinos, sendo estas de baixa velocidade. Os problemas de tráfego, como demoras no trânsito e acidentes, abordados anteriormente, são consequências diretas de um mau planeamento de estradas, resultante de um número insuficiente de estradas para cobrir todas as necessidades de uma certa região [9].

As questões de tráfego rodoviário não são uma novidade nem um problema associado apenas à sociedade dos dias de hoje. Este tipo de problemas atinge a humanidade há bastantes séculos, estendendo-se até ao Império Romano. Nesse tempo chegaram a ser tomadas medidas para banir o tráfego com rodas na cidade de Roma, em determinados horários, ou até banir completamente as carroças de entrar nas muralhas da cidade. Medidas como estas tiveram de ser tomadas pois, dentro das cidades, não existia organização nem planeamento a nível de estradas. Com o passar dos anos, e com o aumento do número de veículos motorizados em relação aos transportes a cavalo, foram sendo introduzidos novos elementos de segurança e de controlo do tráfego rodoviário. Alguns exemplos são a criação dos sinais de trânsito e a obrigatoriedade da carta de condução [9].

No que toca ao controlo de tráfego rodoviário, este pode ser dividido em diferentes categorias, dependendo do nível de tecnologia utilizada e da época. Em primeiro lugar surgiu o controlo de tráfego manual, normalmente associado aos policias sinaleiros, não existindo qualquer tipo de tecnologia associada, apenas trabalho humano. Este começou a ser substituído, mais tarde, pelo controlo de tráfego automático, como é o caso dos semáforos com tempos fixos, instalados em cruzamentos ou intersecções. Mais tarde, já com a introdução de novas tecnologias no mercado, começaram a ser criados sistemas inteligentes baseados em processamento de imagem. As imagens podem ser captadas de locais altos, processadas por algum tipo de computador e utilizadas para determinar os intervalos de tempo dos semáforos luminosos. Em alguns locais, são também utilizadas tecnologias de comunicação sem fios para comunicar entre veículos de emergência e recetores instalados em intersecções. Um exemplo prático deste tipo de controlo, é o estado de um semáforo ser alterado com o passar de uma ambulância [9].

Dentro da categoria do controlo de tráfego computorizado, como é o caso dos semáforos e dos sistemas com processamento de imagem, existem ainda outros tipos de dispositivos e sistemas que não afetam diretamente o trânsito, mas tomam um papel indireto, considerados, assim, sistemas passivos. Estes dispositivos são responsáveis por analisar e recolher dados sobre o tráfego no meio envolvente. Estão incluídos neste conjunto, tecnologias como radares, câmaras e sensores no pavimento, que têm evoluído significativamente ao longo dos últimos anos, estando cada vez mais aptos a detetar transgressões a nível de velocidade, sinais vermelhos e até detetar matrículas, por exemplo. Na presença dos dados recolhidos, as entidades reguladoras tomam certas medidas, com o objetivo de melhorar a circulação em determinadas vias e tornar os condutores mais cautelosos, aumentando também a segurança. As medidas tomadas resultam, normalmente, de violações de medidas de controlo e leis do tráfego rodoviário.

Das diversas transgressões às leis rodoviárias, o incumprimento dos limites de velocidade é o tipo de transgressão mais comum, a nível mundial, e que causa mais acidentes. Segundo dados recolhidos pelas autoridades britânicas, entre abril e junho de 2020, 63% dos veículos excederam o limite de velocidade em zonas limitadas a 30 *mph* (aprox. 48 km/h) e 53% excederam o limite de velocidade em autoestradas [10]. Na Noruega, outro estudo realizado a jovens até aos 25 anos, indica que 80% dos acidentes envolvendo a colisão de veículos, tem como principal causa o incumprimento dos limites de velocidade [11].

Os dispositivos referidos anteriormente são instalados e utilizados de diferentes formas. Primeiramente, os radares, que poderão ser fixos ou portáteis, são utilizados para detetar incumprimentos de limites de velocidade. Baseiam-se em câmaras muito rápidas que conseguem medir a velocidade de deslocação de um veículo. Em alguns casos, estes são ligados a painéis que exibem a velocidade a que o condutor se desloca, mas, na maioria dos casos, as fotografias das matrículas são utilizadas para aplicar uma multa por contraordenação. Além dos radares de velocidade, câmaras são também utilizadas, em locais como interseções, para detetar a passagem indevida de sinais vermelhos. Por fim, existem também os sensores instalados no pavimento, que podem ser utilizados para detetar a velocidade e o tipo dos veículos. É neste grupo de dispositivos que se insere o *NEXT-road*, que será o foco deste projeto.

Este dispositivo, implementado à superfície do pavimento, é capaz de aproveitar a energia cinética dos veículos, transformando-a em energia elétrica. Simultaneamente, a

velocidade dos veículos é reduzida e são obtidos dados do tráfego rodoviário, como número de veículos, peso e velocidade.

1.2. MOTIVAÇÃO

Este projeto surgiu de uma proposta colocada pela empresa *Pavnext*, que visa desenvolver um sistema de monitorização e comunicação de dados de tráfego para aplicar aos seus módulos de *Pavement Energy Harvesting*, os *NEXT-road*. O grande desafio neste projeto é perceber de que forma os respetivos dados podem ser obtidos do ambiente em redor, e de que forma todo o sistema de aquisição e transmissão pode ser implementado de maneira modular e escalável.

Os módulos *NEXT-road* serão instalados nas estradas, em conjuntos de 10 unidades e além de adquirirem dados exteriores, referentes aos veículos que por eles passam, também são capazes de gerar energia elétrica com a passagem desses veículos. Os dados recolhidos, depois de analisados e filtrados, serão enviados para um controlador local que os irá armazenar temporariamente até realizar uma transmissão dos mesmos para uma base de dados remota. Finalmente, os dados referentes aos veículos serão tratados e disponibilizados em uma plataforma *online*, que irá também facultar, de forma privada, dados de monitorização dos dispositivos.

De todo este projeto, a proposta apresentada foca-se no desenvolvimento do sistema de recolha de dados, através da leitura de sensores, e da transmissão dos mesmos, por fio, até ao controlador local, instalado à face da estrada. Todo o restante projeto, como é o caso da parte mecânica, geração de energia e gestão da base de dados, será estudado e projetado por outros colegas e profissionais da empresa.

Na Figura 2 está representada uma demonstração todo o projeto *NEXT-road*, no qual estará incluído este trabalho.

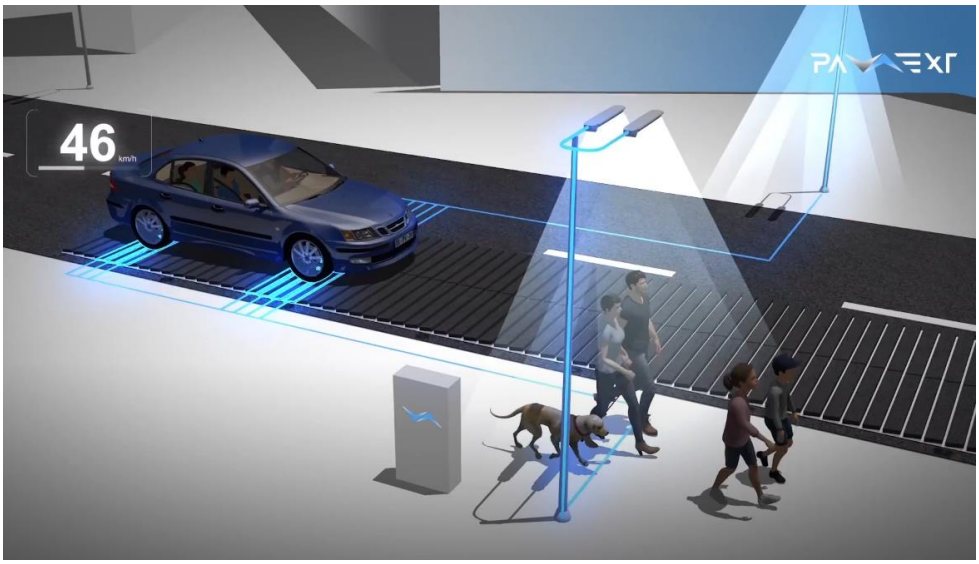


Figura 2 Demonstração do projeto *NEXT-road*

A Figura 3 ilustra uma abordagem inicial ao problema apresentado. Nela é possível observar os diferentes módulos instalados no solo, conectados a um controlador que irá fazer a comunicação com o exterior, posicionado fora da estrada.

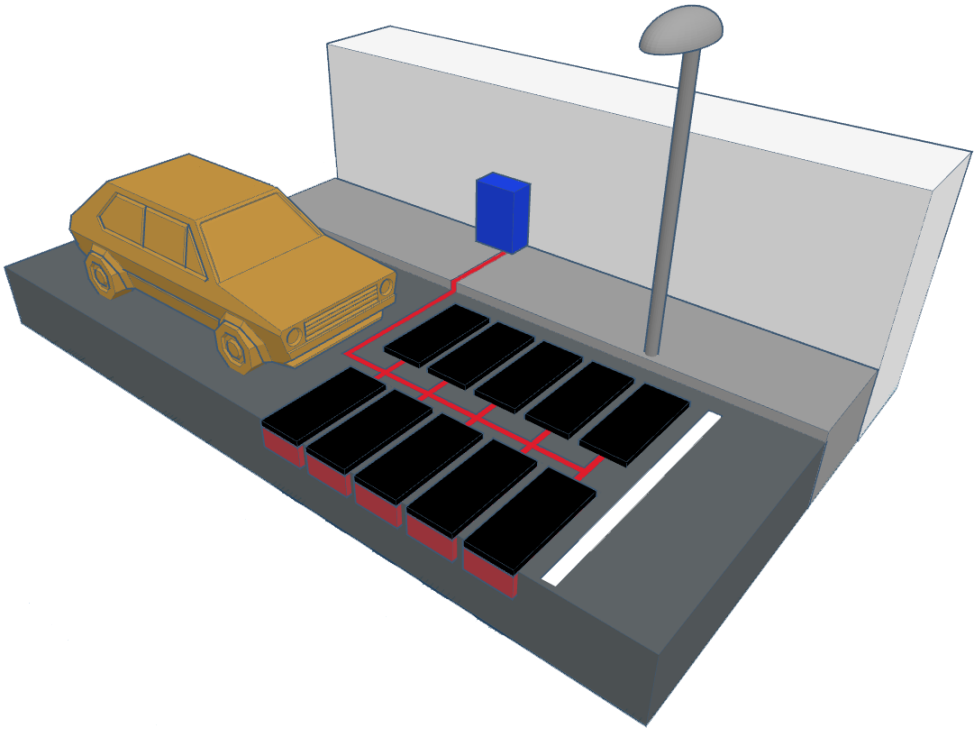


Figura 3 Ilustração do problema

1.3. OBJETIVOS

O sistema será constituído por uma rede de microcontroladores e sensores, interligados cumprindo uma hierarquia específica, dividida em 4 diferentes níveis, tal como representado na Figura 4. Cada um destes níveis, terá um papel importante para o sistema, e terá associado diferentes tipos de dados.

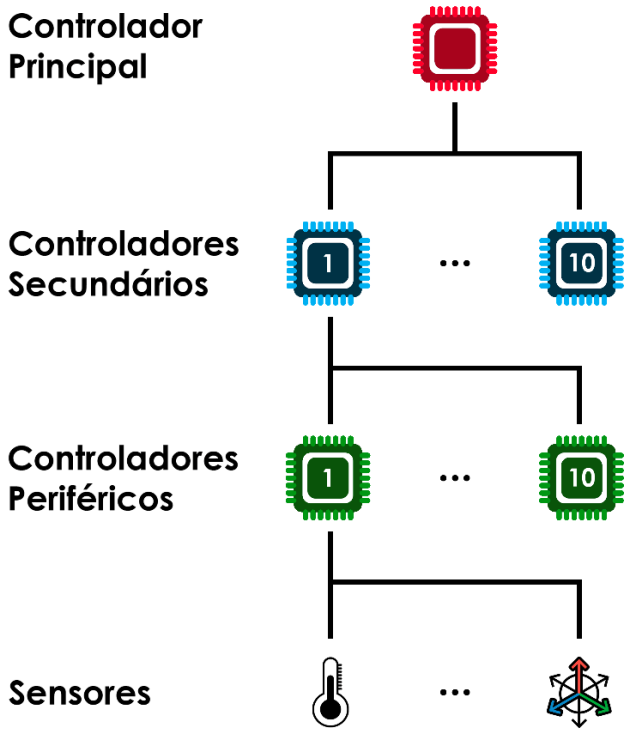


Figura 4 Diagrama de Hierarquia de Controladores

Começando pelo nível mais baixo, aqui encontrar-se-ão todo o tipo de sensores que vão obter os dados do ambiente e de monitorização dos módulos, sendo eles a deteção e peso dos veículos, temperatura e humidade internas do módulo e os dados relativos à energia gerada com a passagem dos veículos. A recolha deste tipo de dados será essencial para que, posteriormente, possam ser calculados e determinados diferentes tipos de dados indiretos, como, por exemplo, a velocidade dos veículos.

O seguinte nível, designado de Controladores Periféricos, irá compreender os microcontroladores instalados individualmente em cada um dos módulos *NEXT-road*. A ideia será que os sensores, correspondentes ao nível hierárquico inferior, introduzidos em cada módulo, comuniquem diretamente com este microcontrolador, e enviem a informação captada para que esta seja processada.

Por cada conjunto de 10 Controladores Periféricos será instalado um Controlador Secundário, correspondente ao seguinte nível hierárquico, que irá recolher a informação proveniente dos Periféricos. É neste nível que serão calculados dados como as velocidades iniciais e finais dos veículos. Após analisada e filtrada, a informação útil será transmitida para o último nível hierárquico, onde se encontra o Controlador Principal. Este dispositivo será responsável por agregar toda a informação do sistema, e realizar a comunicação sem fios com o exterior.

Utilizando esta abordagem hierárquica, a implementação e desenvolvimento de todo o sistema será facilitada, pois irá permitir que este seja realizado por etapas. Em primeiro lugar irá ser feito um estudo inicial para perceber que dados são necessários adquirir e monitorizar, assim como a que frequências estes necessitam de ser adquiridos. Ao mesmo tempo, o mercado de microcontroladores e sensores será analisado, para perceber que tipo de *hardware* existe, está disponível e é mais indicado para a implementação neste sistema.

Assim que o estudo prévio seja concluído, irá ser feita uma seleção de microcontroladores, sensores e restantes circuitos de instrumentação necessários para começar a implementação em uma fase de testes. Após estruturado o sistema inicial, com todos os componentes necessários para realizar as leituras e comunicar entre dispositivos, utilizando um *firmware* base, avançar-se-á para o desenvolvimento das PCB (Placas de Circuito Impresso ou *Printed Circuit Board*) para cada um dos diferentes controladores. A implementação dos circuitos em PCB permitirá eliminar erros e problemas associados a más conexões e contactos, devido à utilização de *breadboards*.

Em suma, os objetivos e requisitos para este projeto podem ser descritos pelos seguintes pontos:

- Estudo preambular - especificação de dados a monitorizar e frequências de aquisição; protocolos de comunicação, microcontroladores e sensores a utilizar;
- Projeto do Controlador Periférico – *hardware* e *firmware* - a introduzir nos módulos *NEXT-road* da Pavnext;
- Projeto do Controlador Secundário – *hardware e firmware* – a conectar a um conjunto de 10 Controladores Periféricos;
- Colaboração no projeto do Controlador Principal – desenvolvimento do *firmware* de comunicação para conectar a um conjunto de 10 Controladores Secundários;
- Validação experimental de laboratório e de campo.

1.4. CALENDARIZAÇÃO

Este projeto foi desenvolvido ao longo de, sensivelmente, um ano. De forma a demonstrar o trabalho realizado ao longo dos vários meses, foi construída a Tabela 1**Erro! A origem da referência não foi encontrada.**, com alguns dos pontos principais de todo o projeto.

Tabela 1 Calendarização

Tarefas	Out 20	Nov 20	Dez 20	Jan 21	Fev 21	Mar 21	Abr 21	Mai 21	Jun 21	Jul 21	Ago 21	Set 21	Out 21
Estudo do Projeto													
Testes de sensores e comunicação entre microcontroladores													
Desenvolvimento da estrutura de endereços CANFD													
Firmware do Controlador Periférico													
Firmware dos Controladores Principal e Secundário													
Testes de laboratório com breadboards em protótipos NEXT-Road													
Desenvolvimento de PCBs v1 dos Controladores Secundário e Periférico													
Testes com PCBs e sistema completo													
Desenvolvimento e testes de PCB v2 do Controlador Secundário em protótipos NEXT-road													
Testes no terreno e ajustes finais													
Escrita do Relatório													

1.5. ORGANIZAÇÃO DO RELATÓRIO

Este relatório está dividido em 9 Capítulos, cada um com os respetivos subcapítulos, de forma estruturar e dividir melhor toda a informação.

No Capítulo 1 começa-se por fazer um enquadramento do tema das *smart cities* e do controlo de tráfego rodoviário de forma a introduzir o projeto e a proposta apresentada pela empresa. Além disso, são também apresentados os objetivos do projeto e a sua calendarização.

No segundo capítulo é apresentado um estudo teórico a diferentes tipos de microcontroladores e IDEs de alguns dos fabricantes mais presentes no mercado. Desta forma, foi possível fundamentar a escolha dos vários microcontroladores escolhidos para este projeto.

O estudo teórico prolonga-se até ao terceiro capítulo, onde são apresentados e estudados diferentes tipos de protocolos de comunicação, com possibilidade de serem utilizados neste projeto.

No Capítulo 4 são apresentados os requisitos do sistema de monitorização e controlo de dados a desenvolver.

A arquitetura do sistema é apresentada no quinto capítulo. Além da arquitetura geral, este capítulo subdivide-se e descreve as arquiteturas de cada um dos tipos de controlador, assim como dos barramentos CANFD desenvolvidos e da estrutura de IDs das mensagens.

Os Capítulos 6 e 7 são os mais extensos deste relatório, e explicam como foram desenvolvidas as componentes de *hardware* e *firmware* do sistema, respetivamente. Em cada um deles é feita a divisão por cada tipo de controlador presente no sistema.

Por fim são apresentados os resultados dos diversos testes em diferentes fases de protótipo do sistema, feitos ao longo do decorrer do projeto, no Capítulo 8, e de seguida são apresentadas as conclusões do projeto, no Capítulo 9, onde se aborda quais os requisitos que foram cumpridos e futuros desenvolvimentos e melhoramentos que o sistema poderá sofrer.

2. SISTEMAS EMBEBIDOS

Um Sistema Embebido (SE) pode ser caracterizado como um sistema computacional projetado para realizar uma tarefa específica. Tanto pode ser um sistema independente, como se pode englobar em um sistema maior, com ou sem mais sistemas embebidos. Este tipo de sistema é, geralmente, um conjunto de *hardware* construído em volta de um microprocessador, no qual é aplicado um *software* dedicado. A complexidade de um SE varia consoante a sua aplicação, podendo este estar conectado a mais ou menos periféricos, ter ou não interfaces gráficas e ser mais ou menos preciso nas suas tarefas [12], [13].

Além das particularidades descritas anteriormente, para que seja considerado um Sistema Embebido, este deve conter as seguintes características:

- Limitações em termos de custo, tamanho, potência e/ou desempenho, que devem ser cumpridas;
- Baseados em microprocessadores ou microcontroladores;
- Compatibilidade com diferentes microprocessadores e arquiteturas;
- Requisitos de tempo real apertados (utilizam muitas vezes sistemas operativos de tempo real (RTOS));

- Memória, normalmente ROM, onde é guardado o *software*;
- Conectado a periféricos de entrada e de saída;
- Trabalhar em condições ambientais adversas.

A estrutura de um SE é definida por diferentes componentes de *hardware* desde os sensores até aos atuadores. Os sensores medem e transformam uma grandeza física em um sinal elétrico. Existe uma vasta gama de sensores para todo o tipo de leituras, quer seja temperatura, humidade, luminosidade, ruido, movimento, entre outros. O sinal produzido pelo sensor é posteriormente injetado em um Conversor Analógico-Digital (ADC) que, tal como o nome do componente indica, irá transformar o sinal elétrico em um sinal lógico capaz de ser lido pelo processador. Além do *input* de dados, o processador é capaz de fazer *output* de um sinal lógico, com o intuito de controlar um atuador. Este sinal é, primeiramente, enviado para um Conversor Digital-Analógico (DAC), que realiza a função oposta do ADC. Finalmente, após convertido o sinal lógico para um sinal elétrico, este será enviado para o atuador, que irá executar uma ação física [12]–[15].

Hoje em dia, o contacto com SE é inevitável, estando estes presentes em todo o tipo de locais e equipamentos, desde áreas como os transportes, telecomunicações, *smart buildings*, agricultura, saúde, segurança, entre inúmeros outros exemplos [12].

2.1. MICROCONTROLADORES

Um microcontrolador é um pequeno circuito integrado capaz de realizar operações, e controlar o sistema no qual está incorporado. Este é normalmente constituído por um processador, memórias e um conjunto de periféricos que o permitem comunicar com os restantes circuitos integrados presentes no sistema. Desta forma, é umas das peças chaves para o desenvolvimento de um SE.

Não existe um microcontrolador ideal para cada tipo de aplicação ou SE, sendo que existem milhares de produtos no mercado, das mais variadas empresas e com as mais variadas características. No entanto, o mercado dos microcontroladores divide-se em diferentes gamas, destinadas a diferentes níveis de exigências. A arquitetura, o consumo energético, a frequência, a memória, os periféricos, a performance são algumas das características que distinguem diferentes tipos de microcontroladores [13], [16].

2.1.1. ARQUITETURA ARM

A arquitetura ARM (*Advanced RISC Machines*) é um conjunto de arquiteturas dedicadas a processadores, que usam como base uma arquitetura RISC. Atualmente, é a arquitetura mais usada do mundo, tendo sido produzidos cerca de 180 biliões de processadores ARM até 2021 [17], [18]. Embora possam ser encontrados mais facilmente em dispositivos menores, como *smartphones* e/ou sistemas embebidos, esta arquitetura é, também, utilizada em computadores, servidores e, inclusive, no maior supercomputador do mundo [19]. Algumas vantagens de trabalhar com um processador deste tipo são o equilíbrio de performance vs. eficiência energética, segurança e a quantidade de informação disponível sobre esta arquitetura, o que ajuda no desenvolvimento de aplicações [20].

Existem inúmeras versões da arquitetura ARM, sendo estas divididas em diferentes famílias, desde os ARM7 até às famílias *Cortex*, dentro das quais os processadores partilham bastantes das suas características. No que toca a microcontroladores, a família de arquiteturas mais comum de ser utilizada é a ARM *Cortex-M*, devido, em parte, às suas otimizações de consumo energético e custo [21].

Apesar de projetar uma gama enorme de arquiteturas, a ARM não fabrica os seus processadores. Em contrapartida, a empresa licencia os seus projetos a outras gigantes do mundo da eletrónica, como por exemplo *Apple* [22], *Huawei* [23], *Microchip* [24], *Qualcomm* [25], *STMicroelectronics* [26], entre muitas outras. Desta forma, as empresas podem desenvolver o seu *hardware* incorporando o processador ARM. Tal como exemplificado na Figura 5, de todos os componentes deste microcontrolador, apenas o processador é desenvolvido pela ARM.

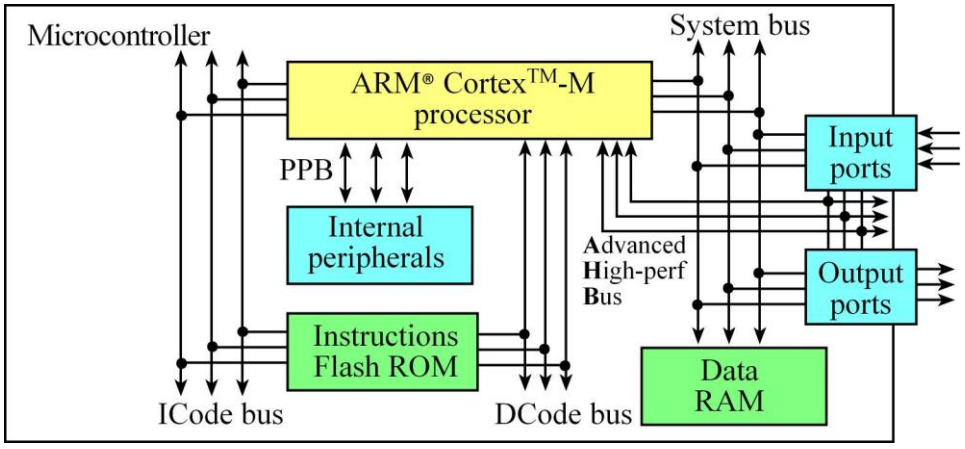


Figura 5 Arquitetura de um microcontrolador ARM *Cortex-M* [27]

2.1.2. ANÁLISE DE DIFERENTES MICROCONTROLADORES / MODELOS E MARCAS

A escolha de um microcontrolador para uma determinada aplicação é uma tarefa que envolve um bom conhecimento de mercado. Esta escolha pode ser facilitada caso sejam estabelecidos alguns pré-requisitos, de forma a diminuir a gama de microcontroladores que se encaixem corretamente no estipulado.

No que toca ao processador, atualmente, o mais comum é encontrar um modelo ARM instalado no microcontrolador, contudo, nem todas as fabricantes optam por utilizar esta arquitetura. Deste modo, encontram-se no mercado outro tipo de arquiteturas, como é o caso dos processadores *Xtensa* [28], utilizados, por exemplo, em alguns modelos de microcontroladores fabricados pela *Espressif* [29]. Diretamente associado ao processador, está a frequência de operação, que deriva do cristal oscilador utilizado, podendo este ser interno ou externo.

As diferentes memórias presentes num microcontrolador são outro ponto determinante na escolha de um produto que cumpra critérios estabelecidos para o projeto. Num microcontrolador, podem ser encontradas três tipos de memórias: ROM ou *Flash*, *SRAM* e *EEPROM*. A memória ROM ou *Flash* é responsável por armazenar o programa do microcontrolador, não sendo alterada aquando do funcionamento do mesmo [30].

De modo a facilitar a escolha dos microcontroladores a utilizar neste projeto, foi feita uma pesquisa de diferentes marcas e respetivas famílias de microcontroladores. Alguns dos dados obtidos foram compilados em uma tabela, que pode ser consultada no **Erro! A origem d a referência não foi encontrada.**.

2.2. IDES

Como forma de avaliar as diferentes opções de microcontroladores presentes no mercado, para implementar neste projeto, foi feita, a par da análise de características de *hardware* dos mesmos, uma análise a alguns IDE (*Integrated Development Environment*) de diferentes empresas e destinados à programação de diferentes microcontroladores. Optou-se, assim, por analisar os seguintes *softwares*:

- *DAVE 4.0* – IDE desenvolvido pela *Infineon* [31];
- *MPLAB X IDE* – IDE desenvolvido pela *Microchip* [32];

- *STM32Cube* – IDE desenvolvido pela *STM* [33].

2.2.1. DAVE 4.0

Começando pelo *DAVE 4.0*, este é um IDE gratuito, desenvolvido pela *Infineon*, e destina-se à criação de *software* e programação de microcontroladores da empresa, como é o caso da família XMC. Utiliza como base o IDE *Eclipse* e inclui funções como: *GNU C-compiler*, *debugger*, *addons* de geração de código, repositórios com exemplos de código e implementações e ferramentas para gestão de recursos de *hardware*. Este *software* está disponível apenas para sistemas operativos *Windows* de 32 e 64 *bits* e pode ser obtido através do *website* oficial da *Infineon*. Após o preenchimento de um formulário, é enviado um email para o endereço indicado, com o *link* para *download*.

Ao contrário do habitual, o *software* disponibilizado para *download* funciona como uma versão portátil, não sendo necessário fazer a instalação no computador. Para iniciar o IDE, basta executar o ficheiro "*DAVE.exe"* presente na pasta "*eclipse*". Ao iniciar o executável, é pedido para escolher um diretório para funcionar como *workspace*, ou seja, onde serão armazenados todos os projetos e respetivos ficheiros. Além dos ficheiros integrantes do *software*, são também fornecidos no *download*, alguns documentos com informações, como por exemplo, *release notes*, detalhes da instalação, ajuda inicial e licença de utilização.

Após iniciado o *software*, é apresentada a sua janela inicial. Esta está dividida em vários painéis, identificados na Figura 6 com diferentes números e cores:

1. Barra de ferramentas - dá acesso rápido e prático a diferentes funções importantes do IDE;

2. Explorador do Projeto – apresenta todos os projetos e respetivos ficheiros e diretórios;

3. Perspetivas – permite alterar o aspeto e o conteúdo visualizado na janela principal. Por padrão, é possível alterar entre as perspetivas IDE, CE e *Debugger*, mas é possível adicionar novos atalhos para diferentes perspetivas;

4. Editor de código e Configurador de *apps* – neste painel é possível editar o código dos diversos ficheiros do projeto e/ou editar as configurações das diversas *apps* utilizadas;

5. Árvore de dependências das *apps* – neste painel é possível visualizar, através de um diagrama, todas as *apps* instaladas no projeto e a ligação de dependência entre elas. Clicando nas *apps*, abre-se no painel número 4, a janela de configuração da mesma;

6. Dependências de *apps* e conectividade de sinais de *hardware* – este painel só está disponível para projetos CE e permite visualizar tanto as *apps* e as suas dependências, em um formato diferente do painel 5, como permite visualizar as ligações de *hardware* entre os diversos componentes referentes às *apps* instaladas e configuradas.

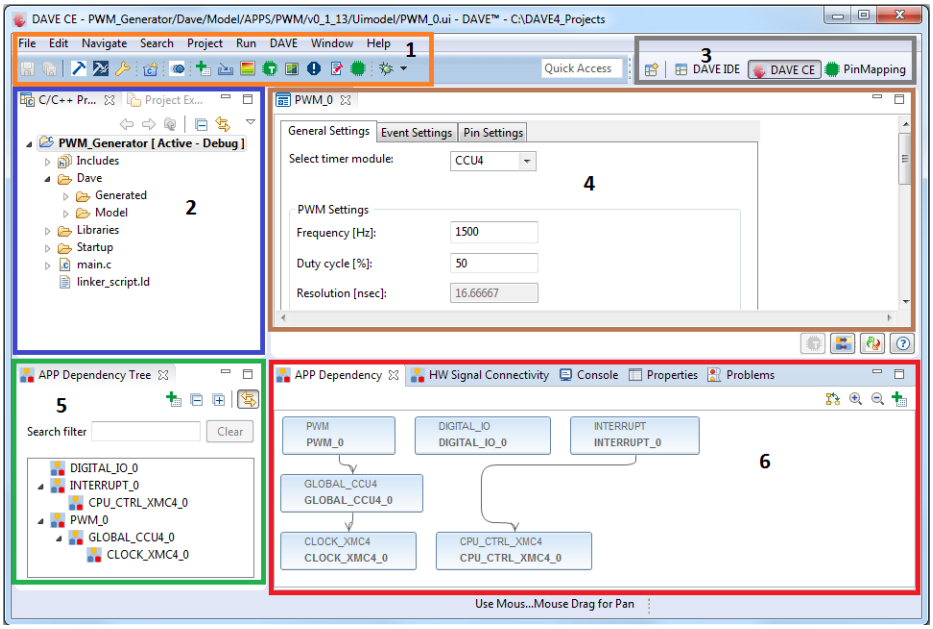


Figura 6 Janela principal - *DAVE 4.0*

Para iniciar um novo projeto, o utilizador deve utilizar o menu "*File* > *New* > *DAVE Project*", abrindo assim a janela de criação de projeto. Nesta janela é introduzido o nome e escolhido o tipo de projeto, de várias aplicações possíveis. Por fim, o utilizador deve indicar qual o microcontrolador que pretende utilizar, da lista disponibilizada pelo *software*.

Para facilitar na programação e configuração de opções de *hardware*, como *timers*, protocolos de comunicação, pinos IO, entre outros, o *DAVE 4.0* apresenta uma solução com

base em *apps*, pequenos *add-ons* com configurações pré-programadas para cada um dos periféricos presentes no microcontrolador. Estas podem ser utilizadas caso o projeto seja criado no modo "*DAVE CE Project*". A sua instalação pode ser feita através do menu "*Add new app*", onde é apresentada uma lista com diferentes opções para diferentes configurações (Figura 7).

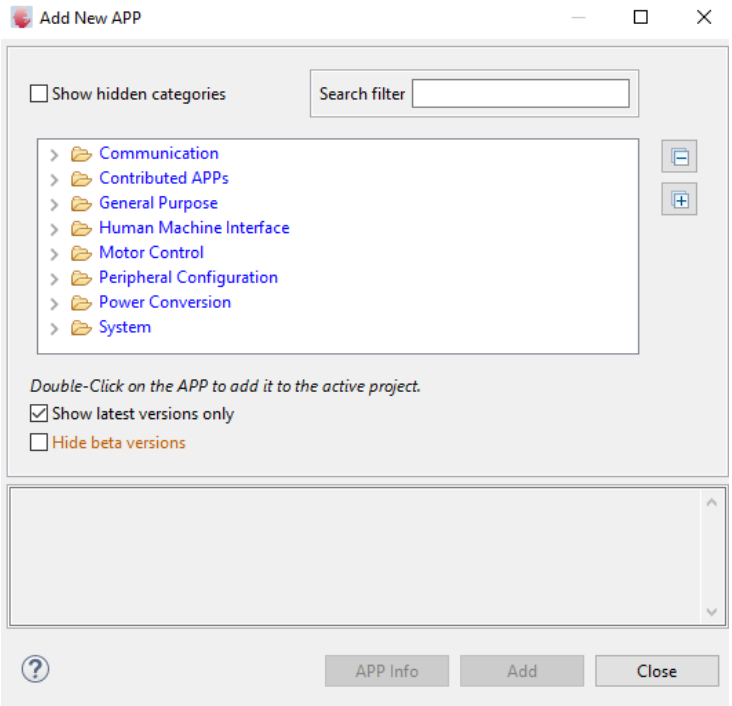


Figura 7 Janela "*Add new app*" - *DAVE 4.0*

Ao serem adicionadas *apps* ao programa, automaticamente surge a árvore de dependências, na metade inferior da janela do *DAVE 4.0*, de uma forma gráfica para que facilmente seja possível consultá-las (Figura 8).

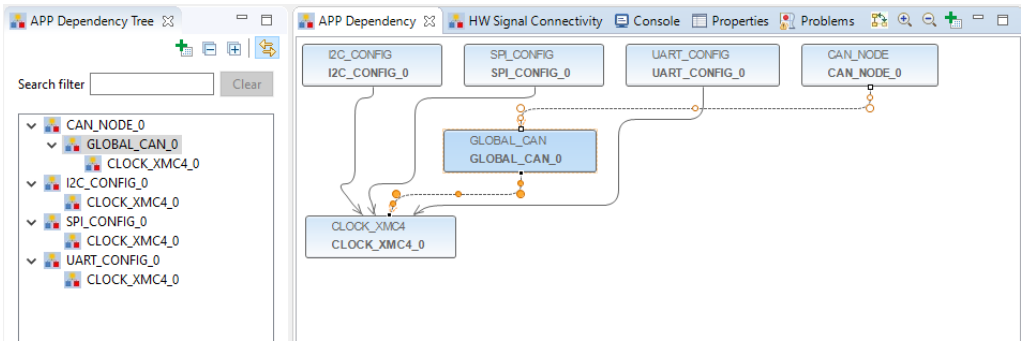


Figura 8 Arvore de Dependências de *APPs* - *DAVE 4.0*

Clicando nas diferentes *apps*, uma nova *tab* é aberta na metade superior da janela, onde é possível alterar as suas configurações, como se pode verificar na Figura 9 com as configurações da *app CLOCK_XMC4_0*.

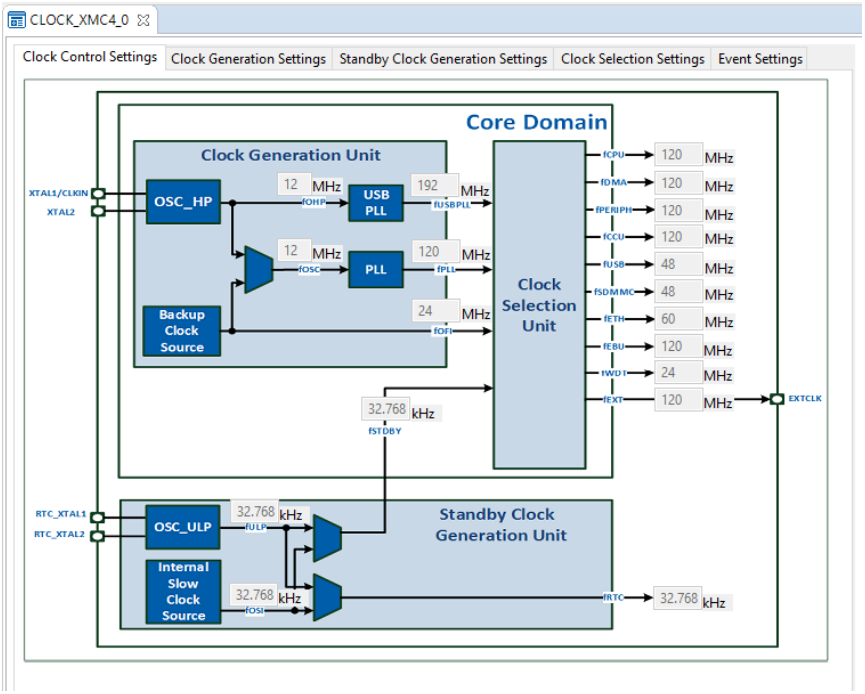


Figura 9 *Tab* de configurações da *app* de *clock – DAVE 4.0*

Presente neste IDE, existe também outra funcionalidade útil, o "*Manual Pin Allocator*", que permite ao utilizador escolher quais os pinos do microcontrolador que pretende usar para as funções das *apps*, previamente instaladas.

No momento em que o utilizador desejar programar o microcontrolador com o código desenvolvido, poderá utilizar a função "*Debug*" presente neste IDE, e assim correr o código ao mesmo tempo que verifica o estado do *hardware*, de *flags* e de variáveis.

De forma a testar este *software* de uma maneira mais prática, foi desenvolvido um pequeno programa e utilizou-se, para teste, uma placa de desenvolvimento com um microcontrolador XMC4500. O código consistia em alternar o estado lógico de um LED (*Light-emitting Diode*) presenta na placa. Depois de desenvolvido e compilado o programa, ligou-se a placa ao computador, através de um cabo USB e utilizou-se a função "*Debug*", presente no painel 1. No momento de fazer o *upload* do programa para o microcontrolador, uma mensagem de erro foi apresentada. Após uma extensa pesquisa sobre o possível motivo do erro, foi encontrada a origem do problema, estando esta relacionada com a falta do *driver* que permite

fazer o *debug* da placa de testes. Tanto a solução como o *driver* foram encontrados em *sites* que não o oficial da *Infineon*.

Concluindo, o IDE *DAVE* 4.0 demonstra ter uma curva de aprendizagem bastante acentuada, isto devido à dificuldade que existiu em encontrar informação oficial sobre como utilizar corretamente o *software*, assim como não foi encontrada nenhuma informação sobre o erro que aconteceu no momento da programação da placa, além de ajuda de outros utilizadores em fóruns comunitários.

2.2.2. MPLAB X IDE

Depois da análise ao *DAVE 4.0*, iniciou-se um estudo do seguinte IDE, o MPLAB X da *Microchip*. Ao contrário do IDE anterior, o MPLAB X utiliza como base o *software Netbeans*, em vez do *Eclipse*, e encontra-se disponível para *Windows*, *Linux* e *MacOs*. Por este motivo, o ambiente gráfico do IDE é bastante diferente, como se verifica pelo painel inicial da Figura 10.

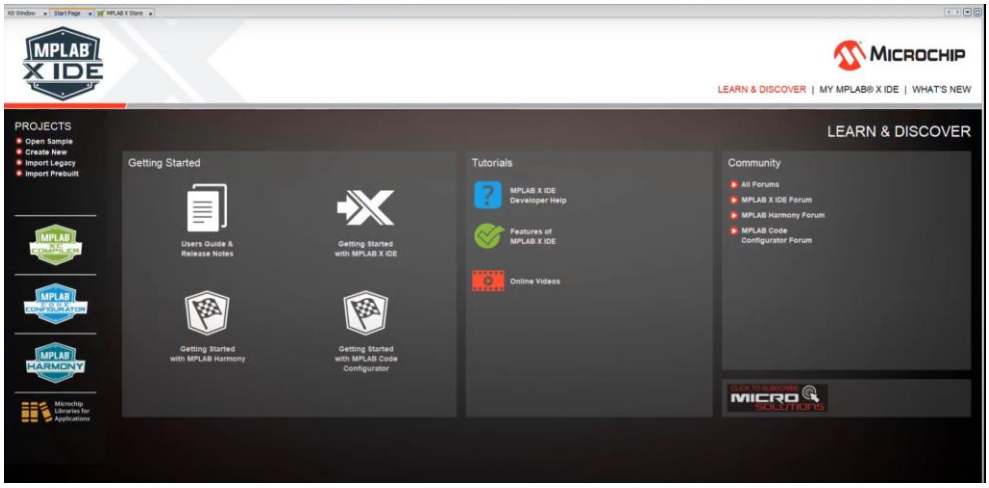


Figura 10 Painel inicial - MPLAB X IDE

Este *software* pode ser facilmente obtido através do *website* oficial da *Microchip*. No momento da instalação, é possível escolher quais bibliotecas de famílias de microcontroladores se pretende utilizar, permitindo assim que a instalação seja ligeira e não seja ocupado tanto espaço no disco com ficheiros que não serão utilizados. A qualquer momento, após a instalação do IDE, é possível instalar e desinstalar pacotes de microcontroladores através da função "*Packs Manager*" (Figura 11), não sendo necessário atualizar todo o *software* sempre que um produto novo for lançado para o mercado.

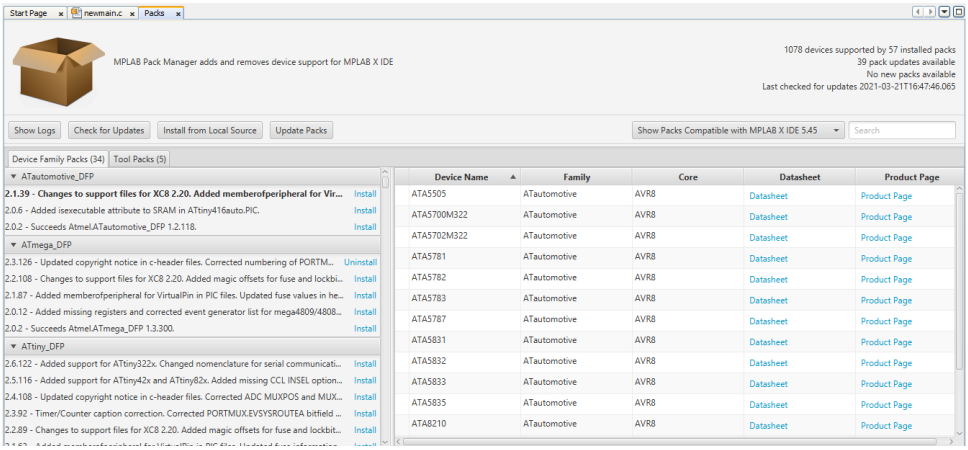


Figura 11 Janela *Packs Manager* – MPLAB X

Semelhante aos pacotes de microcontroladores, este *software* também permite instalar novas funcionalidades a gosto do utilizador, denominadas de *plugins*. De entre os *plugins* oficiais e os criados pela comunidade, o *Code Configurator* e o *Data Visualizer*, desenvolvidos pela própria *Microchip*, são dois dos mais úteis para utilizar no desenvolvimento e *debugging* do código.

O *Code Configurator* [34] (Figura 12), que existe também como *software* independente, é um *plugin* gráfico capaz de gerar blocos de código C de forma intuitiva, para determinado microcontrolador. Desta forma, a configuração de *timers*, interfaces de comunicação e outros periféricos, pode ser feita de forma rápida e prática, sem que o utilizador necessite de programar, individualmente, cada um dos bytes de memória que armazenam as configurações do periférico em questão.

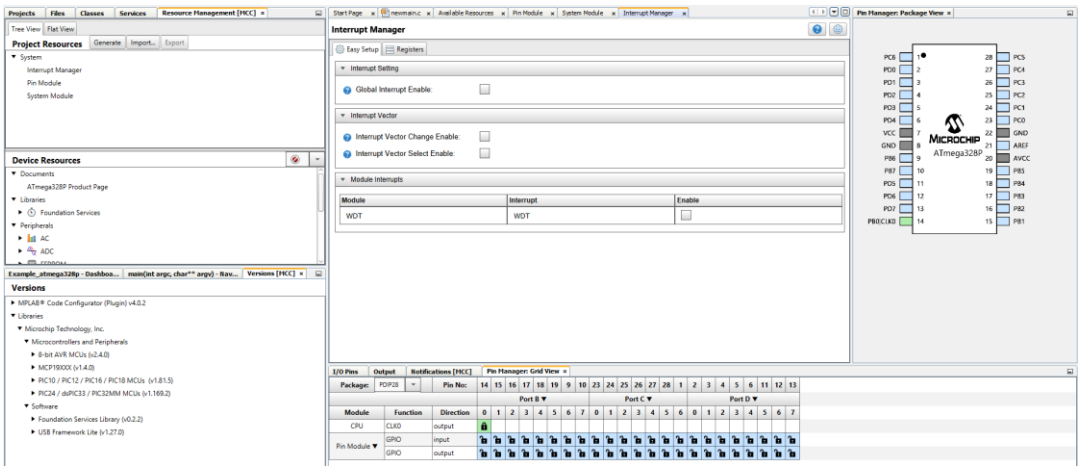


Figura 12 *Code Configurator* – MPLAB X

Através do *plugin Data Visualizer* [35] (Figura 13), são introduzidas mais ferramentas de *debugging* que podem ser utilizadas para, por exemplo, gerar gráficos com valores obtidos de leituras do microcontrolador. Tal como o *Code Configurator*, este pode ser instalado separadamente do MPLAB X, como um programa *standalone*.

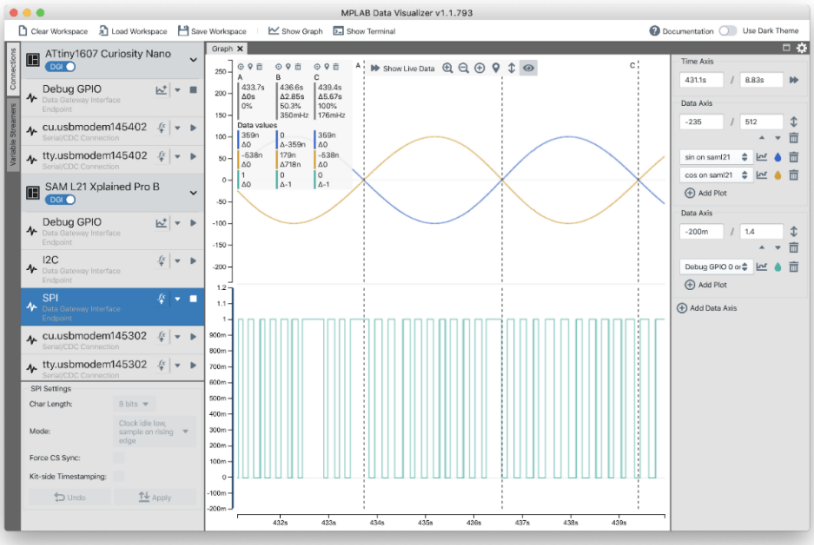


Figura 13 Exemplo de gráfico usando o *Data Visualizer* [35]

Tal como realizado anteriormente, com o IDE *DAVE 4.0*, foi utilizado um microcontrolador *atmega328p*, presente em uma placa ARDUINO UNO R3, com o intuito de validar o processo de programação do *software* MPLAB X. Da mesma forma que no exemplo anterior, foi feito um pequeno código que alternava o estado lógico de um LED. Primeiramente, foi necessário instalar o *pack* de microcontroladores da família do *atmega328p*, através da ferramenta "*Packs Manager*". Após instalado, apenas foi necessário utilizar a função de *debug* e o microcontrolador foi automaticamente programado, sem ter sido necessário instalar nenhum *software* adicional, tal como no IDE anterior.

2.2.3. STM32CUBE

O último IDE a ser analisado foi o *STM32CUBE* da *STMicroelectronics*, desenvolvido para programar a vasta gama de microcontroladores fabricados pela marca. Em semelhança ao *DAVE 4.0*, este *software* é baseado em *Eclipse*, logo, há aspetos em que os ambos os *softwares* são muito idênticos, como é o caso do aspeto da janela de trabalho. Este encontra-se disponível para as três plataformas principais (*Windows*, *Linus* e *OSX*).

O primeiro passo para obter este *software* é realizar o registo através do *website* da *ST*. Posteriormente, o utilizador irá receber um *email* contendo um *link* na qual pode fazer o *download* do programa. Na instalação, além dos termos de serviço e da escolha do diretório de instalação, o utilizador tem também a opção de adicionar dois *drivers* para os programadores dos microcontroladores, o que é recomendável.

Assim que o *software* é iniciado, é apresentada uma primeira janela onde o utilizador pode definir o diretório do *workspace* que pretende utilizar. Esta opção permite que sejam criados *workspaces* para diferentes projetos, o que melhora a organização. Depois disso será finalmente apresentada a janela principal do *software*. Neste caso de estudo, sendo utilizado um microcontrolador *STM32*, optou-se pela opção "*STM32 Project*", disponível no menu *New*. Esta opção permite que o *software* crie um projeto dedicado ao microcontrolador pretendido, através do primeiro menu de seleção, como demonstrado na Figura 14.

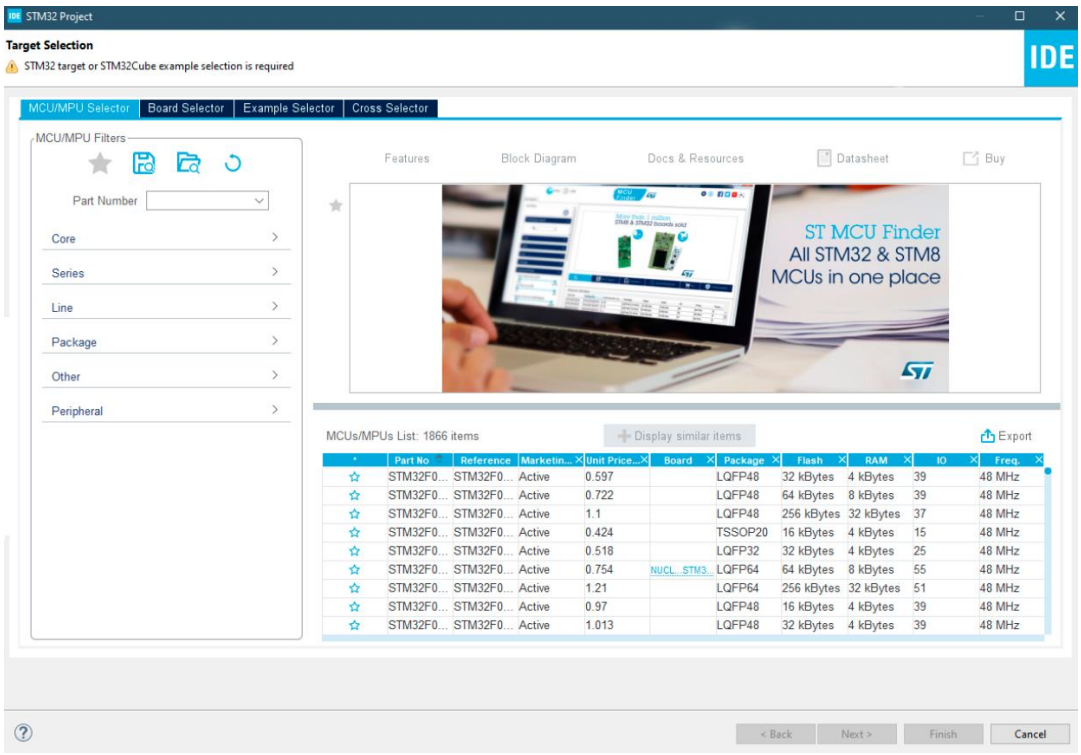


Figura 14 Janela de seleção de microcontrolador – *STM32Cube*

Após selecionado o microcontrolador e criado o projeto, é apresentada a janela de *Device Configuration Tool* (Figura 15), que permite ao programador, configurar de forma gráfica e sem recurso a código, os diferentes periféricos, *pinout* e funcionalidades do microcontrolador em questão. Depois de feitas todas as alterações necessárias, ao gravar o

projeto, o *software* irá gerar todo o código referente às novas funcionalidades e opções configuradas e adicionar aos existentes ficheiros de código já criados.

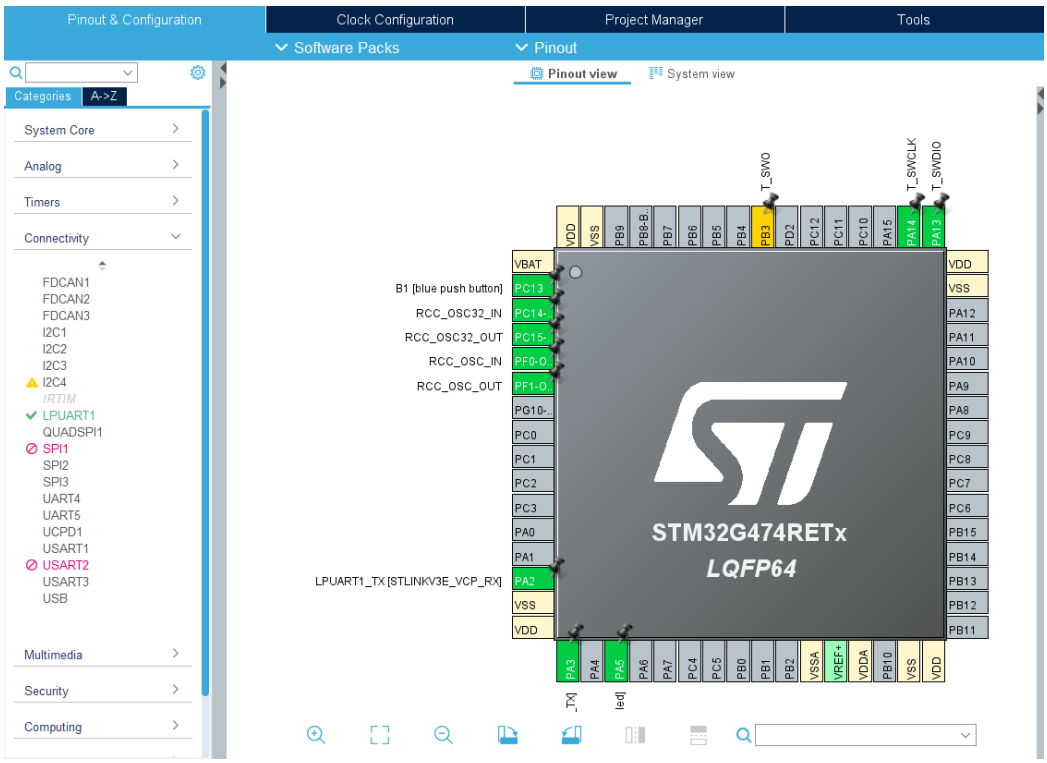


Figura 15 *Device Configuration Tool – STM32Cube*

Esta ferramenta também permite alterar os diferentes *clocks* do microcontrolador através de um diagrama de frequências, apresentado na opção *Clock Simulator* (Figura 16).

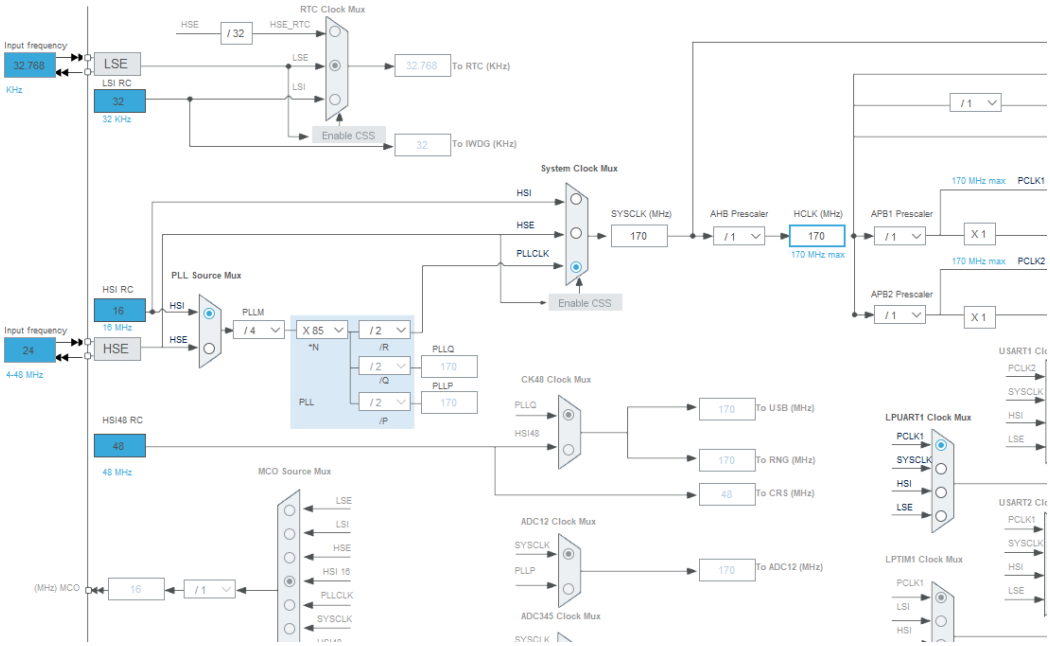


Figura 16 *Clock Simulator – STM32Cube*

As restantes funcionalidades do *software* assemelham-se às apresentadas pelo *DAVE 4.0*, como é o caso da barra de ferramentas na parte superior da janela (1) seguida do editor de perspetivas (2), editor de código no centro da janela (3) e o explorador de projeto à esquerda (4). Podem ser adicionados e alterados os diferentes painéis ao gosto do utilizador, que também podem apresentar diferentes funcionalidades consoante a perspetiva escolhida, por exemplo, C/C++ ou *Debug* (Figura 17).

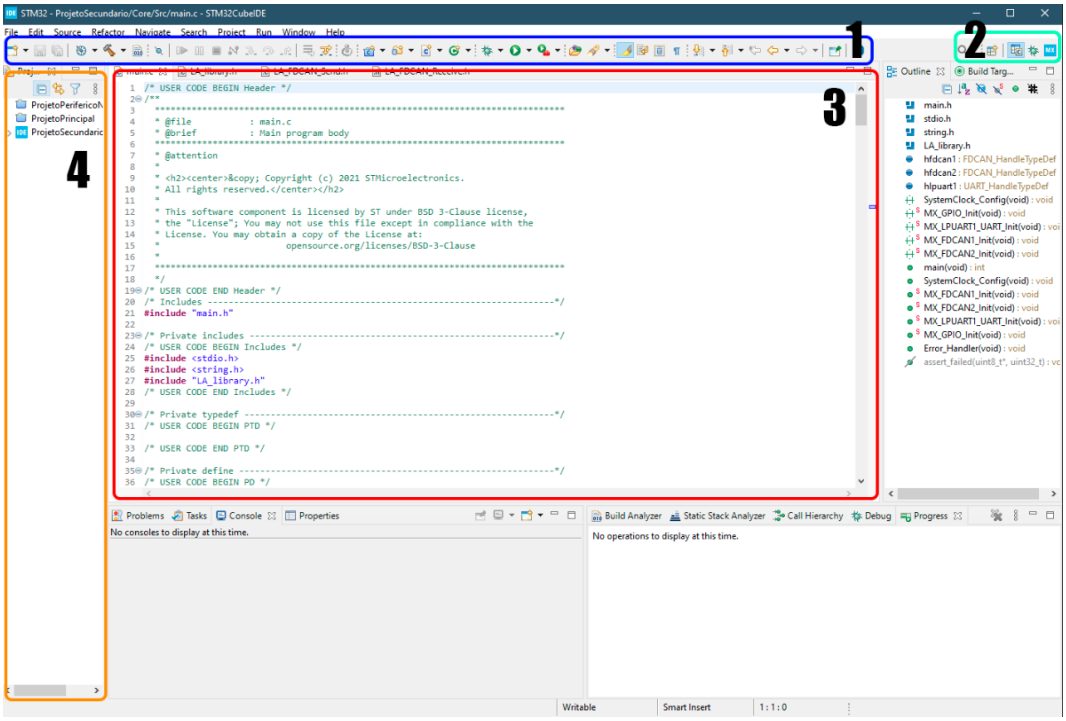


Figura 17 Janela Principal – *STM32Cube*

Para testar na prática este software, utilizou-se uma placa de desenvolvimento *NUCLEO-G474RE*, comercializada pela STM, e desenvolveu-se um programa idêntico ao desenvolvido nos restantes IDE, apresentados anteriormente.

2.2.4. CONCLUSÕES

Finalizada a análise a todos os IDE selecionados, para efeitos comparativos, foi construída a Tabela 2 com algumas características dos diferentes *softwares*, para facilitar a análise:

Tabela 2 Características de diferentes IDE

IDE	DAVE 4.0	MPLAB X	STM32Cube
Empresa	*Infineon*	*Microchip*	*STMicroelectronics*
Plataformas	*Windows 32/64-bits*	*Windows / Linux / OSX*	*Windows / Linux / OSX*
Download	Através de *email*	Direto do *website*	Através de *email*
Software base	*Eclipse*	*Netbeans*	*Eclipse*

Em termos de acesso ao *software*, nenhum dos três apresenta grandes dificuldades, sendo possível ter acesso ao mesmo em poucos minutos, com uma pesquisa nos respetivos *websites*. A avaliação dos diferentes IDE foi realizada em um computador *Windows*, não tendo existido problemas em termos de utilização dos mesmos, no entanto, o facto de o *DAVE 4.0* apenas estar disponível para esta plataforma, pode tornar-se impeditivo para alguns utilizadores, sendo o primeiro ponto negativo, deste IDE, face ao *MPLAB X* e ao *STM32Cube*.

Em termos de apresentação, por serem baseados em *Eclipse*, os IDE *DAVE 4.0* e *STM32Cube* apresentam um *layout* muito semelhante, distanciando-se do ambiente do *MPLAB X*. Ainda assim, as diferentes funções dos IDE encontram-se em locais correspondentes.

O processo de criação de um projeto é similar entre os três *softwares*, não existindo um aspeto que se destaque positiva ou negativamente. Dependendo do IDE utilizado, o processo poderá variar, contudo, os aspetos mais importantes passam por indicar o microcontrolador a utilizar e o nome a atribuir. Após a criação do projeto, os *softwares* apresentam diferentes formas para gerar o código de funções padrão dos microcontroladores. No caso do *DAVE 4.0*, esta geração pode ser feita através das *apps*, existindo uma *app* para

cada função do microcontrolador. Já o *MPLAB X* e o *STM32Cube* utilizam mecânicas semelhantes, através das funcionalidades *Code Configurator* e *Device Configuration Tool*, respetivamente. Nestas duas últimas, é exibida uma imagem do *pinout* do microcontrolador e as funções associadas a cada pino, assim como uma lista de possíveis configurações.

Para finalizar, cada um dos três IDE apresenta aspetos positivos e negativos, face aos restantes. O *MPLAB X* destaca-se positivamente pela facilidade de acrescentar *plugins*, tanto oficiais, como desenvolvidos pela comunidade. Esta funcionalidade pode tornar o *software* mais completo, e ao mesmo tempo, ao gosto do utilizador, sem que sejam instaladas funcionalidades adicionais inúteis para o projeto a ser desenvolvido. O *DAVE 4.0*, da *Infineon*, de um modo geral, demonstrou ter uma curva de aprendizagem mais acentuada, face aos outros IDE, o que se torna um aspeto negativo. O *STM32Cube* cumpre todas as funções a que se propõe, e que se espera de um IDE, não havendo pontos negativos a realçar.

3. PROTOCOLOS DE COMUNICAÇÃO

A escolha de um protocolo de comunicação é uma das decisões que têm de ser tomadas aquando do desenvolvimento de um sistema que envolva uma componente de *hardware*. Este define o tipo de sinal, a sua transmissão e a forma como é codificada a informação enviada.

Desde o aparecimento dos primeiros dispositivos eletrónicos, que diversos protocolos de comunicação têm sido desenvolvidos e adotados por diferentes empresas, e aplicados às mais diversas áreas da ciência e tecnologia. De maneira a conhecer melhor o panorama atual, irá ser feita uma análise de diversos protocolos de comunicação com fios e uma comparação entre as suas características. O objetivo deste estudo será perceber quais os protocolos que melhor se ajustam ao objetivo do trabalho e às necessidades que este impõe.

3.1. CONCEITOS BASE

Antes de avançar para o estudo dos protocolos de comunicação é necessário abordar alguns conceitos base, que serão abrangentes à maioria dos protocolos.

3.1.1. COMUNICAÇÃO SÉRIE E PARALELO

A comunicação em série ou paralelo está relacionada com a forma como os dados são transmitidos e recebidos entre os vários dispositivos envolvidos na comunicação. Esta característica define de que modo está implementado o canal de transmissão de dados.

Numa comunicação série, os dados são enviados *bit* a *bit* através de uma única linha de transmissão. A ordem com que os *bits* são enviados é importante, para que o recetor consiga descodificar corretamente a informação recebida (Figura 18) [36].

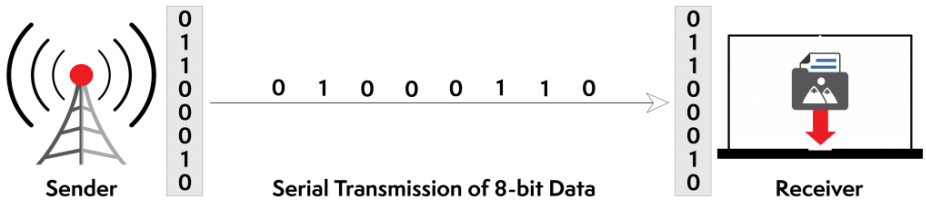


Figura 18 Transmissão de dados em Série [36]

Em contraste, na comunicação em paralelo, os dados são transmitidos em conjuntos de n *bits* através de n vias de comunicação. Este método permite que o dispositivo envie ou receba n *bits* de informação em simultâneo (Figura 19) [36].

Figura 19 Transmissão de dados em Paralelo [36]

À primeira vista, os protocolos com comunicação em paralelo parecem apenas apresentar vantagens face aos protocolos série, pois permitem um maior fluxo de dados por instante de tempo. Na prática, isto não se verifica, pois, as comunicações em paralelo

também apresentam limitações que, por consequência, reduzem a sua velocidade face às comunicações em série.

Em primeiro lugar, existe a necessidade da informação ser recebida toda ao mesmo tempo, por parte do recetor, isto é, todas as vias paralelas têm de estar sincronizadas e fazer chegar os dados no mesmo instante. Ao aumentar a velocidade de transmissão, aumenta-se também a dificuldade de manter todas as linhas sincronizadas.

Outra desvantagem das comunicações em paralelo é um fenómeno denominado de *crosstalk* que pode afetar negativamente o canal de transmissão de dados. Este fenómeno nada mais é que interferência elétrica criada entre as diferentes linhas do canal. Neste caso, o aumento do número de linhas e/ou o aumento da velocidade de transmissão estão diretamente relacionados com o aumento desta interferência nas comunicações.

No caso das comunicações série, o *crosstalk* não é tão comum devido ao número de linhas de dados ser reduzido a uma ou duas linhas, no caso de haver uma linha para cada direção de comunicação. Neste caso, a velocidade de transmissão pode ser aumentada drasticamente, face às comunicações em paralelo, pois não existe a necessidade de sincronismo entre linhas, nem preocupações com o *crosstalk*.

Embora existam diferenças claras entre o funcionamento dos dois modos de comunicação, cada um apresenta as suas vantagens dependendo da aplicação, sendo ambos utilizados, atualmente, nos mais diversos casos.

Um dos casos onde facilmente se encontram implementações em paralelo, são nos displays de vários tipos de dispositivos, como por exemplo *smartphones*. Neste caso, o maior fluxo de dados em simultâneo permite implementar características como o aumento do *refresh rate* dos displays [37].

Em termos de conectores, as comunicações em paralelo apresentam desvantagens face às comunicações série, sendo elas o tamanho dos conectores, geralmente superior, e o maior número de pinos/vias, tornando-se mais propício a danos e a desgaste. No caso dos computadores pessoais, verifica-se que, praticamente todos os conectores, utilizam interfaces de comunicação série. Alguns exemplos são: USB (*Universal Serial Bus*), SAS (*Serial Attached Small Computer System Interface*), SATA (*Serial AT Attachment*) e PCIe (*Peripheral Component Interconnect - Express*). No lado das comunicações em paralelo existem interfaces como IEEE 1284 e IDE *Cable*.

3.1.2. COMUNICAÇÃO SÍNCRONA E ASSÍNCRONA

Uma comunicação também pode ser classificada como síncrona ou assíncrona. A diferença entre estes dois tipos está na parametrização dos tempos de transmissão.

Em uma comunicação síncrona (Figura 20), os *bits* são transmitidos em instantes de tempo bem definidos, sendo necessária a sincronização entre *Controller* e *Peripheral*. Esta sincronização poderá ser feita através de uma linha de *clock*. Com este método, as transmissões têm a vantagem de ser mais rápidas e serem menos propícias a erros[38].

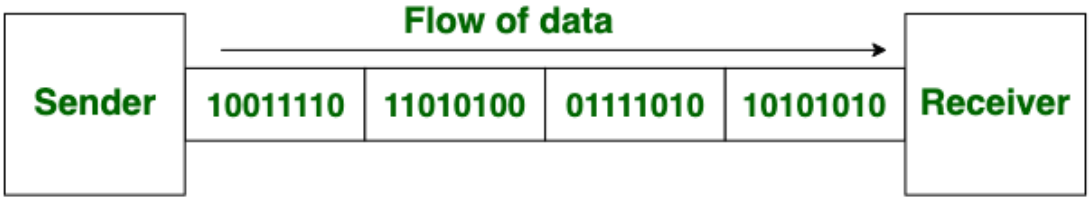


Figura 20 Transmissão Síncrona

No caso da comunicação assíncrona (Figura 21), a transmissão de dados é feita sem ser necessária uma sincronização entre os dispositivos *Controller* e *Peripheral*. Os dados são normalmente enviados *byte* a *byte* juntamente com uma *flag* (*start bit* se for no início do pacote de dados ou *stop bit* se for no fim). Estes *bits* têm a objetivo de identificar o início e o fim do pacote de dados. As vantagens deste tipo de comunicação face à comunicação síncrona, é não necessitar de uma linha de *clock* para sincronizar ambos os dispositivos envolvidos na comunicação [38].

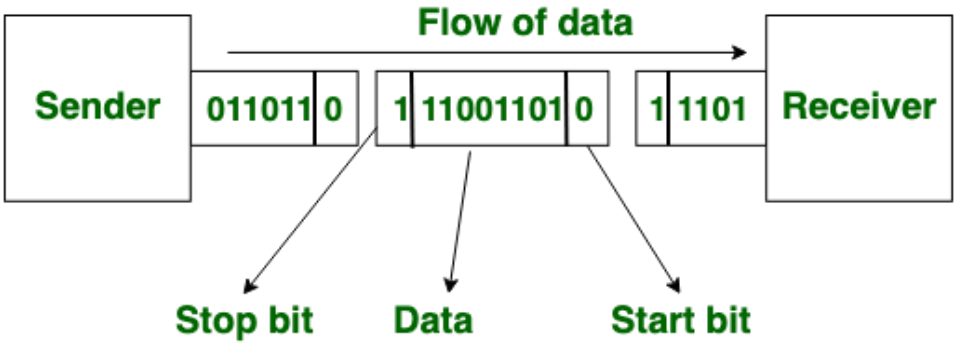


Figura 21 Transmissão Assíncrona

3.1.3. *CONTROLLER* E *PERIPHERAL*

Controller e *Peripheral* são dois termos utilizados para identificar os dispositivos face à sua posição de controlo em uma comunicação. O *Controller* é quem inicia e controla a transmissão de dados enquanto o *Peripheral* age consoante o que lhe é pedido. Dependendo do protocolo utilizado, um sistema pode ser *multi-controller*, quando existe mais do que um dispositivo capaz de iniciar comunicações. Neste caso, um mesmo dispositivo pode tomar a posição de *controller* e/ou *peripheral*. O sistema pode também ser considerado *multi-peripheral* quando existe mais do que um dispositivo deste tipo.

3.1.4. *FULL-DUPLEX* E *HALF-DUPLEX*

O termo *Duplex* é utilizado para identificar comunicações bidirecionais feitas entre dois dispositivos. Desta maneira, os prefixos *Full* e *Half* são atribuídos dependendo da simultaneidade da transmissão (Figura 22).

Uma comunicação é considerada *Full-Duplex* quando ambos os dispositivos envolvidos são capazes de receber e transmitir dados simultaneamente, como é o caso de uma chamada de telecomunicações. No caso da comunicação *Half-Duplex*, ambos os dispositivos podem agir como recetores ou transmissores (*Duplex*) mas não simultaneamente, por exemplo o sistema *Walkie-Talkie*.

Paralelamente a este tipo de comunicações existem outros tipos de sistemas que, mesmo estabelecendo uma comunicação entre dois dispositivos, esta não se considera *Duplex* pois não é feita em ambas as direções, o que acontece em casos como monitores ou televisões. A este tipo de comunicação dá-se o nome de *Simplex*.

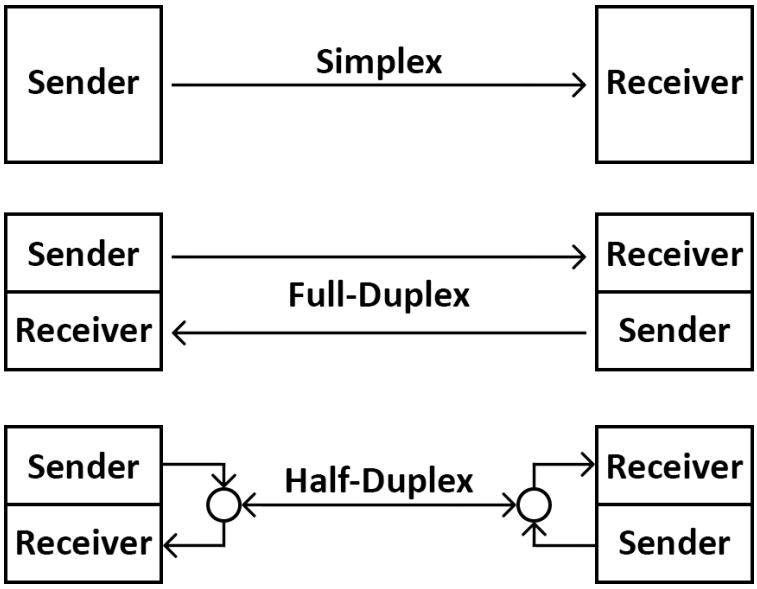


Figura 22 Diferentes tipos de sistemas *Duplex*

3.2. I^2C

O protocolo I^2C (*Inter-Integrated Circuit*) caracteriza-se como sistema de barramento a 2 fios, série e bidirecional, desenvolvido pela *Philips Semiconductors* (agora NXP) em 1982. Aquando do seu desenvolvimento, o objetivo seria a criação de um barramento simples e eficiente para comunicação entre diferentes circuitos integrados. Atualmente, é um protocolo facilmente implementado com a grande maioria dos microcontroladores, consegue atingir velocidades de até 5 Mbit/s e é utilizado em bastantes dispositivos/periféricos como ADCs, DACs, LCDs e diversos sensores [39]–[42].

3.2.1. HISTÓRIA E VERSÕES

Ao longo dos anos, o I^2C sofreu diversas atualizações tendo em vista o seu melhoramento e aumento de funcionalidades. Inicialmente, este era um protocolo destinado a uso interno da *Philips*, com uma velocidade de transmissão de apenas 100Kbit/s.

Em 1992 deu-se a primeira atualização do protocolo, tendo sido marcada pela adição de um novo modo de funcionamento, *fast-mode* (Fm), permitindo atingir velocidades de comunicação de até 400 Kbit/s, modo de endereçamento até 10 *bits* e pela sua padronização.

A segunda revisão deste protocolo adicionou outro modo de funcionamento, *high-speed mode* (Hs), aumentando substancialmente a velocidade máxima de transmissão para 3.4 Mbit/s, em 1998 [43].

Uma terceira revisão surgiu mais tarde, já em 2007, adicionando o *fast-mode plus* (Fm+), com uma velocidade de transmissão de até 1 Mbit/s, mantendo a compatibilidade com os modos Fm e *standart-mode*, assim como uma tabela de identificação de dispositivos através de ID [44].

Por fim, a quarta revisão do I^2C adicionou o *ultra fast-mode* (UFm). Este modo consegue atingir velocidades de até 5 Mbit/s, e para tal, utiliza uma diferente versão do barramento, sem resistências de *pull-up* [45].

Atualmente, este protocolo encontra-se na sua 6ª versão, tendo as duas últimas (5ª e 6ª) sido limitadas a pequenas atualizações e correção de erros [42], [46].

3.2.2. DESCRIÇÃO

O protocolo I^2C utiliza duas linhas para a comunicação, ao qual estão conectados todos os dispositivos que pretendam utilizar o barramento: linha de dados SDA (*serial data*) e linha de relógio SCL (*serial clock*). Como apenas existe uma linha para a transmissão de dados, o protocolo limita-se a uma comunicação *half-duplex*. Os dispositivos conectados ao barramento podem ser utilizados como transmissores ou recetores de informação, sendo possível o mesmo dispositivo utilizar ambas as funções, embora alternadamente. Além das limitações em termos de endereçamento, o número de dispositivos no barramento também é limitado pela capacitância do mesmo (máximo 400 pF).

Este protocolo estabelece que ambas as linhas, SDA e SCL, funcionam em modo coletor aberto (*open-drain* no caso do MOSFET), necessitando de resistências de *pull-up* entre o barramento e a alimentação. A utilização deste método permite que o transístor, interno aos dispositivos e ligado a cada linha, controle o estado lógico das linhas, funcionando como um interruptor. Isto significa que quando o barramento não está a ser utilizado, o transístor mantém a linha desconectada do *ground*, logo, o nível de tensão nas linhas é alto, que corresponde a nível lógico "1", Quando se pretende obter o nível lógico "0", o transístor faz a ligação da linha ao *ground* através da ligação coletor-emissor (Figura 23) [47].

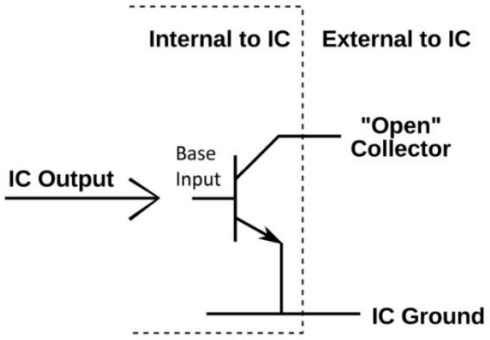


Figura 23 Esquema de ligação em coletor aberto [48]

Embora a tensão fornecida ao barramento (V_{DD}) seja, tipicamente, 3.3 ou 5 V, este aceita níveis de tensão mais elevados, assim como dispositivos com diferentes níveis de tensão, como demonstrado na Figura 24. Por ser possível implementar este protocolo com diferentes níveis de tensão, estabelece-se que as tensões de referência para determinar o estado lógico da linha, *low* ou *high*, são, respetivamente, $V_{IL} = 0{,}3\ V_{DD}$ e $V_{IH} = 0{,}7\ V_{DD}$ [42].

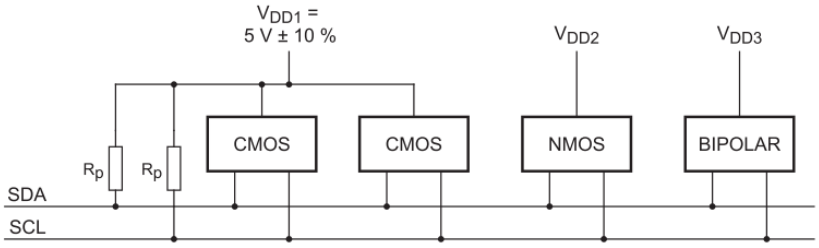


Figura 24 Dispositivos com diferentes níveis de tensão a partilhar o mesmo barramento I^2C [42]

A leitura dos *bits*, na linha de dados (SDA), deve ser feita no período em que a linha de *clock* (SCL) se encontra *high*. Em contrapartida, o estado *low* da linha de *clock*, representa o momento em que deve ser feita, caso necessária, a troca do estado lógico da linha de dados, tal como representado na Figura 25.

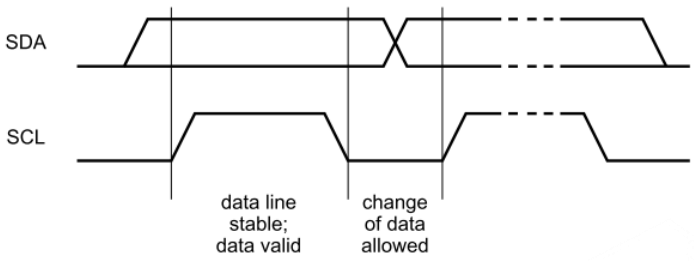


Figura 25 Leitura de *bits* no barramento I^2C [42]

Normalmente, assim que um dispositivo inicia uma transmissão, os restantes limitam-se a operar como recetor, mas, eventualmente, dois dispositivos poderão iniciar as suas transmissões ao mesmo tempo. O sistema de coletor aberto permite que um dos dispositivos perceba que a linha está ocupada e pare de transmitir, quando ambos transmitem valores lógicos diferentes. Neste caso, o dispositivo a transmitir valor lógico "1", não conseguirá elevar a tensão na linha pois o valor lógico "0" prevalece, logo este deteta que existe outro dispositivo a transmitir na linha.

Para determinar o início e o fim de cada transmissão, impõe-se o envio dos chamados s*tart* e *stop bits*. Como descrito anteriormente, as transições do estado lógico da linha, no decorrer de uma transmissão, ocorrem quando a linha de *clock* se encontra *low*. Neste caso, para diferenciar os *bits* de dados dos *start* e *stop bits*, estes são enviados enquanto a linha de *clock* se encontra *high*. Assim, na linha de dados, uma transição de estado *high* para *low* representa o início de uma transmissão e uma transição de *low* para *high* representa a condição de paragem da transmissão.

Como se percebe, o I^2C não utiliza endereçamento através de *hardware*, pois não conta com nenhum sistema de linhas de seleção de *peripherals*. Em contrapartida, neste protocolo o endereçamento é feito através de um mecanismo de *software*, e cada dispositivo instalado no barramento I^2C deverá ter um endereço de 7 *bits* associado. Existe a opção de usar endereços de 10 *bits*, mas esta opção só se justifica caso a gama de endereços para 7 *bits* não seja suficiente para o número de dispositivos instalados. No entanto, é possível utilizar dispositivos com endereços de 7 e 10 *bits* no mesmo barramento. Se no barramento só existir um dispositivo capaz de tomar a posição de *controller*, este não necessita de ter um endereço associado, pois será o único a iniciar transmissões.

Desta forma, depois de iniciada a transmissão, a primeira informação a ser enviada para a linha de dados é o endereço do dispositivo com o qual se pretende comunicar, seguido do *bit* R/W, que indica se é uma operação de leitura (R – *bit* 1) ou escrita (W – *bit* 0). Imediatamente a seguir, é enviado para o barramento um *bit* de *acknowledge*, utilizado sempre que é enviado um *byte* (8 *bits*) através do canal de dados. Este *bit* é enviado pelo dispositivo recetor e indica o sucesso da transmissão, informando o transmissor se pode enviar o próximo *byte* de dados. No caso de sucesso, o *bit* irá tomar o valor lógico "0", *acknowledge* (ACK), e poder-se-á dar continuação à transmissão. Caso contrário, o *bit* irá tomar o valor lógico "1", *not acknowledge* (NACK), e o *controller* decidirá se interrompe a transmissão ou volta a reenviar o *byte* anterior.

Após o envio do primeiro *byte* e respetivo ACK, a transmissão pode tomar um de dois rumos distintos (Figura 26): a direção de transmissão permanece inalterada ou a direção de transmissão é alterada. No primeiro caso, o *controller* continua a transmissão enviando os próximos *bytes* de dados seguidos do ACK, enviado pelo *peripheral*. No segundo caso, o *peripheral* toma a posição de transmissor, passando a enviar os *bytes* de dados, e o *controller* a de recetor, passando a enviar os *bits* ACK. Existe ainda um terceiro caso, onde o *peripheral* toma a posição de transmissor e envia o *bit* ACK logo após o envio do *byte* de dados. Para continuar a transmissão, o *controller* deverá enviar novamente um *start bit*, seguido do endereço do *peripheral*.

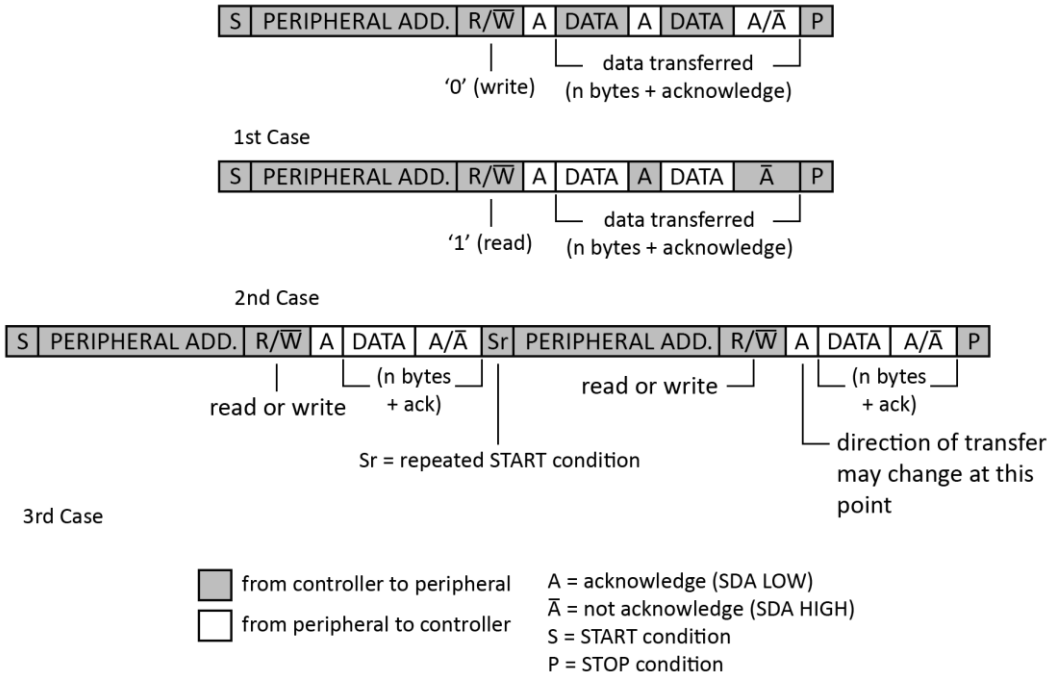


Figura 26 Modos de transmissão I^2C (adaptado de [42])

Existe uma exceção na construção da trama, que acontece quando o endereço utilizado contém 10 *bits*. Neste caso, o endereço necessita de ser dividido em 2 *bytes*. Os primeiros 5 *bits* do primeiro *byte* são sempre 11110, seguidos dos 2 *bits* mais significativos do endereço (10 *bits*) do *peripheral* e finalizando com o *bit* R/W. Neste momento, os *peripherals* aos quais o endereço corresponda aos primeiros *bits* do endereço enviado irão responder com um ACK. De seguida, o *controller* irá enviar o segundo *byte* do endereço, contendo os últimos 8 *bits*, e neste instante, o *peripheral* ao qual corresponda o endereço irá enviar o segundo ACK. A partir deste momento, o *peripheral* já está selecionado e a restante comunicação acontece de mesma forma que para os endereços de 7 *bits*.

3.3. SPI

O *Serial Peripheral Interface* (SPI) é, tal como o I^2C, um dos protocolos mais utilizados em comunicações a curta distância, em sistemas embebidos. Este é um protocolo série, bidirecional e síncrono desenvolvido pela *Motorola* em 1979. Ao contrário do I^2C, o SPI permite realizar comunicações *full-duplex*, devido ao maior número de linhas no seu barramento. Utiliza uma arquitetura de *controller-peripheral* com *single-controller*, responsável por controlar toda a transmissão, assim como a seleção dos *peripherals*.

3.3.1. BARRAMENTO

O barramento do SPI apresenta uma configuração diferente de outros protocolos, como por exemplo o I^2C, sendo este composto por 4 linhas de comunicação: 1 linha de clock (SCLK), 2 linhas de dados (COPI e CIPO) e uma linha de *chip select* (CS).

Por ser um protocolo síncrono, o SPI implementa uma linha dedicada de *clock* (SCLK) para sincronização entre *controller* e *peripheral*. O sinal é gerado pelo *controller*, durante todo o período das transmissões, com uma frequência que deve ser suportada por ambos os dispositivos. Esta frequência está diretamente relacionada com a velocidade de transmissão de dados, que não está limitada em termos de protocolo, embora, tipicamente, tenha valores a rondar as dezenas de MHz [49], [50].

As linhas de dados *Controller Output Peripheral Input* (COPI) e *Controller Input Peripheral Output* (CIPO) são a base para a comunicação *full-duplex* do protocolo SPI. Tal como os próprios nomes indicam, cada uma destas linhas é responsável por cada uma das direções de comunicação - *controller* para *peripheral* e *peripheral* para *controller* – e definem os *inputs* e *outputs* de cada dispositivo.

Ao contrário do protocolo I^2C que utiliza endereçamento através de *software*, no protocolo SPI o endereçamento é feito através de *hardware*, com a linha *chip select* (CS). Este método implica que exista uma linha dedicada para cada *peripheral* que esteja conectado ao barramento, podendo a conexão ser feita através de um *multiplexer*, controlado pelo *controller*, ou diretamente ligada ao *controller*. Embora não exista um limite de *peripherals* ligados ao barramento, com o aumento do número de dispositivos na linha, aumenta a complexidade do *hardware* necessário para manter o sistema de *chip select* funcional. Num momento em que não haja transmissão de dados, todas as linhas de SS

deverão ser mantidas em estado lógico "1" (*high*), mantendo todos os dispositivos desconectados do barramento. Desta forma, sempre que se realizar uma transmissão e for necessário escolher um *peripheral*, a linha CS, respetiva a esse dispositivo, deverá passar ao estado lógico "0" (*low*), durante o tempo da transmissão [50].

3.3.2. POLARIDADE E FASE DO *CLOCK*

O sinal de relógio, do protocolo SPI, pode ser modificado através dos parâmetros *Clock Polarity* e *Clock Phase*, associados aos *bits* CPOL e CPHA do registo SPICR1, respetivamente. A polaridade define o valor lógico da linha SCLK para os estados ativo e repouso. Com o *bit* CPOL = 1, a linha deverá ter estado *high* quando em repouso e *low* no momento do pulso do relógio. Com CPOL = 0, acontece o contrário, com a linha em estado *low* em repouso e *high* no momento do pulso. O *bit* CPHA define em que tempo do pulso de *clock* deve ser feita a leitura de *bits* das linhas COPI e CIPO. Quando CPHA = 1, a leitura do *bit* de dados ocorre nas mudanças de estado do *clock* pares. Se CPHA = 0, a leitura dos dados é feita nas mudanças de estado do *clock* ímpares [50]. Na Figura 27 são representados os quatro modos possíveis de configuração do pulso de relógio.

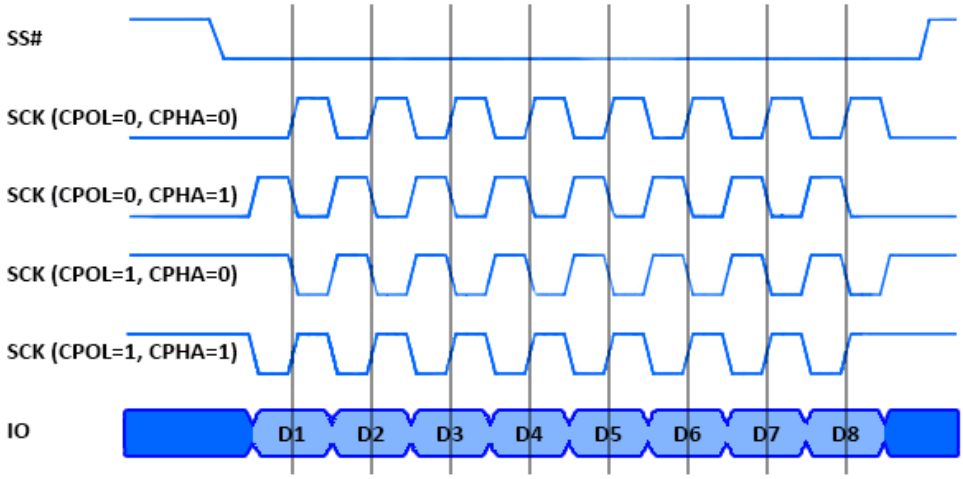


Figura 27 Polaridades e Fases do *clock* no protocolo SPI [51]

3.3.3. TRANSMISSÃO DE DADOS

No protocolo SPI, a transmissão de dados entre *controller e peripheral*, é feita através de 2 *shift registers* (*SPI Data Registers*), um em cada dispositivo. Juntamente com as ligações COPI e CIPO, forma-se uma ligação em anel entre os dois registos, como se verifica

na Figura 28. A conjugação destes 2 *shift registers* forma um registo distribuído entre ambos os dispositivos. A cada transmissão, os dados dos registos são trocados, ou seja, os *bits* do *controller* passam para o *peripheral* e vice-versa. O número de impulsos de *clock* necessários para completar uma transmissão é igual ao número de *bits* de cada registo, que, tipicamente, toma valores como 8 ou 16 [52].

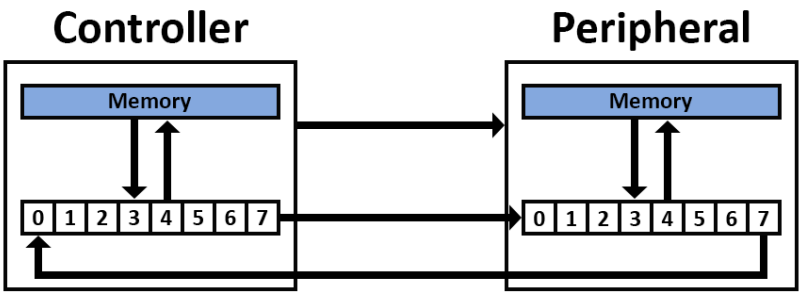


Figura 28 *Shift Registers* do protocolo SPI

A cada impulso de *clock*, o *controller* envia 1 *bit* para o *peripheral*, através da linha COPI, e vice-versa, através da linha CIPO, concebendo-se assim uma transmissão *full-duplex*. Apesar de em algumas transmissões, a informação útil ser unidirecional, continua a acontecer a troca integral dos dados dos registos, sendo que um deles é ignorado.

No primeiro *clock edge*, cada um dos dispositivos realiza o *shift out* do *bit* mais significativo do seu registo e coloca-o na respetiva linha de transmissão de dados. De seguida, no *clock edge* seguinte, dá-se a leitura dos *bits* das respetivas linhas – COPI para o *peripheral* e CIPO para o *controller* – que irão ocupar o *bit* menos significativo de cada registo. Dependendo dos dispositivos utilizados, poderá ser possível alterar os parâmetros de *Shift out*, passando estes a realizar o *Shift out* do *bit* menos significativo e a ler o *bit* para a posição de mais significativo. Este ciclo repete-se até que todos os dados sejam trocados entre os registos. No final, o impulso de *clock* é desligado e o sistema de *chip select* desselecciona o *peripheral* [50].

3.3.4. CONFIGURAÇÕES DE *PERIPHERALS*

O sistema *controller-peripherals* pode ter diferentes configurações, que afeta diretamente o modo como os dados são transmitidos entre dispositivos, e o modo como é realizado o *chip select*.

O caso mais comum é existir uma linha de *chip select* para cada *peripheral*. Neste caso todos os dispositivos partilham o mesmo barramento COPI e CIPO, como se pode verificar pelo exemplo da Figura 29 [53].

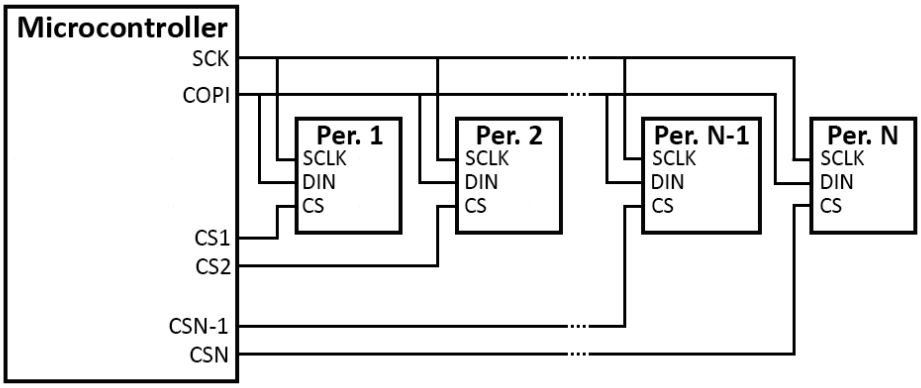


Figura 29 Configuração SPI com diferentes linhas de *chip select* (adaptado de [53])

Em alternativa ao à configuração base, existe a opção de implementar o sistema com uma arquitetura diferente, denominada *daisy chain*. Este caso reduz a complexidade da implementação de *hardware*, mas, por outro lado, dificulta a implementação de *software* do protocolo. Neste tipo de configuração, existe apenas uma linha de *chip select*, partilhada entre todos os dispositivos, e as linhas de COPI e CIPO são ligadas entre dispositivos, criando assim um anel entre os registos SPI e todos os dispositivos presentes no barramento, como se verifica na Figura 30. O que acontece com esta configuração, é que os *bits* de dados vão atravessando o anel virtual à medida que o impulso de *clock* está ativo [53].

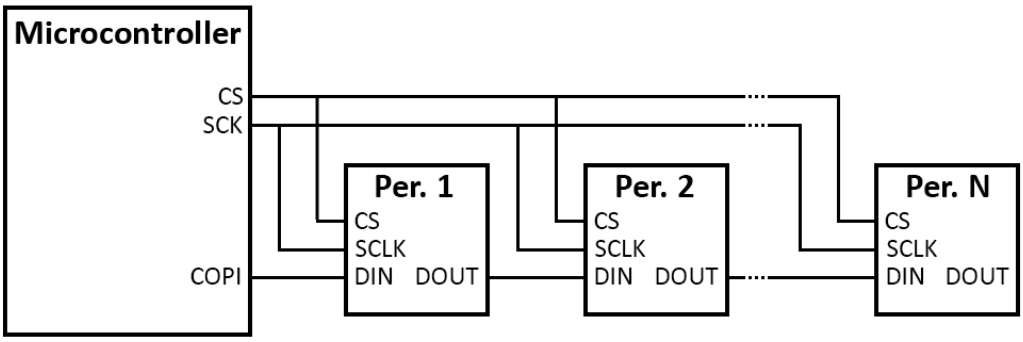


Figura 30 Configuração SPI em *daisy chain* [53]

Neste caso, quando se pretende enviar dados desde o *controller* até ao *peripheral* α, e os registos tenham β *bits*, a transmissão deve acontecer durante $\varkappa_1$ impulsos de *clock*, respeitando a equação 1:

$$\varkappa_1 = \alpha \times \beta \quad (1)$$

Para receber os dados de um qualquer *peripheral* α, sendo N o número máximo de dispositivos no barramento, a transmissão deverá acontecer durante $\varkappa_2$ impulsos de *clock*, conforme a equação 2:

$$\varkappa_2 = (2N - \alpha) \times \beta \quad (2)$$

3.4. USART

USART (*Universal Synchronous Asynchronous Receiver Transmitter*) é um dos protocolos mais utilizados para comunicação entre dispositivos, devido à sua versatilidade de funcionamento entre modo síncrono ou assíncrono e facilidade de implementação. A comunicação entre 2 dispositivos, um *controller* e um *peripheral*, é feita através de 2 linhas de comunicação, em modo assíncrono (UART), tal como ilustrado na Figura 31: TX, para escrita, e RX, para leitura. No caso de ser feita uma comunicação síncrona, é necessário utilizar mais uma linha de comunicação entre os 2 dispositivos – a linha de *clock* CLK – embora a implementação de comunicação assíncrona seja mais comum [54]. Sendo utilizadas duas linhas para transferências de dados, este protocolo é considerado *full-duplex*.

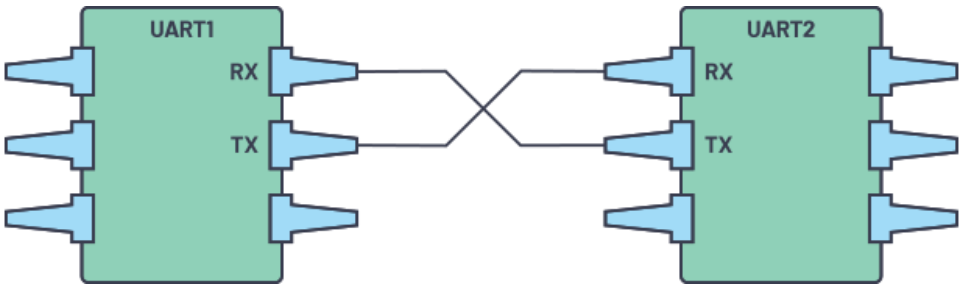


Figura 31 Interface de comunicação UART [54]

Não existindo um *clock* na comunicação assíncrona, é necessário definir um *baud rate* com o mesmo valor, para ambos os dispositivos, para que a comunicação seja realizada com sucesso. Este valor representa a velocidade com que os *bits* de dados são enviados do

dispositivo transmissor para o dispositivo recetor, e normalmente tomam valores entre os 9600 bps (*bits per second*) e 1500000 bps [54], [55].

Tal como nos restantes protocolos, nas comunicações por UART é definido um pacote de envio de dados constituído por vários conjuntos de *bits*: *start bit*, *data bits*, *parity bit* e *stop bits* (Figura 32). O valor de tensão das linhas, quando não existe transmissão (*idle*), é mantido num nível alto, correspondendo ao valor lógico 1 [54].

Figura 32 Pacote UART [56]

O *start bit*, enviado pelo transmissor, tem sempre um valor de tensão baixo, valor lógico 0, e dá início à comunicação, marcando a transição da linha do estado *idle* para transmissão. De seguida são enviados os *bits* de dados que podem variar de 5 a 9 *bits* de dados, normalmente do LSB (*Last Significant Bit*) para o MSB (*Most Significant Bit*). No caso de não serem utilizados 9 *bits* de dados, pode ser utilizado um *parity bit* na transmissão. Este *bit* é utilizado como um mecanismo para deteção de erros na transmissão, e toma os valores 1 ou 0 consoante o número de *bits* 1 enviados no conjunto de dados seja par ou ímpar, respetivamente. Por fim, podem ser utilizados 1 ou 2 *stop bits* que tomam sempre o valor lógico 1, ou seja, valor de tensão elevador, igual ao estado de *idle* [54], [57].

3.5. CAN

O protocolo CAN (*Controller Area Network*), também conhecido como CAN *bus*, criado em 1983 pela *Bosch*, permite implementar uma rede de comunicação entre microcontroladores e outros dispositivos conectados ao respetivo barramento. Um dos seus objetivos principais do seu desenvolvimento, seria criar um protocolo direcionado a veículos automóveis que tivesse um alto nível de segurança e que fosse eficiente face ao preço de implementação. Este desenvolvimento resultou num protocolo de comunicação à base de mensagens, capaz de atingir uma velocidade de transmissão de dados de 1 Mbit/s, algumas dezenas de metros e interligar até 120 dispositivos, sendo que qualquer um deles pode realizar o envio de mensagens [58].

3.5.1. BARRAMENTO E SINAL

Como meio físico, é aconselhada a utilização de par entrelaçado de modo a reduzir a sensibilidade a ruídos, das linhas. Em cada terminal do barramento, deve ser utilizada uma resistência de 120 ohm, eletricamente ligada entre as duas linhas, tal como é exemplificado na Figura 33 [58], [59].

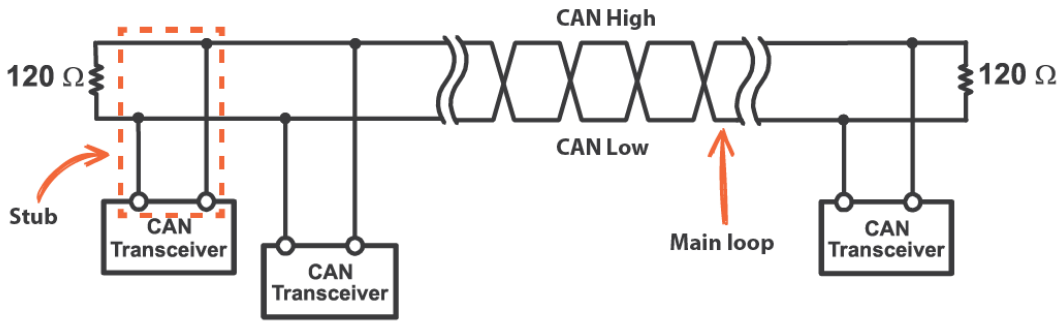


Figura 33 Barramento CAN [59]

As ligações entre dispositivos são feitas através de duas linhas, CANH (CAN *High*) e CANL (CAN *Low*), com níveis de tensão 1,5 V e 3,5 V, respetivamente, criando assim uma tensão diferencial (V_{DIFF}) de 2 V. Deste modo, estabelece-se que no envio de um *bit* recessivo (1) os valores de tensão de ambas as linhas passam a 2,5 V. No envio de um *bit* dominante (0), as tensões nas linhas mantêm os valores de referência, anteriormente indicados, tal como ilustrado na Figura 34 [58]–[60].

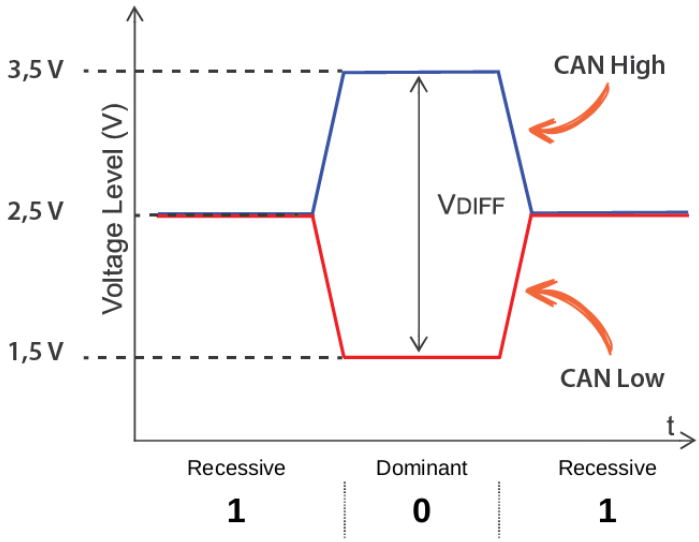


Figura 34 Níveis de tensão na Comunicação CAN [61]

Toda a informação transmitida no barramento CAN, é agrupada em mensagens ou pacotes de dados. A estes é sempre atribuído um nível de prioridade, de forma a evitar o acontecimento de erros e colisões na linha, durante o funcionamento do sistema. Outra das principais características deste protocolo é que as mensagens são sempre enviadas para todos os dispositivos conectados ao barramento, permitindo assim que com apenas um envio, uma mensagem seja enviada para mais do que um dispositivo. Depois de enviada e aceite, cada mensagem será filtrada pelo dispositivo, que irá recebe-la ou ignora-la consoante o seu ID, tal como é ilustrado na Figura 35 [58], [62].

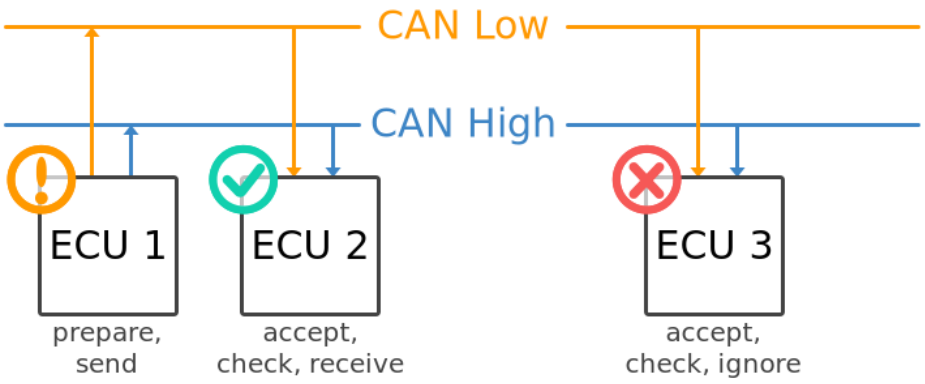


Figura 35 Funcionamento do envio e receção de mensagens CAN [62]

Ao longo dos anos, o protocolo CAN sofreu algumas alterações e melhoramentos, com novas versões a serem lançadas. A primeira versão do CAN, conhecida como CAN 1.0 foi oficialmente lançada em 1986. Alguns anos mais tarde, em 1991, a *Bosch* publicou a versão CAN 2.0 que introduziu a possibilidade de utilizar endereços de ID de 29-*bit* (*extended*), ao invés dos 11-*bit* (*standard*), sendo esta a versão mais atual do CAN base, utilizada atualmente. Desta forma, utiliza-se o nome CAN 2.0A para dispositivos com ID de 11-*bit* e CAN 2.0B para endereços de 29-*bit*. Além do CAN base, algumas outras variantes foram desenvolvidas, como são o caso do CAN FD e do CANopen [58], [61], [62].

3.5.2. ESTRUTURA DO PACOTE

O pacote CAN 2.0 está subdividido em diferentes conjuntos de *bits*, que podem ser diferentes consoante o tipo de pacote utilizado na transmissão. A primeira opção é utilizar o formato base CAN 2.0A, que utiliza um identificador de mensagem de 11 *bits*, representado na Figura 36, ou um formato estendido CAN 2.0B, que permite utilizar um identificador de

29 *bits*. As diferenças entre os dois formatos encontram-se nos campos *arbitration* e *control*. No caso de um pacote base, este pode ser dividido nos seguintes conjuntos de *bits* [58], [62]:

- SOF (*Start of Frame*) – 1 *bit* – tem como objetivo informar os outros dispositivos que irá ser iniciada uma comunicação e é sempre um 0 dominante;
- *Arbitration Field* – constituído pelos conjuntos de *bits* de identificação e RTR:
    - ID – 11 *bits* – identifica e define a prioridade da mensagem; quanto menor for o valor, maior é a prioridade;
    - RTR (*Remote Transmission Request*) – 1 *bit* – informa se o dispositivo envia dados (0 dominante) ou se pede dados (1 recessivo);
- *Control Field* – constituído pelos conjuntos de *bits* IDE, um *bit* de reserva e os *bits* com o tamanho dos dados:
    - IDE (*Identifier Extension Flag*) – 1 *bit* – define se o pacote está no formato base (0 dominante) ou estendido (1 recessivo). No formato base, este *bit* pertence ao *control field* e no formato estendido pertence ao *arbitration field*;
    - DLC (*Data Length Code*) – 4 *bits* – especifica o tamanho do campo de dados em *bytes*;
- Dados – 0 a 8 *bytes* – neste campo encontra-se a informação útil da mensagem (*payload*);
- CRC (*Cyclic Redundancy Check*) – 16 *bits* – este conjunto de *bits* é utilizado como um mecanismo de segurança, para confirmar a integridade do pacote de dados. O último *bit* deste conjunto é chamado de CRC *Delimiter* e é sempre um *bit* 1 recessivo;
- ACK (*Acknowledge*) – 2 *bits* – estes *bits* têm o nome de ACK *Slot* e ACK *Demiliter*. O primeiro é *bit* é uma resposta dos restantes dispositivos do barramento a informar que se receberam corretamente o campo CRC. Caso a

transmissão tenha sido feita com sucesso, os dispositivos deverão enviar um 0 dominante. O segundo *bit* é sempre 1 recessivo, e serve para limitar fazer e limitação do ACK *Slot*;

- EOF (*End of Frame*) – 7 *bits* – este conjunto de *bits* marca o final do pacote CAN.

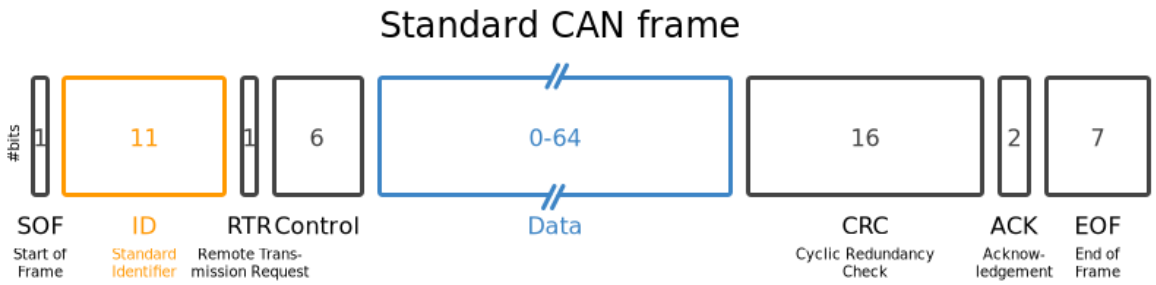


Figura 36 Campos do pacote CAN [62]

3.5.3. CANFD

O CANFD (CAN *Flexible Data-Rate*) é um protocolo que funciona como uma extensão do protocolo CAN *bus*. Este foi desenvolvido em 2011 e lançado oficialmente em 2012 pela *Bosch*, e, em relação ao protocolo base, acrescenta alguns melhoramentos importantes [63].

Os principais objetivos desta nova versão seriam aumentar a velocidade de transmissão, a quantidade máxima de dados por pacote e diminuir a probabilidade de erros ou falhas nos envios. Para conseguir atingir estes objetivos, dois desafios tiveram de ser ultrapassados. Em primeiro lugar, com o aumento do número de *bits* de dados, o tempo de ocupação do barramento seria maior, sendo a solução aumentar o *bit-rate* da transmissão. Em segundo lugar, e por consequência do aumento do *bit-rate*, a dificuldade seria perceber de que forma poderia ser aumentado o *bit-rate* de maneira a não causar colisões entre transmissões no barramento [64].

A solução encontrada para estes problemas passou por aumentar o *bit-rate* apenas na transmissão de dados e não na transmissão do início e do fim do pacote CAN, ilustrado na Figura 37. Se o *bit*-rate de toda a transmissão fosse aumentado, a distância máxima do

barramento iria diminuir drasticamente, pois seria necessário garantir que a transmissão de um *bit* acontece no mesmo intervalo de tempo em todos os nós do barramento [63]–[65].

As novas características introduzidas no CANFD, face ao CAN 2.0, levaram a uma alteração à estrutura do pacote, que permite a utilização de um maior volume de dados, até 64 *bytes*, e diferentes *bit-rates*. Na Figura 37 é possível observar 4 diferentes tipos de pacotes: CAN 2.0 com 3 *bytes* de dados (pacote A), CANFD com 3 *bytes* de dados (pacote B), CAN FD com 3 *bytes* de dados e 4 vezes o *bit-rate* base (pacote C) e CANFD com 64 *bytes* de dados e 10 vezes o *bit-rate* base (pacote D) [64].

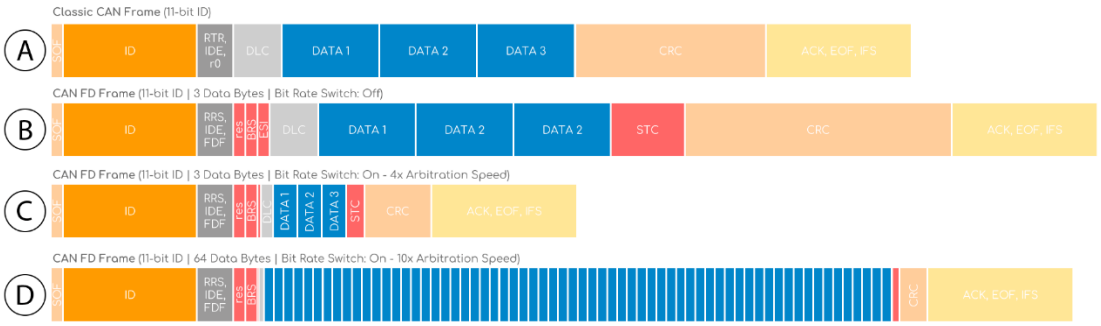


Figura 37 Comparação de pacote CAN e CANFD com diferentes velocidades de transmissão [64]

Tal como se constata pela Figura 37, o CANFD nem sempre é a melhor opção a adotar, face ao CAN 2.0, para enviar dados. Comparando o caso A com o caso B, percebe-se que, devido aos campos adicionais do pacote do CANFD (pacote B), este tem um maior período de transmissão, face à opção com CAN 2.0 (pacote A) e o mesmo volume de dados. Se for utilizado um *bit-rate* mais elevado, a situação altera-se, sendo possível, com um *bit-rate* 10 vezes superior (pacote D), enviar 64 *bytes* de dados mais rapidamente que 3 *bytes* com o *bit-rate* base.

Conclui-se então que o CAN FD apresenta uma vantagem em termos de velocidade de transmissão, quando é possível aumentar o *bit-rate* ou quando o volume de dados é superior a 8 bytes.

4. ANÁLISE DE REQUISITOS

O sistema proposto tem como objetivo principal estabelecer uma rede de monitorização e transmissão de dados entre um conjunto de módulos *NEXT-road*. Não sendo, o atual protótipo, a primeira versão destes módulos, foram criados, anteriormente, diferentes sistemas de monitorização e transmissão de dados com diferente *hardware* e *firmware*. Atendendo a esta informação, uma tarefa importante consistiu em estudar e analisar estas primeiras abordagens a este sistema de transmissão. Desta forma, foi possível, não só, perceber como foram cumpridos os requisitos nos anteriores sistemas, assim como estabelecer novos requisitos para o novo sistema a implementar.

Os requisitos propostos podem ser atribuídos aos diferentes módulos que constituem o sistema, no entanto, estes são principalmente definidos pelos dados que são necessários adquirir. Este conjunto é composto pelos dados relativos à passagem dos veículos, juntamente com os dados de monitorização do estado de cada um dos módulos, instalados na estrada. Desta forma, consideraram-se os seguintes itens:

- Velocidade e massa dos veículos
- Contagem de veículos
- Temperatura e humidade internas dos módulos
- Energia produzida

Além dos dados a adquirir, aquando do desenvolvimento do projeto, decidiu-se que o sistema deveria ser escalável e facilmente reparável, considerado de *plug and play*. Com escalável, pretende-se que o sistema funcione com um número variável de dispositivos, mas principalmente, que tenha um limite máximo de dispositivos elevado, permitindo, assim, criar instalações com um grande número de módulos *NEXT-road*. A facilidade de reparação, impõe que os problemas que surgirem durante o funcionamento, possam ser rapidamente detetados e o sistema possa ser facilmente reparado em caso de avaria ou problemas de *hardware*. Na prática, se algum sensor, microcontrolador ou componente eletrónico deixar de funcionar corretamente, o respetivo módulo eletrónico pode ser substituído rapidamente por um semelhante, sem que seja necessário reprogramar todo o sistema.

4.1. MÓDULO CONTROLADOR PERIFÉRICO

O Controlador Periférico é responsável pela adquisição de todos dados relativos à passagem dos veículos, com exceção da velocidade, pois é um dado obtido indiretamente. Assim sendo, este módulo tem de estar preparado para comunicar com diferentes tipos de sensores, através de distintos protocolos de comunicação, como os estudados e referidos no capítulo 3.

A frequência de aquisição de dados também é um fator importante a considerar. Como a geração de dados é diretamente proporcional à velocidade dos veículos, que passam pelos módulos, a frequência de funcionamento terá de ser superior, para que não aconteçam perdas de informação. Com o intuito de perceber a grandeza destas frequências, foi feito um pequeno estudo, em que se calculou, para diferentes velocidades e distância entre eixos dos veículos, a frequência mínima de aquisição de dados. Os cálculos efetuados ditam que para uma distância entre módulos fixa de 35 cm, uma velocidade de 80 km/h e um volume de dados de 10000 leituras por segundo, a frequência de aquisição de dados é aproximadamente 635 kHz. Apesar de os valores estipulados para a velocidade e volume de dados serem acima do que se espera na realidade, a frequência necessária está muito abaixo do que hoje em dia se consegue obter com certos sensores e microcontroladores. Assim sendo, a escolha dos componentes é facilitada.

Além da aquisição de dados, o módulo em questão terá de ser capaz de comunicar com o Controlador Secundário em conjuntos de, pelo menos, 10 para 1, ou seja, 10 Controladores Periféricos para 1 Secundário.

Em termos de *hardware*, este controlador está limitado ao espaço existente dentro dos módulos *NEXT*-road. Posto isto, devem ser construídos de maneira a acomodar-se tendo em conta a estrutura e construção do módulo, além de não afetar a sua parte mecânica.

4.2. MÓDULO CONTROLADOR SECUNDÁRIO

Os Controladores Secundários, responsáveis por fazer a ponte de comunicação entre os Periféricos e o Principal, estão também encarregues de tratar os dados enviados pelos Controladores Periféricos da sua rede. Dado o enorme volume de dados, estes necessitam de ser filtrados para que apenas informação útil, e não duplicada, seja enviada para o Controlador Principal. Além dos dados recebidos, o Controlador Secundário terá, também, de ser capaz de verificar falhas nos Periféricos e reportar essa informação ao Principal.

A gestão de envio de dados terá de ter em conta que, tal como no caso anterior, existirá um conjunto de, pelo menos, 10 Controladores Secundários ligados ao Controlador Principal, formando um conjunto de 10 para 1. Neste caso, como existirá um maior distanciamento entre os diferentes Controladores (no máximo, aproximadamente, 15 metros), o protocolo de comunicação escolhido para a transmissão de dados, terá de garantir que não ocorrerão falhas no envio e receção de mensagens.

Em termos de *hardware*, este controlador não está tão limitado quanto o Periférico, pois será posicionado numa caixa de ar entre os módulos *NEXT-road*, que por sua vez tem bastante espaço disponível.

4.3. MÓDULO CONTROLADOR PRINCIPAL

No caso do Controlador Principal, sendo o seu desenvolvimento partilhado com outros projetos, não existem requisitos iniciais além de este ter de ser capaz de comunicar com, pelo menos, 10 Controladores Secundários. A parte do *firmware* desenvolvido responsável pela comunicação com os níveis inferiores, deve ser feita de modo a ser facilmente integrável com os outros projetos desenvolvidos em paralelo.

5. ARQUITETURA DO SISTEMA

5.1. INTRODUÇÃO

Para implementação do sistema de monitorização e comunicação de dados, foi necessário, primeiramente, desenvolver uma arquitetura de sistema compatível com os protótipos *NEXT-road*. Deste modo, e tal como pensado inicialmente e descrito no subcapítulo de Objetivos, o sistema foi dividido em vários níveis, ao qual se deram os nomes de Controlador Periférico, Controlador Secundário e Controlador Principal. Cada um destes é colocado em um diferente local do sistema e é constituído por um microcontrolador e restantes periféricos. A comunicação entre eles é estabelecida da seguinte forma. Cada conjunto de 10 controladores periféricos comunica com 1 controlador secundário, através de um barramento denominado de Barramento Secundário. De modo que a informação gerada nos controladores periféricos chegue ao controlador primário, este último comunica com até 10 controladores secundários através de um segundo barramento, denominado de Barramento Primário, semelhante ao caso anterior. Após realizar o estudo acerca dos diferentes protocolos de comunicação (Capítulo 3), optou-se por implementar um sistema de comunicação baseado em CANFD em ambos os barramentos primário e secundário.

Inicialmente considerou-se que o sistema iria ser dividido em uma hierarquia de 4 níveis, no entanto, não se consideraram os Sensores como um nível distinto, pois estes são diretamente controlados pelo microcontrolador do Controlador Periférico correspondente. Desta forma, a arquitetura geral do sistema pode ser ilustrada pela Figura 38.

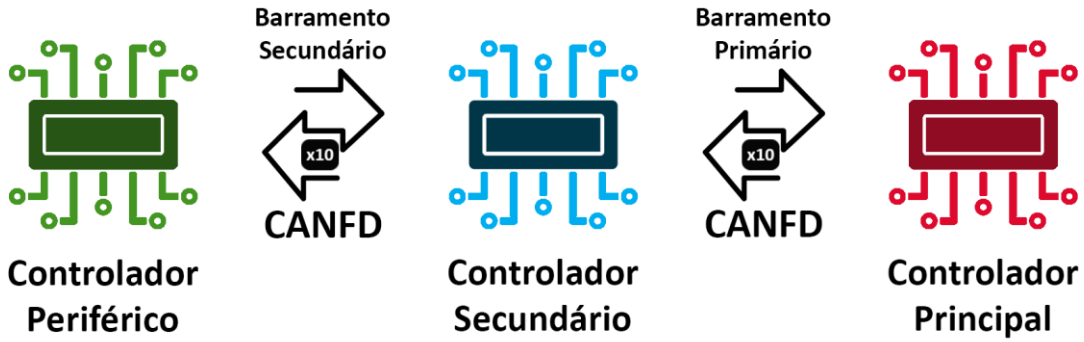


Figura 38 Arquitetura do Sistema

5.2. ARQUITETURA DO CONTROLADOR PERIFÉRICO

Tal como apresentado no Capítulo 4, os Controladores Periféricos são os dispositivos responsáveis por adquirir todos os dados externos ao sistema, através de diversos sensores, ligados e comandados pelo microcontrolador. Estes controladores vão ser colocados, juntamente com os devidos sensores, dentro de cada um dos módulos *NEXT-road*. Desta forma, os módulos podem ser fechados, existindo apenas 3 ligações para o exterior: entrada de energia para alimentar o circuito (microcontrolador e sensores), 2 fios para o barramento CANFD e saída de energia dos geradores. Todas as restantes ligações, entre microcontrolador e sensores, acontecem dentro do módulo. Considerando todos os componentes do Controlador Periférico, a sua arquitetura pode ser definida pela Figura 39.

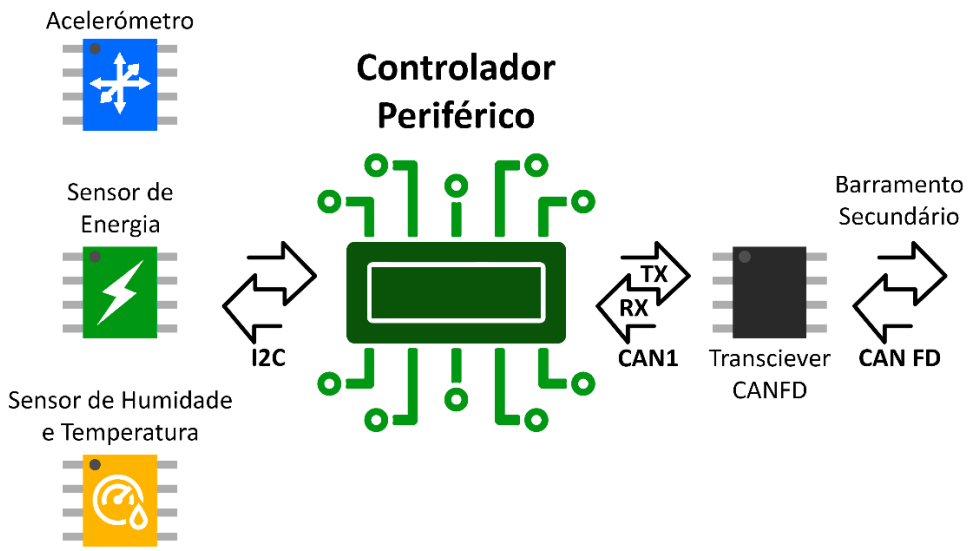


Figura 39 Arquitetura do Controlador Periférico

5.2.1. MICROCONTROLADOR

A escolha do microcontrolador para os controladores periféricos foi feita com base no estudo apresentado no Capítulo 2. Desta forma, adotou-se, para este projeto, os microcontroladores da família *STM32G4* da STM, mais propriamente, a placa de testes *NUCLEO-G431KB*, no caso do controlador periférico. As razões para a escolha deste componente baseiam-se nos seguintes pontos:

- Performance por Preço – o microcontrolador contém diversos tipos de periféricos, permitindo assim testar vários tipos de implementações; consegue atingir uma boa frequência de funcionamento, não se tornando um fator limitador para o funcionamento do sistema;
- Tamanho – tendo este o formato de um *Arduino Nano*, como é possível verificar na Figura 40, facilita a instalação do mesmo dentro dos módulos *NEXT-road*, por ser pequeno, além de, numa fase inicial, permitir realizar testes em *breadboard*;
- Protocolos de Comunicação – a capacidade de utilizar os protocolos CANFD e I^2C, foi um dos fatores principais na escolha do microcontrolador;
- Marca / IDE – a escolha da marca STM justifica-se por vários motivos: conhecimentos pré-adquiridos de *STM32Cube*; grande variedade de microcontroladores no mercado, com diferentes especificações; quantidade de informação disponível *online*.

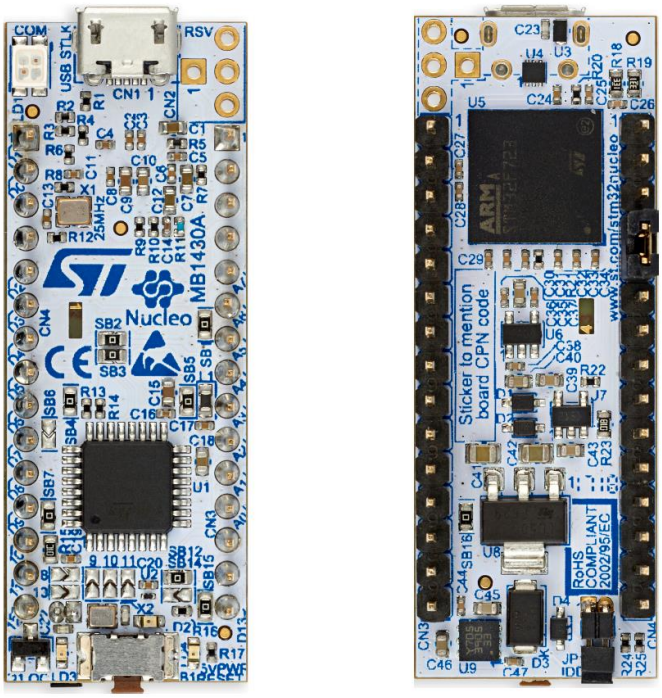


Figura 40 Placa de testes do Controlador Periférico, *NUCLEO-G431KB* [66]

5.2.2. PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação utilizados num sistema são uma peça chave para que a exista uma transmissão de dados eficaz e eficiente entre os diferentes componentes constituintes do sistema. Aquando da escolha do microcontrolador, foram estabelecidos os protocolos de comunicação que iriam ser utilizados no sistema a desenvolver. Foram, desta forma, escolhidos os protocolos CANFD, I^2C e SPI, sendo que este último nunca chegou a ser utilizado na prática.

O protocolo CANFD, tal como descrito no subcapítulo 3.5, apresenta diversas vantagens face aos restantes protocolos, como por exemplo, utilizar apenas dois fios para realizar a transferência de dados, permitir transmissões a grandes distâncias, permitir a criação de barramentos com um grande número de dispositivos conectados, filtros de mensagens, entre outros. Estas são algumas das razões pelo qual este protocolo foi escolhido para implementar o barramento de comunicação entre os controladores periféricos e o controlador secundário.

De forma a criar um barramento de comunicação com o protocolo CAN FD, é necessário utilizar um *transceiver* que transforme os sinais CAN TX (transmissão) e CAN RX (receção), provenientes do microcontrolador, nos sinais CAN *High* e CAN *Low*,

respetivamente, e vice-versa. Após uma busca por *transceivers* para CAN FD, optou-se pelo modelo *MCP2562FD* [67], fabricado pela *Microchip*. Este, apesar de ser alimentado a 5 V, e a tensão de funcionamento das linhas do microcontrolador ser 3.3 V, permite que a tensão dos pinos TX e RX sejam diferentes da tensão de entrada 5V, tendo para isso um pino que deve ser alimentado com uma tensão de referência, tal como se pode verificar pelo exemplo na Figura 41. A vantagem deste *transceiver* é que está disponível em formato DIP, o que facilita os testes em *breadboard*, mas também nos formatos SOIC (*Small Outline Integrated Circuit*) e DFN (*Dual-Flat No-Lead*), para montagem em PCB.

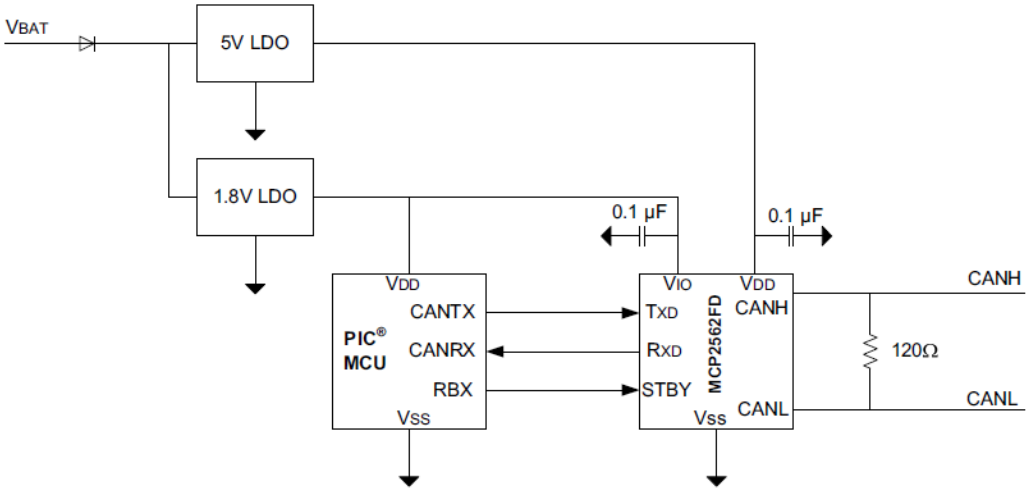


Figura 41 Esquema elétrico do *transceiver* CANFD *MCP2562FD* [67]

Para realizar a comunicação entre microcontrolador e sensores, estabeleceu-se que os protocolos I^2C e SPI deveriam estar disponíveis para utilização, pois estes são os mais comuns para este efeito. Depois de escolhidos todos os sensores, o protocolo SPI acabou por não ser necessário, tendo sido, em contrapartida, utilizado um barramento I^2C para interligar todos os sensores ao microcontrolador.

5.2.3. TENSÃO, CORRENTE E ENERGIA DOS GERADORES

Tal como estabelecido e referido no capítulo de Análise de Requisitos (Capítulo 4), a energia produzida pelos módulos *NEXT-road* é um dos dados que se pretende adquirir com a passagem dos veículos. Com a presença de dois geradores por módulo, a obtenção dos valores de energia pode ser feitas através dos valores de tensão e corrente de cada gerador. Assim, em cada módulo, tem-se duas tensões e duas correntes a analisar.

Com esta informação disponível, fez-se um pequeno estudo de mercado, para tentar perceber que tipo de soluções existiam, que permitissem obter, de uma forma rápida e prática, as energias geradas com os movimentos das tampas.

Umas das primeiras soluções encontradas foram os sensores *ACS725* [68] e *ACS723* [69], ambos fabricados pela *Allegro microsystems*. Estes são sensores de corrente, muito semelhantes, sendo umas das principais e únicas diferenças, a tensão de alimentação de 3.3 V e 5 V, que permitem obter valores de corrente desde os ± 5 V até aos ± 50 V, dependendo do modelo específico.

Ambos os sensores podem ser adquiridos em formato SOIC de 8 pinos ou através de *kits* de desenvolvimento, tal como se verifica na Figura 42.

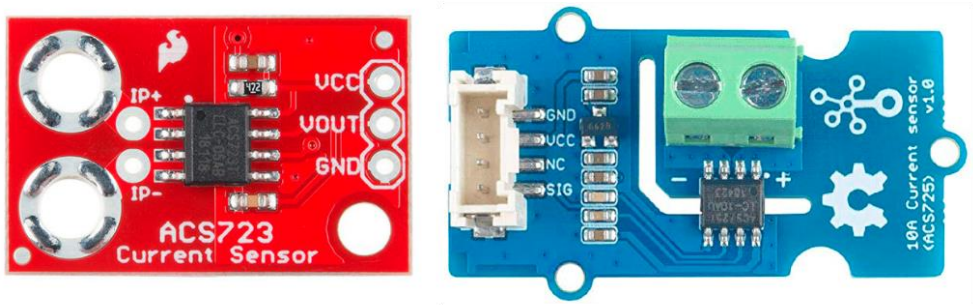


Figura 42 Sensores de corrente ACS723 e ACS725

Estes sensores não comunicam com o microcontrolador através de um processo digital, mas sim analogicamente. Um dos seus 8 pinos gera um sinal analógico, de 0 V à tensão de alimentação, que necessita de ser lido por um conversor analógico-digital (ADC), do microcontrolador. Assim, para o microcontrolador escolhido (*STM32G431KB*) o sensor ACS723 é descartado, pois a sua tensão de alimentação é de 5 V, e as entradas do ADC do microcontrolador só devem aceitar tensões de até 3.3 V.

Desta forma, para obter os valores de energia do módulo, utilizando o sensor ACS725, seriam necessários dois sensores para medir a corrente de cada um dos geradores. Além disso, seria necessário implementar dois circuitos com divisores de tensão, para que a tensão dos geradores não ultrapassasse a tensão máxima de 3.3 V do microcontrolador. No final, seriam necessárias 4 entradas do ADC, 2 para as correntes e 2 para as tensões. Os valores de energia seriam calculados indiretamente através da integração dos valores de potência ($tensão \times corrente$) pelo tempo.

Uma solução mais interessante, tirando partido de um sensor com *output* digital, seria utilizar um sensor da família *PAC193X* [70], fabricados pela *Microchip*. Estes são sensores de monitorização de potência e energia que comunicam através de I^2C, permitem tensões com um máximo de 32 V e podem conter de 1 a 4 *inputs*. Com uma tensão de alimentação de 2.7 V a 5.5 V, esta é uma solução ideal para este tipo de problema, pois permite calcular automaticamente os valores de potência e energia sem serem necessários recursos de processamento do microcontrolador. Em termos de monitorização de energia, este consegue integrar períodos desde 1 ms até ao máximo de 36 horas, dependendo da resolução das leituras efetuadas.

Este sensor pode ser adquirido em formato UQFN ou, no caso do modelo de 4 *inputs* (*PAC1934*), através de um *kit* de desenvolvimento (*MIKROE-2735*) disponibilizado pela empresa *MikroElektronika* [71], como se pode verificar pela Figura 43.

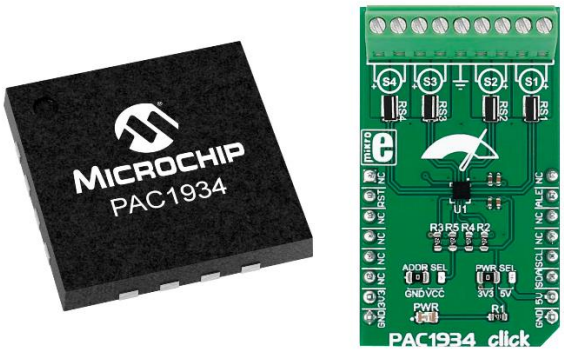


Figura 43 *Kit* de desenvolvimento do sensor *PAC1934* - *MIKROE-2735* [71]

O facto de só ser possível adquirir este sensor nestes dois formatos, dificulta a sua implementação neste sistema, pois, por um lado, no laboratório da *Pavnext*, não existem ferramentas para soldar um componente em formato UQFN, por outro, a placa de desenvolvimento *MIKROE-2735* tem um preço bastante elevado face ao valor normal do sensor.

Apesar de o preço do *kit* de desenvolvimento ser maior, a empresa adquiriu uma destas placas para utilizar em uma fase inicial de testes e verificar a sua *performance*.

5.2.4. TEMPERATURA E HUMIDADE INTERNAS

No que toca à escolha de um sensor para aquisição das temperaturas e humidades internas dos módulos, este processo foi facilitado pelo facto de a empresa já ter alguns *kits* de desenvolvimento do sensor *HTS221* [72], fabricado pela STM (Figura 44). Ainda assim,

foi feito um pequeno estudo de mercado para perceber que outros sensores estariam disponíveis e teriam alguma vantagem significativa a utilizar neste sistema, face ao *HTS221*.

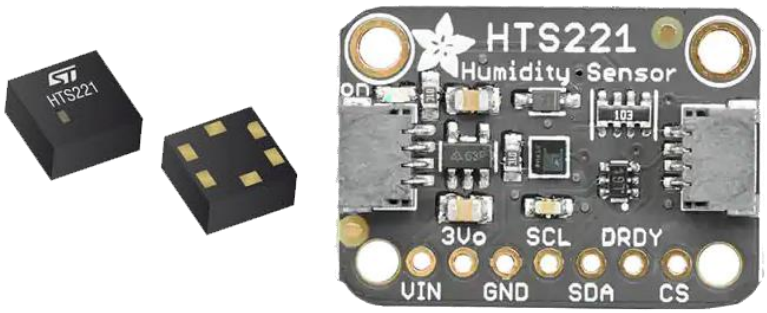


Figura 44 Sensor de Temperatura e Humidade *HTS221*

Alguns sensores como o *MCP9808* [73], da *Microchip*, ou o *BME280* [74], da *Bosch*, foram postos em causa, mas foram rapidamente descartados. O primeiro não foi considerado como uma opção, pois é muito semelhante em termos de características ao já adquirido *HTS221*. Já o *BME280*, além da leitura dos valores de temperatura e humidade, também tem a possibilidade de ler valores de pressão. Para este caso projeto em específico, a mudança de sensor também não se justificaria, pois, a pressão não é um dado necessário a adquirir.

Em suma, o sensor escolhido foi o *HTS221*, pois pode ser alimentado a 3.3 V, comunica com o microcontrolador através de I^2C e opera numa gama de temperaturas de -40 ºC a +120 ºC, suficiente para a aplicação em questão.

5.2.5. MOVIMENTO E POSIÇÃO DA TAMPA

De maneira a detetar o movimento da tampa do módulo *NEXT-road*, pensou-se em fixar um acelerómetro ao interior da tampa para, através das acelerações dos 3 eixos (X, Y e Z), saber a sua posição e movimentação. O eixo Z será o que mais importante dos 3, pois irá determinar a aceleração vertical da tampa, no entanto, os eixos X e Y são necessários pois ajudam a perceber se o movimento da tampa é feito exclusivamente na vertical e não com inclinações nos restantes eixos.

Tal como aconteceu com o sensor de temperatura e humidade, a *Pavnext* já tinha adquirido, anteriormente, alguns acelerómetros *ADXL345* [75], fabricados pela *Analog Devices* (Figura 45). Este acelerómetro consegue realizar leituras até ao máximo de ± 16 g, é alimentado a 3.3 V, comunica através de SPI ou I^2C e contém dois pinos capazes de gerar interrupções com diferentes tipos de movimentos. Desta forma, demonstrou ser um bom

sensor a utilizar neste projeto. Ainda assim, decidiu-se fazer uma pesquisa, com o intuito de encontrar alguma alternativa a este acelerómetro, tendo-se encontrado o acelerómetro *LIS3DH* [76], fabricado pela STM. Sendo este modelo praticamente idêntico em características, em relação ao *ADXL345*, não se optou pela sua aquisição, tendo o *ADXL345* sido escolhido como acelerómetro a utilizar no sistema.

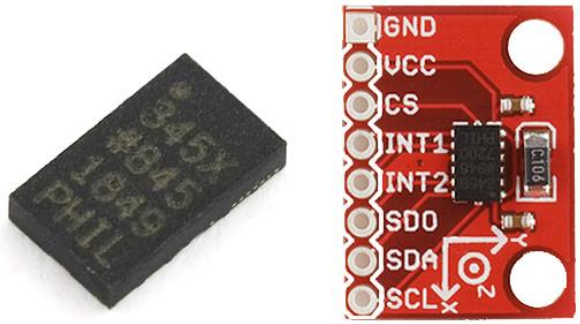


Figura 45 Acelerómetro ADXL345 e respetivo *kit* de desenvolvimento

5.3. ARQUITETURA DO CONTROLADOR SECUNDÁRIO

A arquitetura de sistema do Controlador Secundário é, em teoria, mais simples que a do controlador apresentado anteriormente. Esta é apenas composta pelo microcontrolador e dois *transceivers MCP2562FD*, utilizados para realizar a comunicação com ambos os barramentos de comunicação por CANFD, primário (CAN2) e secundário (CAN1). Com isto, o controlador secundário irá ter 3 ligações ao exterior, sendo elas: entrada de energia para alimentação do circuito, 2 fios para o barramento CANFD secundário e 2 fios para o barramento CANFD primário. Assim, a arquitetura do controlador secundário pode ser ilustrada através da Figura 46.

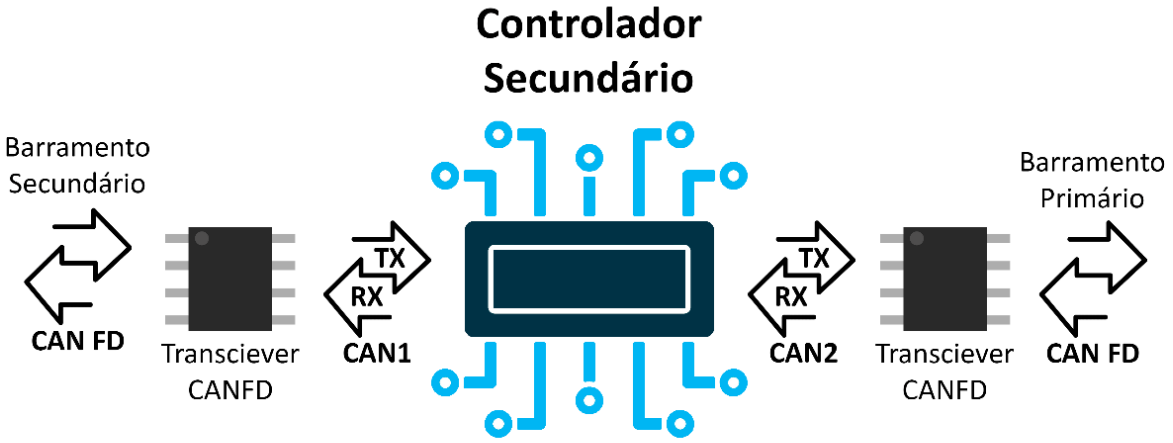


Figura 46 Arquitetura do Controlador Secundário

5.3.1. MICROCONTROLADOR

De maneira a facilitar o processo de escolha do microcontrolador e a programação do sistema, o modelo escolhido para este dispositivo pertence à mesma família do microcontrolador utilizado no controlador periférico. Como resultado, adotou-se a placa de testes *NUCLEO-G474RE* [77] (Figura 47), que, tal como o *NUCLEO-G431KB*, facilita o desenvolvimento de protótipos e de diferentes testes.

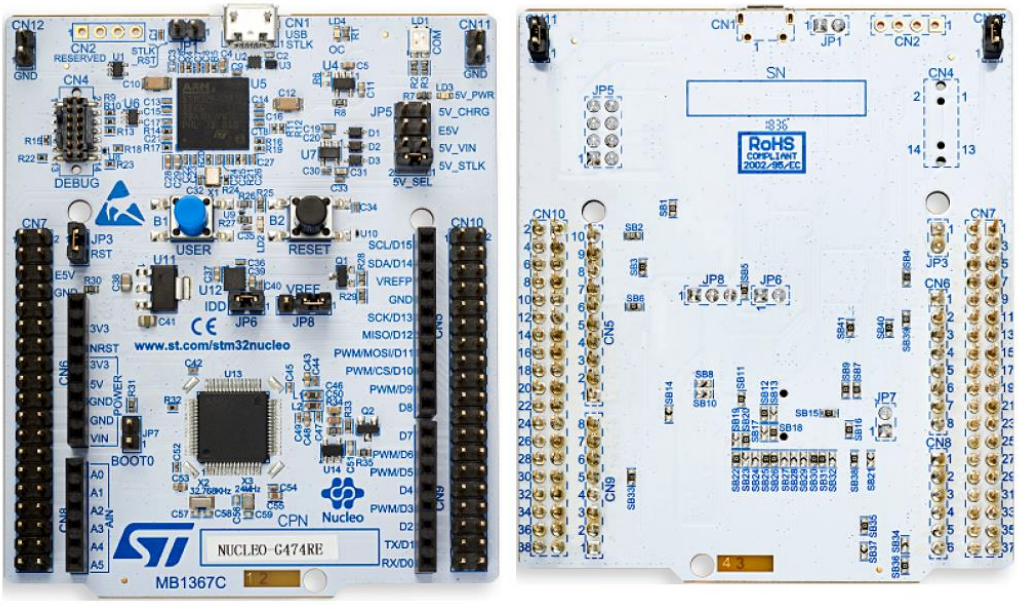


Figura 47 Placa de testes do Controlador Secundário, *NUCLEO-G474RE* [77]

Este microcontrolador, ao contrário do anterior, contém duas interfaces de comunicação CANFD, necessárias para realizar transferências de dados com ambos os barramentos, primário e secundário, em simultâneo.

Tal como referido anteriormente, a comunicação de dados deste controlador é totalmente feita através dos dois barramentos CANFD, não utilizando mais nenhum tipo de protocolo de comunicação.

5.4. ARQUITETURA DO CONTROLADOR PRINCIPAL

O desenvolvimento da arquitetura de sistema do controlador principal foi um processo mais complexo que os casos anteriores. O motivo desta dificuldade deve-se ao facto de o controlador principal ser partilhado entre vários outros projetos da empresa *Pavnext*. Após alguma discussão com os restantes colegas, optou-se por utilizar uma placa de testes *NUCLEO-G474RE*, igual à utilizada no controlador secundário, como microcontrolador. Desta forma, garante-se que o controlador principal consegue comunicar, utilizando do protocolo CANFD, com os controladores secundários através do barramento principal. Não só isso, mas também facilita o desenvolvimento do projeto, no que toca a *firmware*, por ser um microcontrolador da STM e utilizar o IDE *STM32Cube*. Além da *NUCLEO-G474RE*, e tal como os restantes controladores, este necessita de utilizar um *transceiver MCP2562FD* para a comunicação CANFD. Assim sendo, a parte da arquitetura do controlador principal, correspondente a este projeto, pode ser ilustrada pela Figura 48.

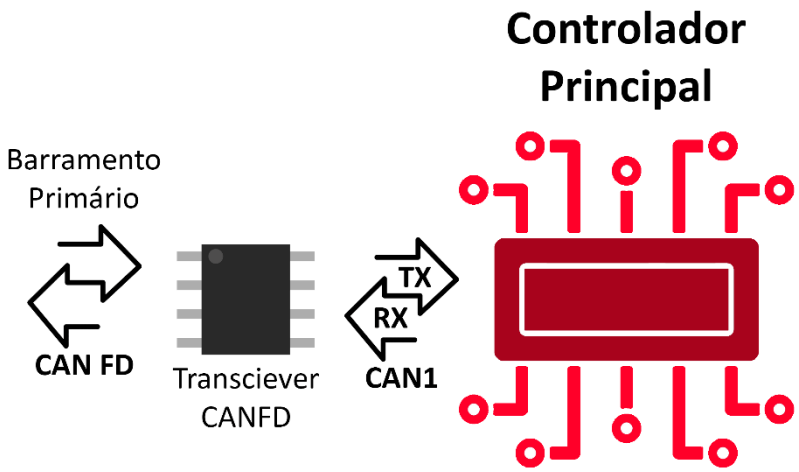


Figura 48 Arquitetura do Controlador Principal

5.5. COMUNICAÇÃO ENTRE CONTROLADORES

A comunicação entre os diferentes controladores foi feita utilizando, exclusivamente, os barramentos criados com base no protocolo CANFD, no entanto, esta é apenas uma das 3 propostas apresentadas para o problema.

Numa primeira proposta, pensou-se em um sistema de comunicação que utilizada o protocolo CAN juntamente com o protocolo *Ethernet*, para o barramento secundário e primário, respetivamente. Esta solução foi rapidamente descartada pois o custo associado à aquisição de microcontroladores com capacidade para o protocolo *Ethernet* era muito elevado, face às demais propostas.

As outras soluções apresentadas utilizariam o protocolo CAN como base para ambos os barramentos, sendo a única diferença, o facto de a segunda proposta utilizar o protocolo CAN base e a terceira proposta utilizar o protocolo CANFD. Não existindo uma diferença de custo significativa quanto ao *hardware* necessário, entre ambas as propostas, a decisão final passou por adotar o protocolo CANFD para os dois barramentos, por ser uma versão mais recente e com mais funcionalidades.

Após escolhido o protocolo, com o intuito de facilitar o processo de envio e receção de dados, e tirando partido da funcionalidade de identificação de mensagens do protocolo CANFD, foi desenhado um sistema de IDs para atribuir a cada um dos controladores e a cada tipo de mensagem enviada. Com este sistema, as mensagens podem ser automaticamente filtradas à entrada de cada um dos controladores, dependendo do ID com o qual estes forem programados.

Utilizando o formato de *extended* IDs, de até 29-*bits*, em hexadecimal, construiu-se um sistema de endereços, com o intuito de facilitar a atribuição dos mesmos a cada tipo de mensagem enviada. Este sistema, pode ser utilizado tanto no barramento secundário como no primário, com apenas algumas alterações. O resultado final é apresentado na Figura 49.

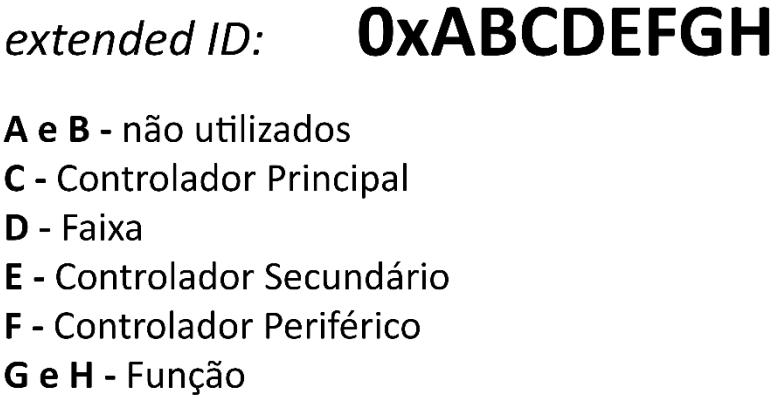


Figura 49 Esquema de IDs para barramentos CANFD

À exceção dos dois primeiros dígitos, é atribuído a cada um dos algarismos do endereço hexadecimal um significado. Para o barramento primário, começando pelo algarismo representado pela letra C, este indica se a mensagem é (valor "1") ou não é (valor "0") proveniente do controlador principal. Pela letra D, é representada a faixa da qual foi enviada a mensagem. Assim, em teoria, este tipo de sistema de mensagens permite que um Controlador Principal possa comunicar com um sistema com módulos instalados em até 15 faixas (1 a F). A letra E indica o número do controlador secundário do qual a mensagem foi enviada. Desta forma, juntamente com a letra D, é possível identificar qualquer controlador secundário do sistema (por exemplo, controlador secundário 3 da faixa 2). Em teoria, é possível criar um sistema com um máximo de 225 controladores secundários, 15 por faixa. Qualquer mensagem em que C = 1, implica que D = E = 0, e vice-versa. De seguida, a letra F indica de que controlador periférico é proveniente a informação enviada pelo controlador secundário. No caso de F = 0, significa que a informação enviada contém dados de vários controladores periféricos. Por fim, as letras G e H, juntamente, indicam qual o tipo de dados que são enviados. Os vários tipos de mensagens foram definidos no decorrer do desenvolvimento do *firmware* e serão referenciados no Capítulo 7.

Para utilizar este sistema no barramento secundário, é necessário cumprir algumas regras e alterações. Primeiramente, todas as letras acima do controlador secundário terão de ter o valor 0, pois este barramento não fornece acesso ao controlador principal e a diferentes controladores secundários. Desta forma, a letra E irá funcionar da mesma maneira que a letra C funciona no barramento primário. Se C = 0, a mensagem é enviada a partir de um controlador periférico. Caso C = 1, a mensagem é enviada a partir do controlador secundário.

Para facilitar a compreensão deste sistema de codificação de endereços, são de seguida apresentados alguns exemplos de possíveis IDs de mensagens CAN enviadas nos diferentes barramentos:

- Barramento Primário:
  - 0x00100002 ou 0x100002 – mensagem com função “02” enviada a partir do controlador principal;
  - 0x00013010 ou 0x13010 – mensagem com função “10” enviada a partir do terceiro controlador secundário da primeira faixa;
  - 0x00021000 ou 0x21000 – mensagem com função “00” enviada a partir do primeiro controlador secundário da segunda faixa;
- Barramento Secundário:
  - 0x00001001 ou 0x1001 – mensagem com função “01” enviada pelo controlador secundário, do barramento secundário correspondente;
  - 0x00000304 ou 0x304 – mensagem com função “04” enviada pelo terceiro controlador periférico, do barramento secundário correspondente;

6. IMPLEMENTAÇÃO DE HARDWARE

Após escolhidos e adquiridos todos os componentes principais de cada um dos controladores, iniciou-se o processo de implementação de *hardware*. Primeiramente, e juntamente com o desenvolvimento inicial do *firmware*, foram montados os primeiros protótipos dos vários controladores em *breadboard*. Desta forma, foi possível testar diferentes montagens, corrigir os erros que surgiram e desenvolver mais facilmente as várias componentes do controlador.

Ambos os controladores secundário e periférico passaram por diferentes versões e protótipos, desde a *breadboard* até à implementação do circuito em PCB. No caso do controlador principal, este foi utilizado em testes com o auxílio de uma *breadboard* e, só mais tarde, juntamente com os restantes colegas que trabalhavam com este controlador, é que foi desenvolvida uma PCB para o mesmo, agregando todo o *hardware* necessário para os diferentes projetos.

6.1. CONTROLADOR PERIFÉRICO

Tal como referido, o controlador periférico passou por diversos protótipos, tanto em *breadboard* como em PCB, até chegar ao produto final. Este foi um processo que evoluiu naturalmente ao longo do desenvolvimento do projeto, acompanhando o desenvolvimento do *firmware* do controlador.

6.1.1. CIRCUITOS DE TESTES EM *BREADBOARD*

O protótipo do controlador periférico em *breadboard* evoluiu ao longo de várias etapas e acompanhou os diferentes testes que foram feitos aos diferentes protocolos de comunicação e diferentes componentes testados.

O primeiro componente a ser testado foi o sensor de temperatura e humidade, o *HTS221*. Este foi colocado na *breadboard*, juntamente com o microcontrolador e foram feitas as seguintes ligações, descritas na Tabela 3.

Tabela 3 Ligações entre microcontrolador *NUCLEO-G431KB* e *HTS221*

Pino do Microcontrolador	**Pino do *HTS221***	**Descrição**
+5V	VIN	5 V
GND	GND	Ground
PA15	SCL	I^2C Clock
PB7	SDA	I^2C Data

Além das ligações entre microcontrolador e sensor, foi necessário ligar os pinos 3V e CS do *HTS221*. Desta forma o pino permanece sempre com o nível lógico "1", necessário para permitir a comunicação utilizando I^2C. Se o nível lógico deste pino for "0", o sensor passa a comunicar através de SPI, o que não é necessário.

Depois de validado o sensor de temperatura e humidade, testou-se, da mesma forma, o acelerómetro *ADXL345*. As ligações são idênticas às do *HTS221* pois, ambos os componentes, funcionam através de I^2C, com a exceção de o acelerómetro utilizar mais uma

linha, para gerar uma interrupção. Desta forma as ligações entre os dois componentes estão descritas na Tabela 4.

Tabela 4 Ligações entre microcontrolador *NUCLEO-G431KB* e *ADXL345*

Pino do Microcontrolador	**Pino do *ADXL345***	**Descrição**
+5V	VIN	5 V
GND	GND	Ground
PA15	SCL	I^2C Clock
PB7	SDA	I^2C Data
PB4	INT1	Interrupção

Para finalizar os testes individuais dos sensores, instalou-se o *PAC1934* na *breadboard* e fez-se as ligações como descrito na Tabela 5.

Tabela 5 Ligações entre microcontrolador *NUCLEO-G431KB* e *PAC1934*

Pino do Microcontrolador	**Pino do *PAC1934***	**Descrição**
+3.3V	3V3	3.3 V
GND	GND	Ground
PA15	SCL	I^2C Clock
PB7	SDA	I^2C Data

Além das conexões de alimentação e transferência de dados ao microcontrolador, de forma a simular diferentes valores de tensão e corrente na entrada S1 do *PAC1934*, utilizou-se uma fonte de alimentação variável, juntamente com uma resistência, seguindo o esquema elétrico da Figura 50. Com este circuito, e variando o valor de tensão da fonte de alimentação, foi possível programar e calibrar o sensor para ler os valores corretos.

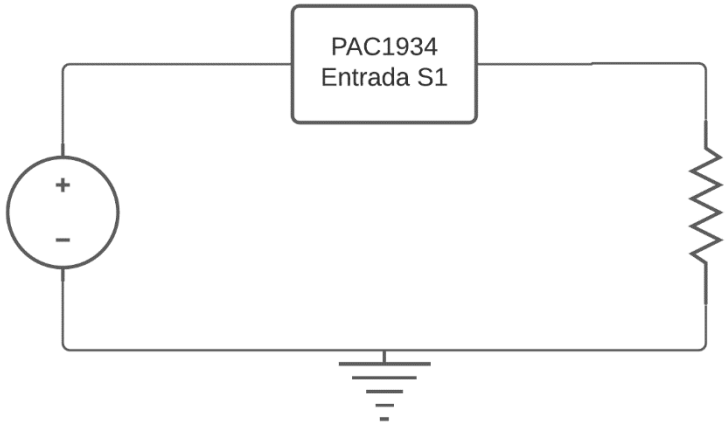


Figura 50 Esquema elétrico do circuito de testes do sensor *PAC1934*

Após testados com sucesso todos os sensores, iniciou-se o processo de testes das comunicações entre microcontroladores, utilizando o protocolo CANFD. Os primeiros testes foram realizados utilizando três *NUCLEO-G431KB* e um *NUCLEO-G474RE* juntamente com quatro *MCP2562FD* e quatro resistências de 120 ohm, criando assim o primeiro protótipo do barramento secundário. As resistências foram ligadas em paralelo em cada um dos terminais de ligação ao barramento dos *transceivers*, tal como indica a teoria. Na prática, e dependendo do formato e tipo de montagem realizada, alguns dos terminais poderão não necessitar de resistência. O circuito pode ser visualizado através da Figura 51.

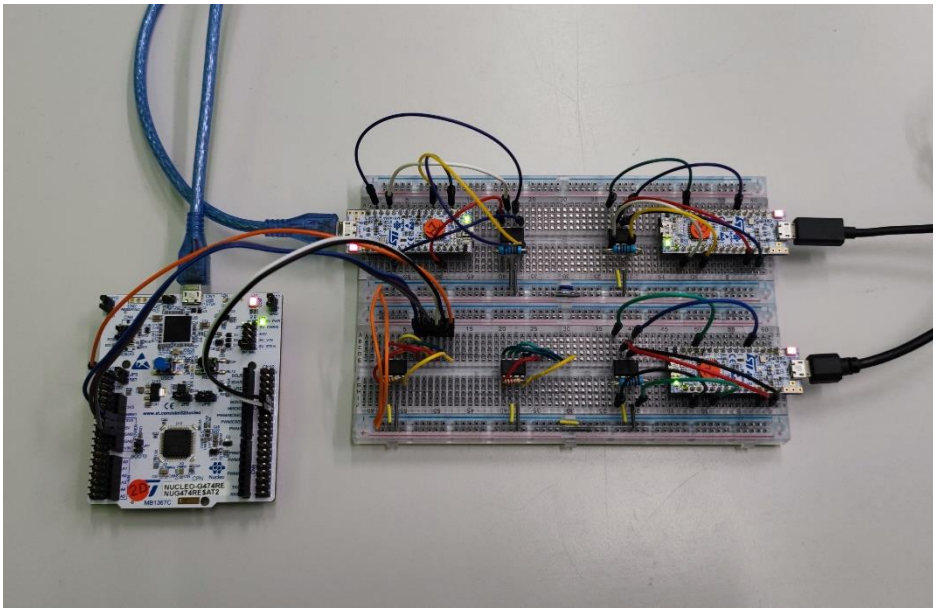


Figura 51 Fotografia do primeiro circuito de testes do barramento CANFD secundário

Além de todos os componentes descritos anteriormente, é visível na Figura 51, um *transceiver* ligado ao barramento, mas desconectado de qualquer microcontrolador. A colocação deste componente foi propositada, com o intuito de perceber se a existência de algum *transceiver* desligado no barramento, poderia afetar a comunicação dos microcontroladores.

Na montagem seguinte, excluiu-se o *NUCLEO-G474RE* e conectou-se um conjunto *ADXL345 + HTS221* a um dos *NUCLEO-G431KB*, de forma a gerar valores para enviar através do barramento. Estes dois sensores foram presos a um pequeno suporte, juntamente com a uma pequena PCB de ligações, desenvolvidos por outros colegas da empresa, de forma a facilitar a instalação dos sensores dentro do módulo *NEXT-road*. A montagem pode ser visualizada na fotografia da Figura 52.

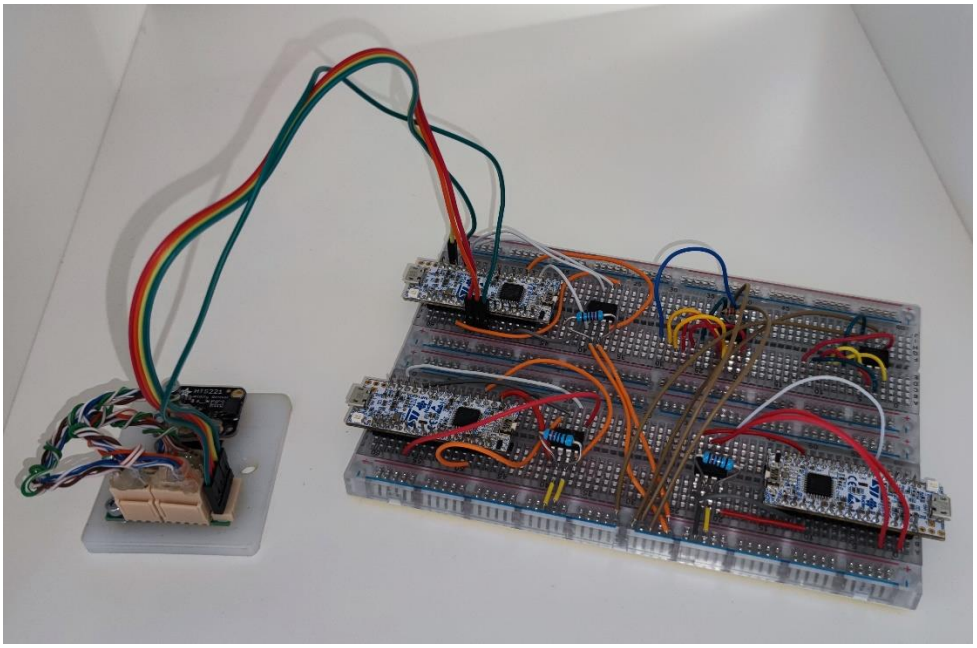


Figura 52 Fotografia do segundo circuito de testes do barramento CANFD secundário

O objetivo destas duas montagens foi aplicar, na prática, os conhecimentos adquiridos previamente, sobre o protocolo CANFD, em termos de *hardware* e *firmware*, e conseguir realizar com sucesso as primeiras comunicações e transferências de dados entre microcontroladores.

6.1.2. PCB V1

Depois de testados todos os componentes em *breadboard*, e o *firmware* de configuração de componentes e protocolos estar estável e funcional, iniciou-se o desenvolvimento da PCB do controlador periférico. Para tal utilizou-se o *software Altium Designer* [78]. Aquando do seu desenvolvimento, e devido a problemas com a falta de componentes eletrónicos, o sensor *PAC1934* teve de ser descartado, embora continue a ser a melhor opção para o sistema. A solução adotada para realizar a leitura dos valores de tensão e corrente dos vários módulos, foi utilizar uma PCB, pré-produzida pela *Pavnext*, que incluía dois sensores de corrente *ACHS-7193* [79] juntamente com dois divisores de tensão, um para cada gerador. Para realizar as leituras ligaram-se as saídas dos divisores de tensão e dos sensores de corrente a quatro terminais do ADC do microcontrolador. Desta forma, foi necessário alterar o esquema elétrico inicial pensado para a construção da PCB do controlador periférico.

O primeiro passo no processo de criação de uma PCB é o desenvolvimento do esquemático do circuito. Neste processo são colocados todos os componentes em uma folha e fazem-se as ligações entre todos os terminais. Nesta PCB, além do *transceiver MCP2562FD* e respetiva resistência de 120 ohm, os restantes componentes resumem-se a conectores para ligar tanto o *NUCLEO-G431KB* como os restantes sensores e a alimentação externa à PCB. O esquema elétrico desta placa encontra-se representado na Figura 53.

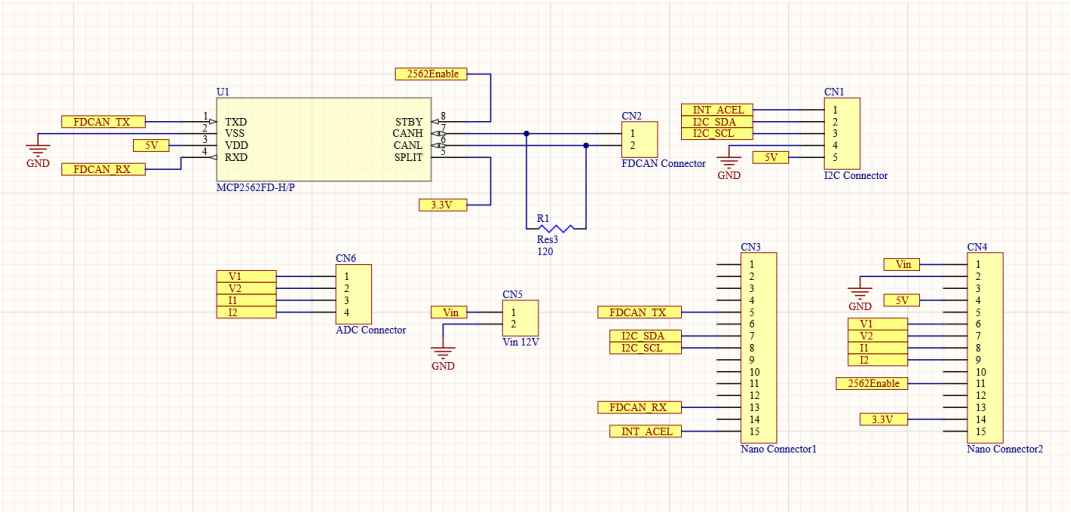


Figura 53 Esquema elétrico primeira versão da PCB do Controlador Periférico

Depois de finalizado o esquema elétrico, inicia-se a construção da PCB, através da colocação de todos os componentes na área da placa e conectando todos os pontos através de pistas e, se necessário, vias. No caso desta PCB, e devido à utilização da placa *NUCLEO-*

G431KB, foi necessário utilizar um guia de posição para colocar ambos os conectores do microcontrolador na PCB, à distância correta um do outro. O ficheiro utilizado foi fornecido pela própria STM, fabricante do microcontrolador. Os restantes conectores foram colocados em volta do espaço do microcontrolador. Para interligar os vários pontos de *ground*, foi utilizada a camada superior da PCB. Na Figura 54, é possível ver o esquema da PCB (esquerda) assim como uma simulação da placa após impressa (direita).

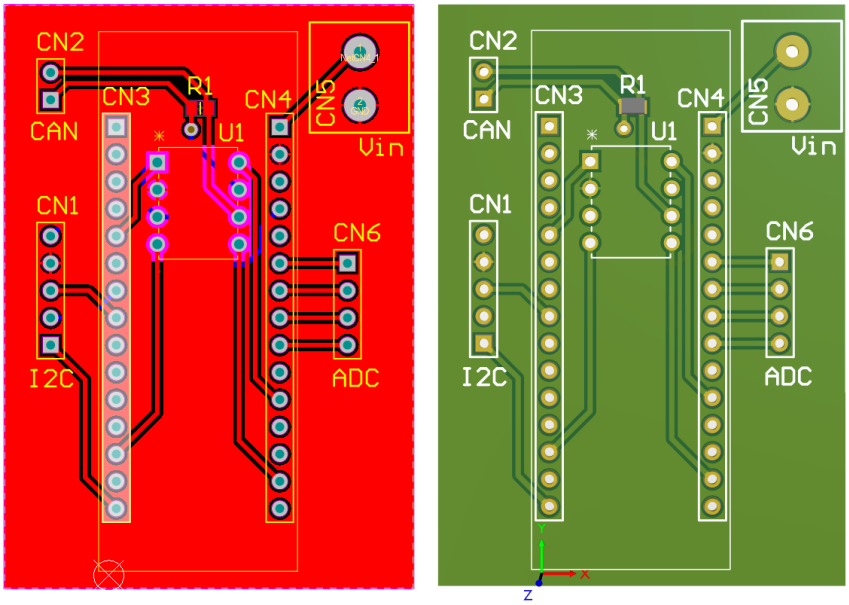


Figura 54 Primeira versão da PCB do Controlador Periférico

Após finalizado o projeto e produzida a PCB, foram soldados todos os componentes e conectores e a mesma foi testada. A Figura 55 apresenta duas fotografias da placa finalizada.

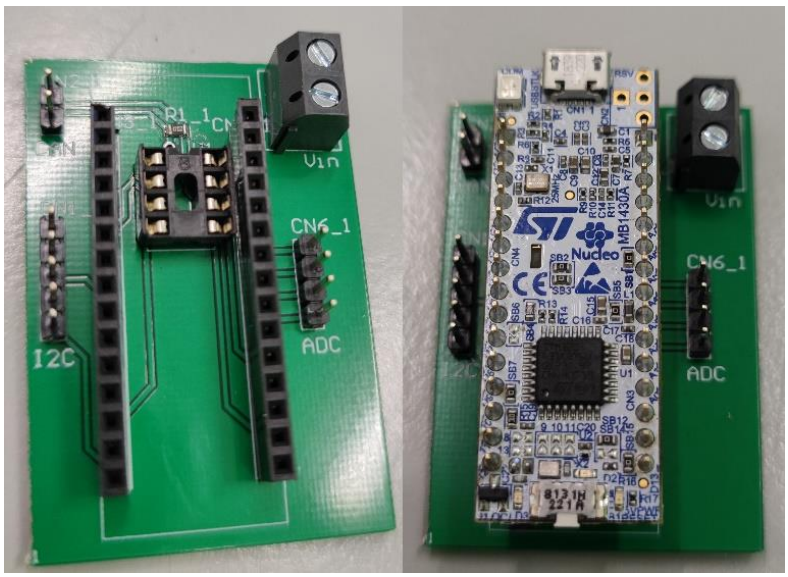


Figura 55 Fotografias da primeira versão da PCB do Controlador Periférico

6.1.3. PCB V2

Durante os testes da PCB, foram descobertos alguns erros que colocaram em causa o bom funcionamento do sistema. Dois dos quatro terminais do ADC escolhidos para realizar a leitura dos valores de energia dos geradores, estavam eletricamente conectados aos dois sinais utilizados pelo protocolo I^2C, SDA e SCL. A escolha incorreta destes dois pinos deveu-se a um pequeno *bug* do IDE *STM32Cube*, que não identificou como uma sobreposição a utilização do protocolo I^2C e das entradas 3 e 13 do ADC2.

Desta forma, a PCB do controlador foi atualizada, passando a ser utilizadas duas entradas do ADC1 e duas entradas do ADC2, tendo ainda sido acrescentado mais um conector para facilitar a alimentação dos sensores de corrente, como é possível visualizar na Figura 56.

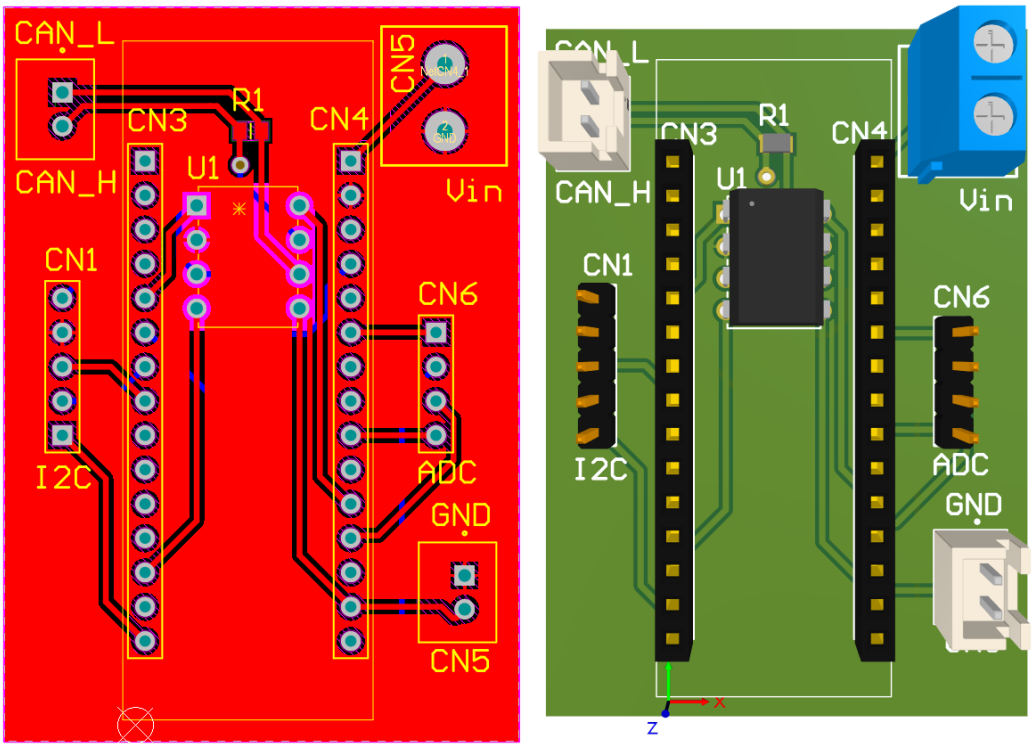


Figura 56 Segunda versão da PCB do Controlador Periférico

Devido a dificuldades relacionadas com escassez de tempo e o elevado preço de importação das novas PCB, as placas V1 foram fisicamente alteradas, de forma a acomodar as alterações pretendidas. Assim, foram interrompidas duas vias do circuito impresso e, utilizando alguns fios, foram estabelecidas as ligações corretas.

6.2. MÓDULO CONTROLADOR SECUNDÁRIO

No que toca ao controlador secundário, não foram desenvolvidos circuitos em *breadboard*, além dos já referidos subcapítulo 6.1.1, utilizados para testar as comunicações do barramento CANFD. Desta forma, após o desenvolvimento da PCB do controlador periférico, iniciou-se o desenvolvimento da PCB do controlador secundário.

6.2.1. PCB V1

Para a primeira versão da PCB do controlador secundário, optou-se por construir um *shield*, tirando partido dos conectores *ST morpho* da placa *NUCLEO-G474RE*. Desta forma, é possível aceder a todos os pinos e construir uma PCB que encaixe perfeitamente na placa de testes.

Novamente, utilizando o *Altium Designer*, iniciou-se o processo pelo desenvolvimento do esquema elétrico (Figura 57) da PCB. Neste caso, foram utilizados dois *transceivers MCP2562FD*, um para cada barramento, as respetivas resistências de 120 ohm e conectores, e, por fim, um conector para a alimentação de energia do circuito. Tal como nas PCBs desenvolvidas para o controlador periférico, foram utilizados *transceivers* no formato DIP por dois motivos: facilidade de troca em caso de avaria e, principalmente, falta de *stock* no formato SOIC.

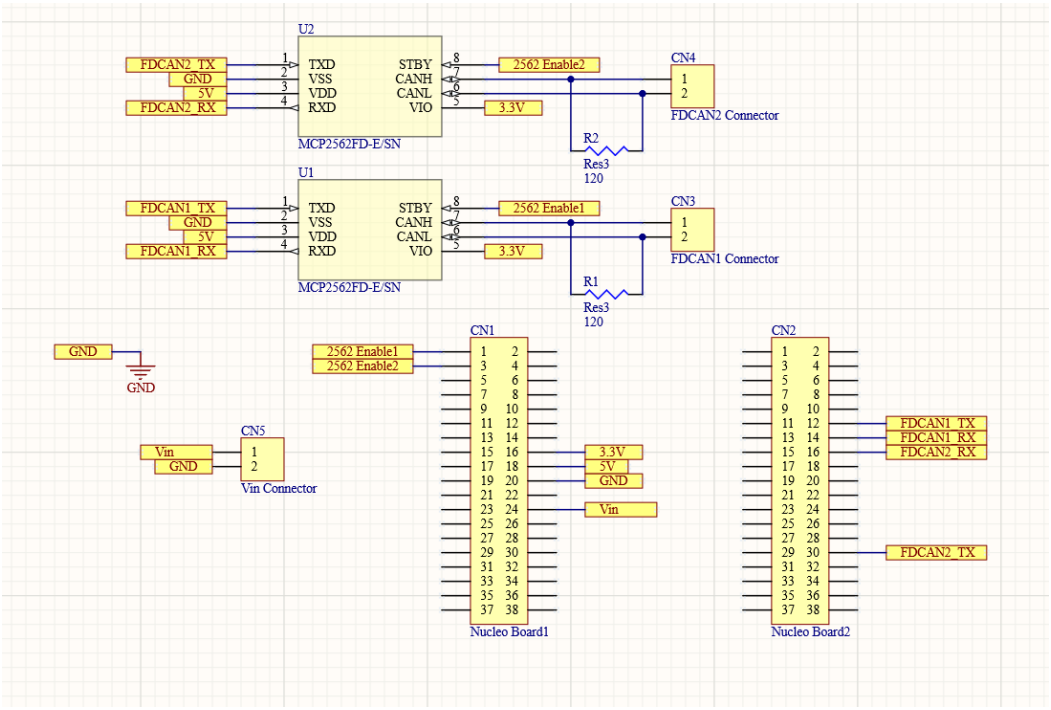


Figura 57 Esquema elétrico da primeira versão da PCB do Controlador Secundário

Da mesma forma que as restantes PCBs, o passo seguinte passa por dispor todos os componentes na folha de projeto. A colocação dos dois conectores relativos ao *ST morpho*, foi feita utilizando um guia da estrutura da placa *NUCLEO-G474RE*. O resultado final pode ser representado pela Figura 58, com exceção das pistas dos *transceivers*, que nesta versão, tinham sido desenvolvidos para o formato SOIC.

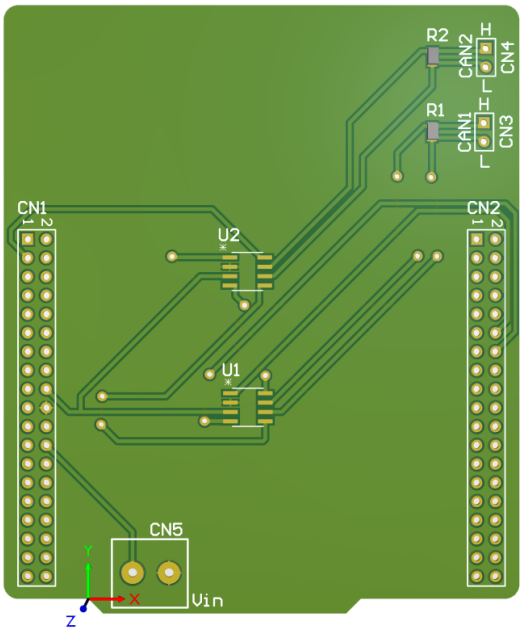


Figura 58 Primeira versão da PCB do Controlador Secundário

Depois de produzidas e soldados todos os componentes, as PCB foram testadas com sucesso. A Figura 59 apresenta uma fotografia da PCB finalizada conectada ao microcontrolador.

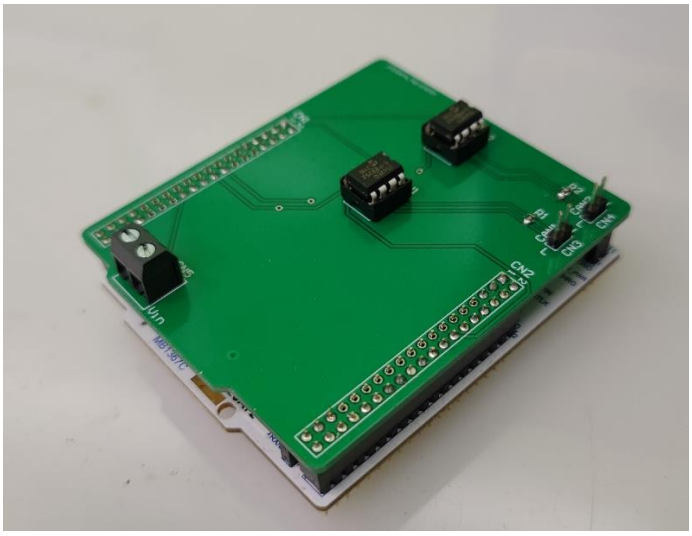


Figura 59 Fotografia da primeira versão da PCB do Controlador Secundário

6.2.2. PCB V2

Após realizar diversos testes e diferentes montagens com a primeira versão da PCB, percebeu-se que, para facilitar o processo de montagem do sistema, esta placa necessitaria de ser melhorada. Desta forma, utilizando o mesmo projeto, acrescentaram-se ao barramento secundário mais 9 conectores, permitindo assim conectar um máximo de 10 controladores periféricos ao controlador secundário, ficando este a funcionar como uma espécie de *hub*. Da mesma maneira que os conectores do barramento, acrescentaram-se 10 conectores para realizar a distribuição de energia para os 10 controladores periféricos. Sendo que na versão anterior, a PCB poderia ser alimentada com uma tensão entre 7 e 12 V, nesta segunda iteração da placa acrescentou-se um regulador de tensão de 8V da série L78 (L7808A) da STM [80]. Juntamente com o regulador, acrescentaram-se dois condensadores de apoio, de 0,33 µF e 0,1 µF, para a entrada e saída, respetivamente, como indicado pelo fabricante na Figura 60. Assim, a PCB pode ser alimentada até uma tensão de 35 V.

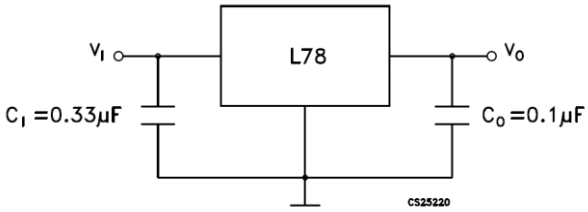


Figura 60 Circuito de implementação do regulador de tensão L78 [80]

Tal como se verifica na Figura 61, o esquema elétrico é semelhante ao da versão anterior da PCB, com a exceção do número de conectores e o circuito do regulador.

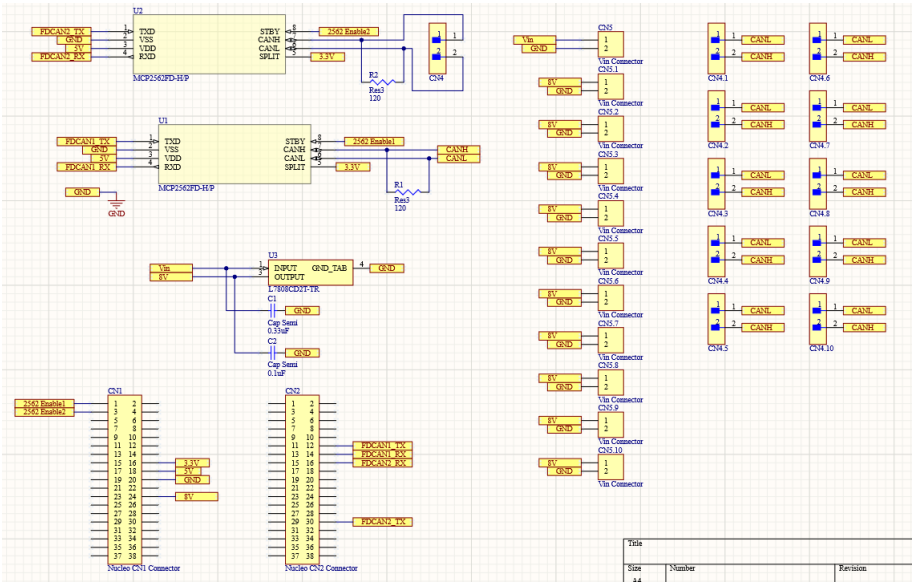


Figura 61 Esquema elétrico da segunda versão da PCB do Controlador Secundário

Nas figuras seguintes é possível visualizar a versão final da PCB do controlador secundário, virtualmente (Figura 62) e depois de soldados alguns dos conectores necessários (Figura 63).

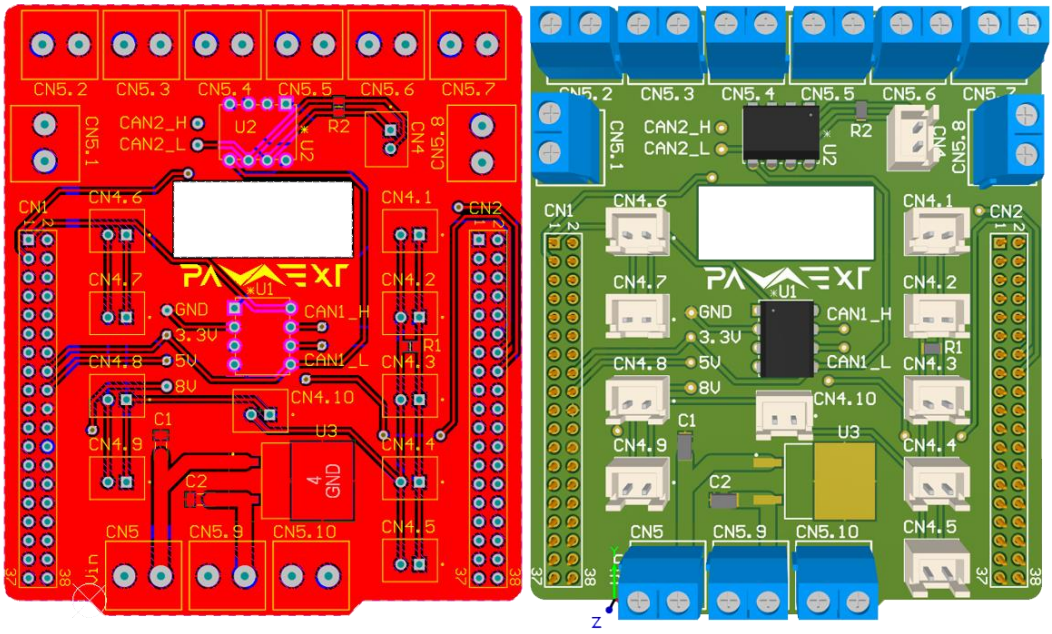


Figura 62 Segunda versão da PCB do Controlador Secundário

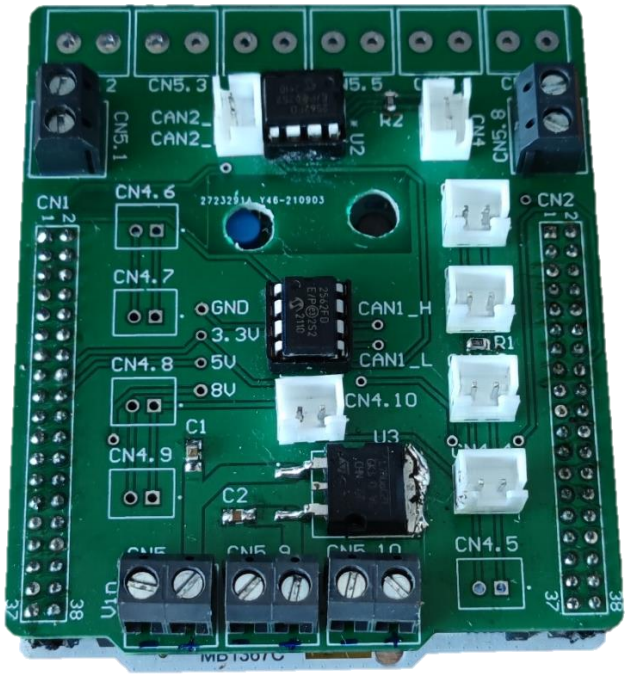


Figura 63 Fotografia da segunda versão da PCB do Controlador Secundário

6.3. MÓDULO CONTROLADOR PRINCIPAL

O Controlador Principal, tal como descrito anteriormente, foi desenvolvido em parceria com outros projetos a decorrer em paralelo na empresa, logo o desenvolvimento da PCB foi também partilhado entre os vários envolvidos. Em relação a este projeto, os únicos componentes necessários na PCB, além do microcontrolador, são o *transceiver MCP2562FD*, a respetiva resistência de 120 ohm e um conector para permitir ligar o controlador primário ao barramento CANFD, correspondente. Na Figura 64 é possível visualizar o resultado final do Controlador Principal, incluindo todos os *shields* desenvolvidos, e onde também é possível ver o conector do barramento CANFD, ao qual estão ligados os fios verde e branco.

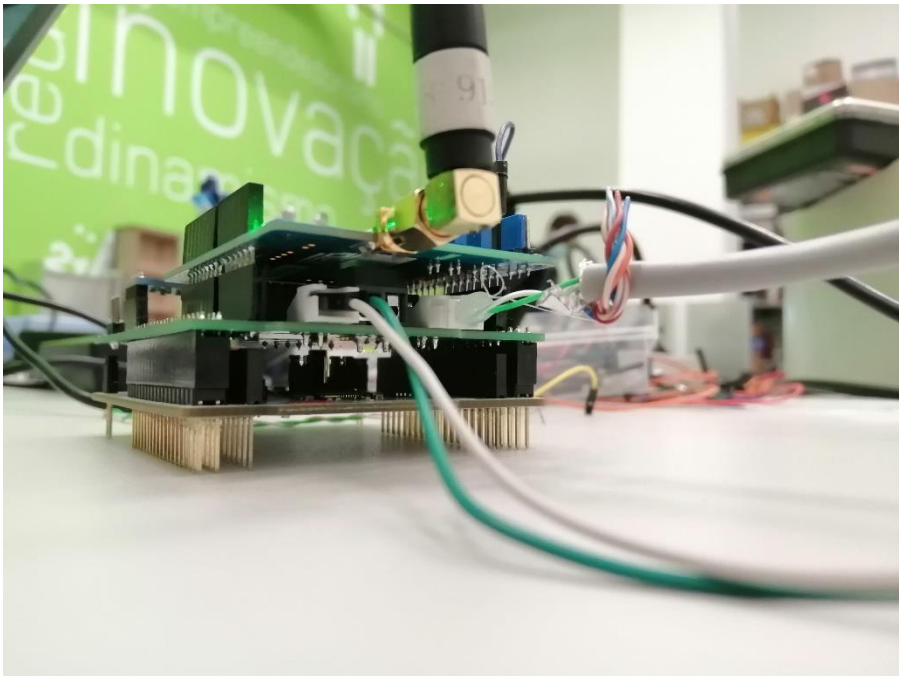


Figura 64 Fotografia da versão final do Controlador Principal

7. IMPLEMENTAÇÃO DE FIRMWARE

O processo de implementação de *firmware*, dos diferentes controladores, ocorreu em paralelo com a desenvolvimento e evolução dos respetivos protótipos de *hardware*. A ferramenta utilizada em todo o processo foi o *STM32Cube*, STM, referido anteriormente no subcapítulo 2.2. Desta forma, o processo de configuração dos diferentes periféricos das *NUCLEO-boards*, como ADC, *timers*, protocolos de comunicação, entre outros, foi simplificado pela *Device Configuration Tool*, presente no IDE.

7.1. CONFIGURAÇÃO DO PROTOCOLO CANFD

A configuração do protocolo CANFD é um dos casos da utilização desta ferramenta. Para que os barramentos funcionem corretamente, os parâmetros de comunicação deste protocolo têm de ser os mesmos em todos os microcontroladores utilizados. Desta forma, este periférico foi um dos primeiros a ser configurado.

Através do painel da *Device Configuration Tool*, começou-se por selecionar a aba *Connectivity* e ativar as interfaces CANFD de cada um dos microcontroladores. No caso dos controladores periféricos e principal, foi ativada a FDCAN1, e nos controladores

secundários, as interfaces FDCAN1 e FDCAN2. Depois de ativadas, foi necessário alterar um conjunto de parâmetros, apresentados na Figura 65.

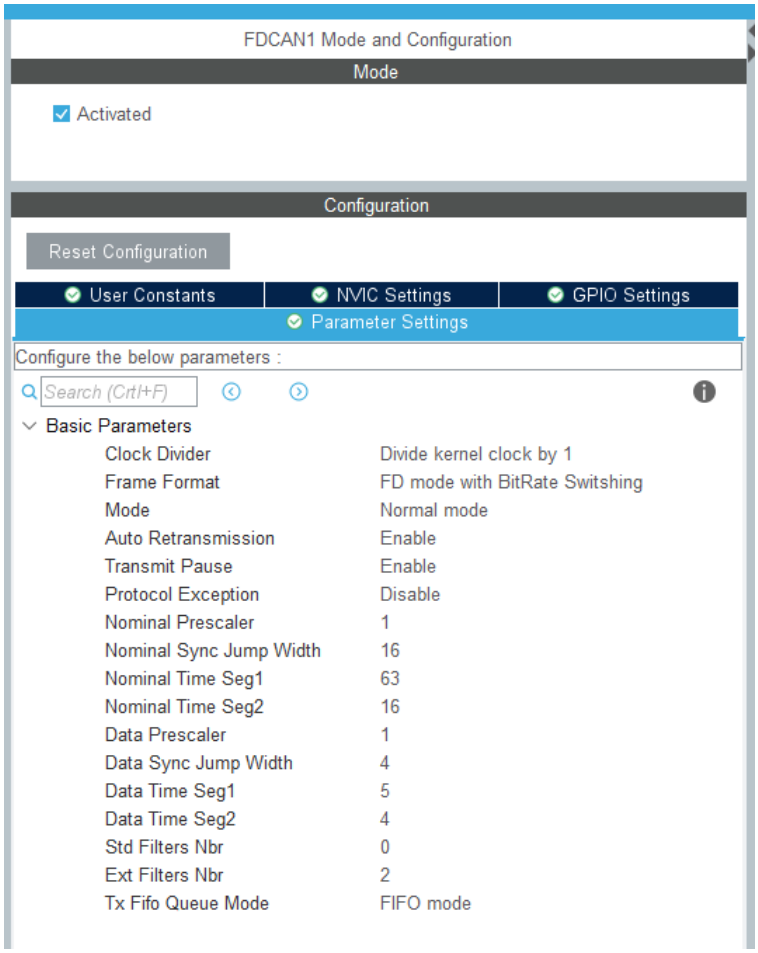


Figura 65 Parâmetros de configuração do protocolo CANFD

Os primeiros parâmetros configurados, são relativos à frequência de funcionamento do barramento. Para facilitar este processo de configuração, assegurou-se que todos os microcontroladores trabalhavam à mesma frequência base de 80 MHz. Assim, os parâmetros *Clock Divider* e *Nominal Prescaler* permanecem com os valores pré-definidos. O *Frame Format* define que tipo de mensagem CAN será utilizada, e tem três opções disponíveis, correspondentes aos três tipos de mensagens apresentados no subcapítulo 3.5.3:

- *Classic Mode*;
- *FD mode without BitRate Switching;*
- *FD mode with BitRate Switching.*

De seguida, o parâmetro *Mode* define o modo de transmissão utilizado pelo microcontrolador, utilizado maioritariamente para testar tanto configurações como o

funcionamento do barramento. Alguns destes modos, como por exemplo os *External Loop Back Mode* e *Bus Monitoring Mode*, representados na Figura 66, foram utilizados para testar o bom funcionamento dos primeiros protótipos do barramento CANFD. Estes permitem controlar a entrada e saída de mensagens, assim como reencaminhar as mensagens enviadas para o *buffer* de receção.

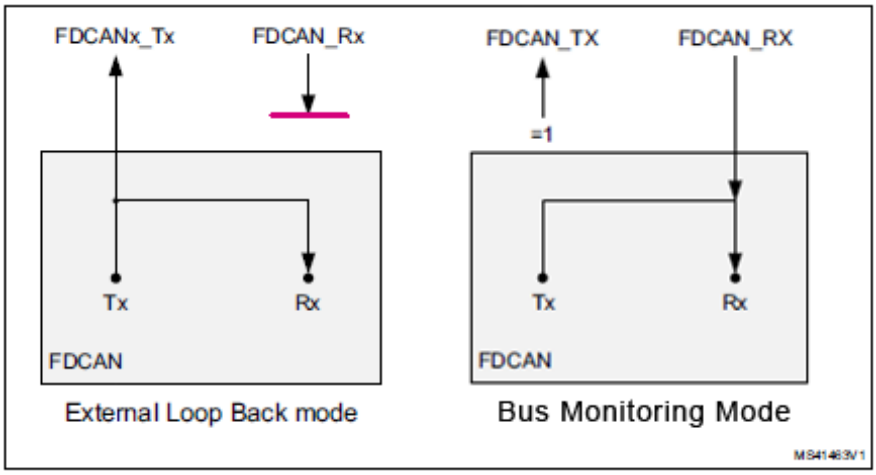


Figura 66 Modos de *Loop back* do protocolo CAN nas placas *NUCLEO* [81]

Os parâmetros *Std Filters Nbr* e *Ext Filters Nbr* definem o número de filtros utilizados por cada microcontrolador para IDs *standard* e *extended*, respetivamente. No caso apresentado na Figura 65, são utilizados 2 filtros para endereços *extended* e nenhum para *standard*.

De seguida, o *Tx FIFO Queue Mode*, tal como o nome indica, define se será utilizado o modo FIFO (*First in first out*), em que as mensagens são enviadas pela ordem de chegada ao *buffer*, ou o modo *Queue*, em que as mensagens são enviadas respeitando a prioridade de IDs. Para este sistema, como se pretende que as mensagens sejam enviadas logo após o envio para o *buffer*, escolheu-se a opção FIFO.

As restantes configurações, foram definidas utilizando como base um código de teste do protocolo CANFD, fornecido por um utilizador nos fóruns da comunidade da STM [82], calculadas através de uma ferramenta *online* [83]. Estes parâmetros definem em que momento de transmissão do *bit* deve ser efetuada a leitura, tal como pode ser visualizado na Figura 67.

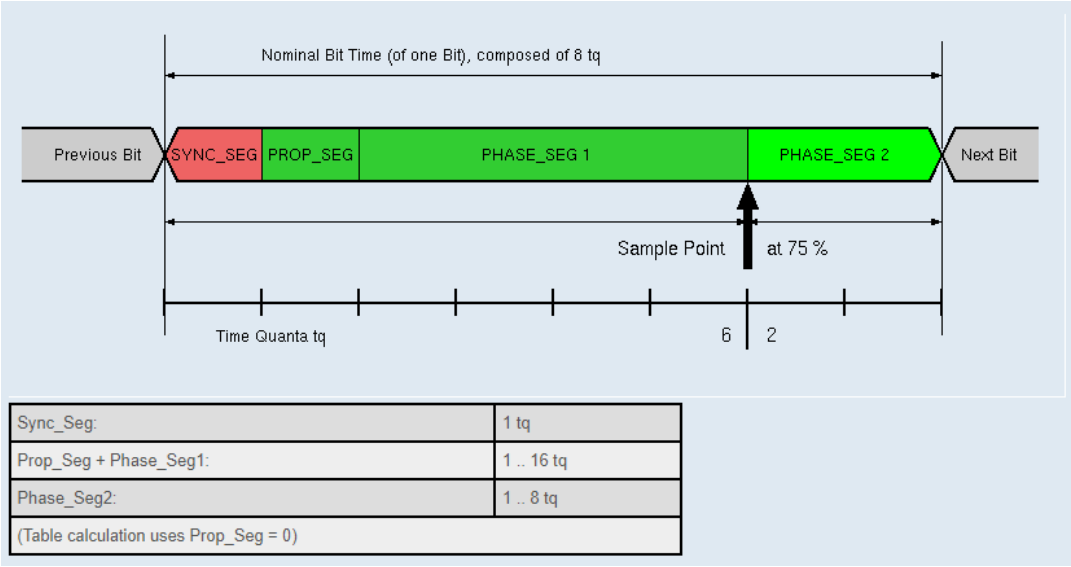


Figura 67 Exemplo de configuração do *Sample Point* do protocolo CANFD [83]

Tal como referido anteriormente, para que os diferentes controladores possam comunicar entre si, utilizando os barramentos, todos devem usar esta lista de configurações.

Além das configurações do protocolo, é sempre necessário definir os parâmetros utilizados para o cabeçalho das mensagens CANFD, sendo os diferentes campos definidos tal como na Figura 68.

```
TxHeader.IdType = FDCAN_EXTENDED_ID;
TxHeader.TxFrameType = FDCAN_DATA_FRAME;
TxHeader.ErrorStateIndicator = FDCAN_ESI_ACTIVE;
TxHeader.BitRateSwitch = FDCAN_BRS_OFF;
TxHeader.FDFormat = FDCAN_FD_CAN;
TxHeader.TxEventFifoControl = FDCAN_NO_TX_EVENTS;
```

Figura 68 Parâmetros utilizados no *header* das mensagens enviadas por CANFD

Além destes 6 campos, é necessário definir o *Identifier* (ID) e o *DataLenght* (quantidade de *bytes* de dados), embora, como estes variam de mensagem para mensagem, são apenas definidos no momento antes do envio.

7.2. CONTROLADOR PERIFÉRICO

O *firmware* do controlador periférico, tal como os dos restantes controladores, foi evoluindo ao longo de todo o projeto, tendo sido diversas vezes reescrito, corrigido, alterado e compilado, até chegar à versão final, que será apresentada de seguida. Esta versão, engloba todos os *firmwares* desenvolvidos para testar todos os diferentes sensores e protocolos de

comunicação, em um só, assim como também inclui toda a lógica para funcionamento do sistema. Primeiramente serão apresentadas as configurações dos diferentes periféricos e sensores, e posteriormente o ciclo principal de todo o programa.

O primeiro passo foi criar o projeto e gerar as configurações iniciais do mesmo. Isto inclui iniciar alguns periféricos, como por exemplo, a USART2, para permitir enviar mensagens para uma consola no computador via USB, facilitando o processo de *debug*. Isto é possível tirando partido do *debugger* incluído nas placas de teste.

7.2.1. CONFIGURAÇÃO DE PERIFÉRICOS

No controlador periférico foram configurados, além do CANFD, dois periféricos, I^2C, ADC e o DMA (*Direct Memory Access*). O primeiro, utilizado para comunicar com os diversos sensores externos, foi configurado da mesma forma que o periférico CANFD, utilizando a *Device Configuration Tool*. Abrindo a aba *Connectivity*, selecionou-se e ativou-se a interface I2C1, deixando todos os parâmetros em *default*, tal como se pode ver na Figura 69. No caso deste microcontrolador, a FDCAN1 é a única interface CAN disponível, mas, para I^2C, são disponibilizadas 3 interfaces (I2C1/2/3). Os dois pinos associados à interface I2C1, são PA15, para SCL, e PB7 para SDA. Depois de gravar as configurações, o código é gerado automaticamente pelo IDE.

Figura 69 Parâmetros de configuração do protocolo I^2C

De seguida, os periféricos ADC e DMA, foram programados em simultâneo pois, neste caso, são utilizados em conjunto.

O DMA, de *Direct Memory Access*, é um componente que permite que, tal como o nome indica, um qualquer periférico aceda diretamente à memória do microcontrolador, sem que os dados passem pelo processador. A vantagem de usar este componente, em conjunto com o ADC, é permitir que o ADC esteja continuamente a ler e a enviar os dados para um registo da memória, sem ocupar tempo de CPU (*Central Processing Unit*) [84].

Para conseguir interligar estes dois periféricos, é necessário configurar o ADC para funcionamento em DMA. Em primeiro lugar, através da *Device Configuration Tool*, na aba *Analog*, ativou-se as interfaces ADC1, com as entradas IN1 (PA0) e IN15 (PB0), e ADC2, com as entradas IN4 (PA7) e IN17 (PA4), todas em modo *Single-ended*. Nas configurações do utilizou-se uma resolução de 12-*bit* e o modo assíncrono com um *prescaler* de 64, tal como se verifica na Figura 70.

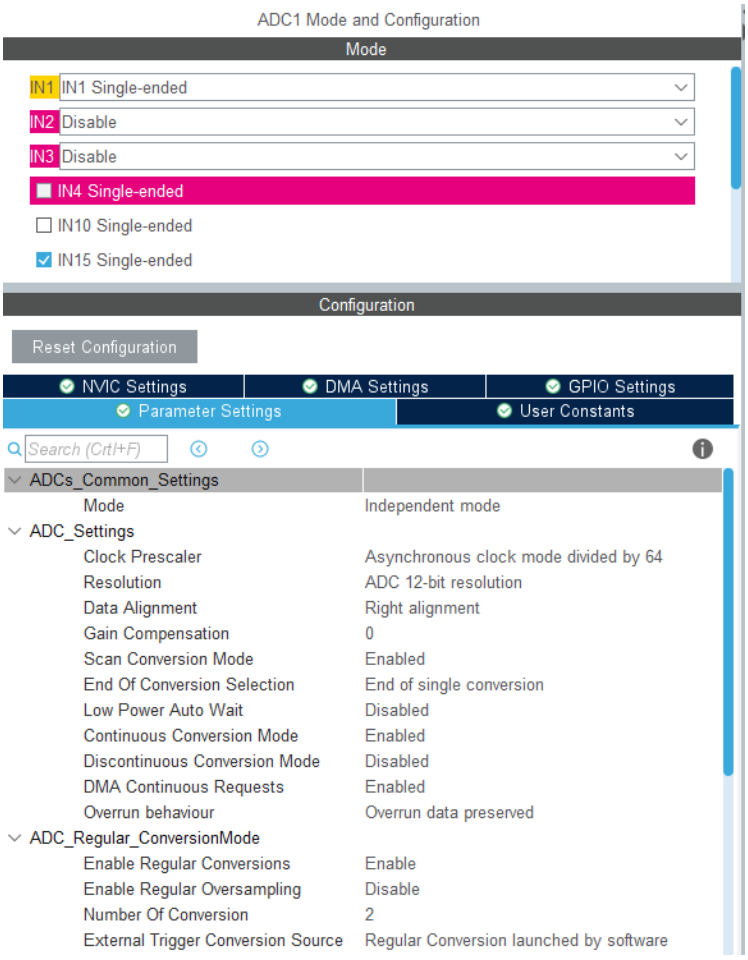


Figura 70 Parâmetros de configuração do ADC1

Depois de iniciados e configurados os dois ADC, é necessário configurá-los com o DMA. Através da aba *System Core*, selecionando a opção DMA, adiciona-se cada um dos ADC através do botão *Add*. Depois de adicionados, em cada um dos DMA *Request* altera-se para o modo Circular, para que os valores lidos sejam atualizados na memória automaticamente, e, para o tamanho de dados, opta-se pela opção *Word*, correspondente a um registo de 32 *bit*, tal como visível na Figura 71.

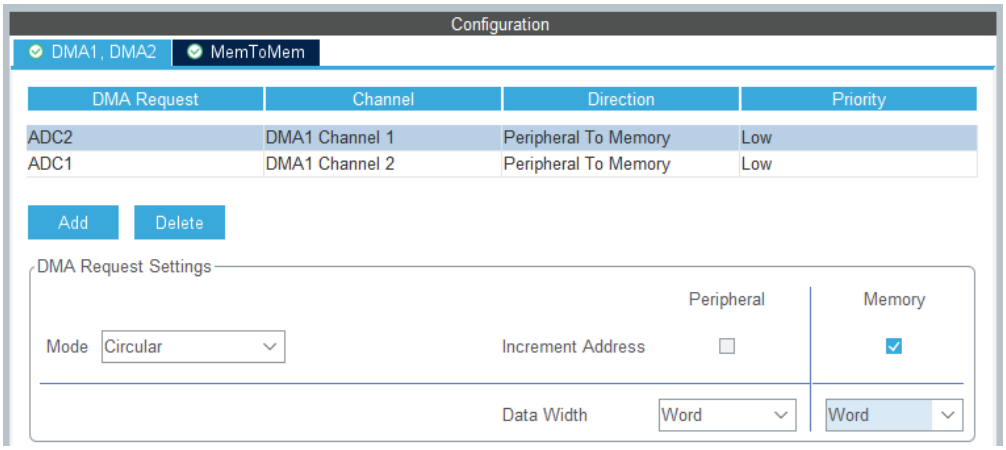


Figura 71 Configurações do DMA para funcionamento com ADC

De seguida configurou-se um pino do microcontrolador como interrupção externa. Para este efeito utilizou-se o pino PB4, ao qual se associou a função *GPIO_EXTI4*.

Por fim, os últimos periféricos a ser configurados foram os *timers* TIM7 e TIM16, através da aba *Timers*. Para o TIM7 foi usado um *prescaler* de 8 e um período de 60000. Com estes valores, o *timer* irá incrementar o seu contador a cada 0,1 µs, chegando a um máximo de 6 ms. No caso do TIM16 (Figura 72), foi utilizado um *prescaler* de 40000, resultando em frequência de 2 kHz (1 *tick* a cada 0.5 ms), e um período de 65536, correspondente a um máximo de 32 s.

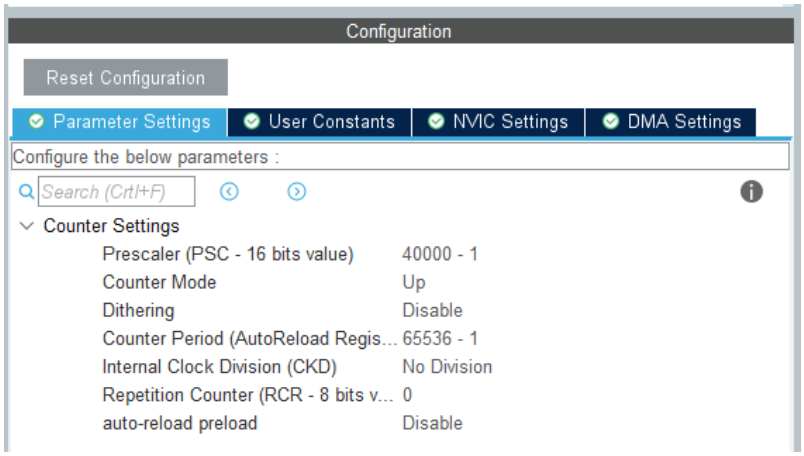


Figura 72 Configuração do TIM16

7.2.2. CICLO PRINCIPAL

Após finalizar toda a configuração dos diferentes periféricos e componentes de *hardware*, iniciou-se o processo de construção lógica do programa do controlador. Primeiramente criaram-se 3 novos ficheiros, com os seguintes nomes: *LA_Library*, para funcionar como uma biblioteca de funções, de modo a não ter todo o código no ficheiro *main*; *LA_FDCAN_Send*, que contém o *switch case* com a estrutura de todas as mensagens enviadas por este controlador e o número referente à posição do controlador face ao sistema, (número do controlador ilustrado na Figura 73); *LA_FDCAN_Receive*, contendo o *switch case* com todas as mensagens que este microcontrolador é capaz de receber.

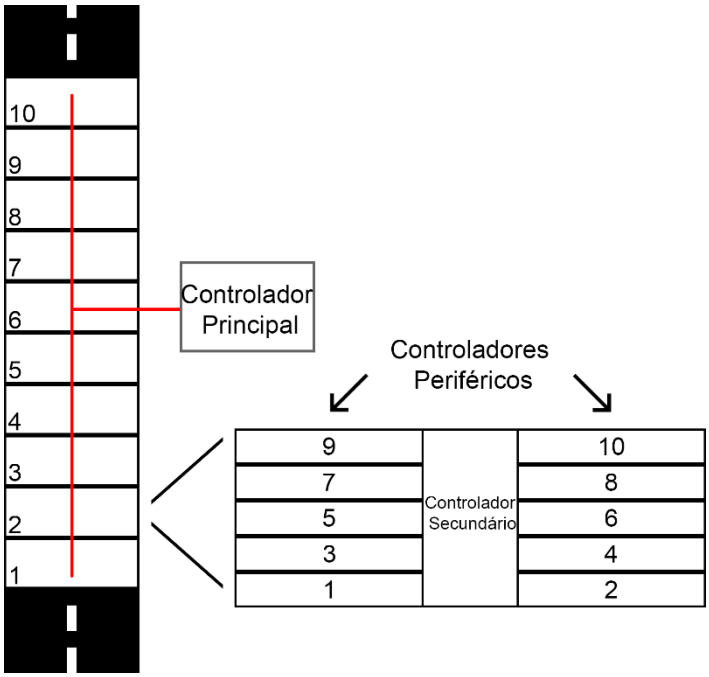


Figura 73 Esquema com numeração dos controladores

O ciclo principal do programa, contido no ficheiro *main.c*, pode ser representado pelo fluxograma da Figura 74. Considera-se o início deste fluxograma como o instante após a execução todas as configurações feitas automaticamente pelo *STM32Cube*, como é o caso da configuração de todos os periféricos.

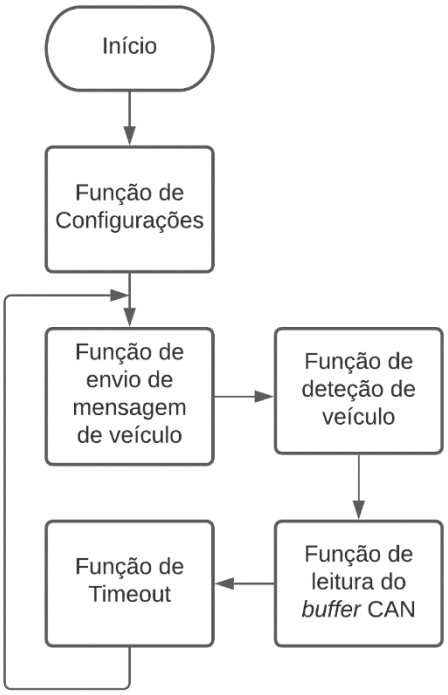


Figura 74 Fluxograma de ciclo principal do programa

Neste ciclo, além da função de configurações, onde são configurados e definidos os parâmetros iniciais de todos os sensores utilizados, são apenas utilizadas quatro funções. Utilizou-se esta abordagem de forma a construir um código o mais organizado e compreensível possível. Cada uma destas funções encontra-se no ficheiro *LA_Library* e representa uma parte diferente da estrutura do programa.

7.2.3. FUNÇÃO DE CONFIGURAÇÕES

A função de configurações, presente no ciclo principal do programa, pode ser representada pelo fluxograma da Figura 75.

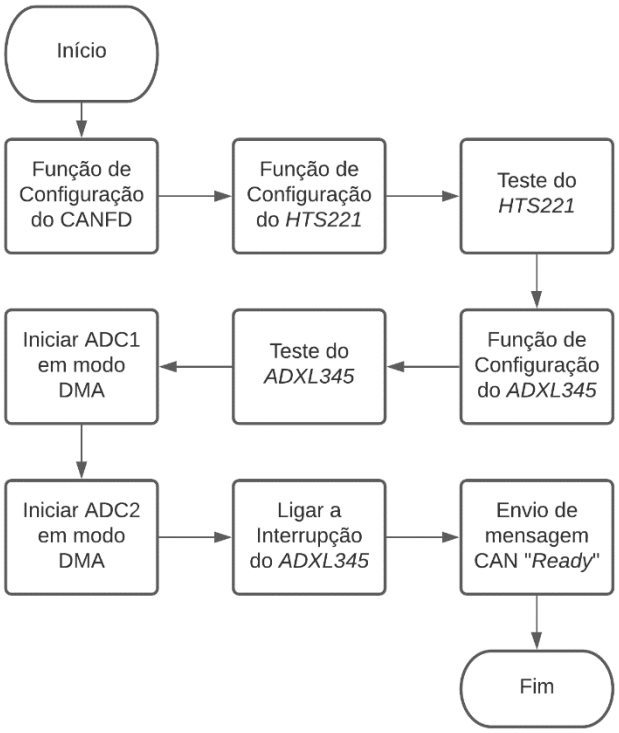


Figura 75 Fluxograma de Função de Configurações Iniciais

A primeira função executada é responsável por aplicar os filtros e configurações aos FIFO do periférico CANFD, e colocar o valor lógico “0” no GPIO conectado ao pino *standby* do *MCP2562FD*, fazendo com o que este se mantenha ligado.

São aplicados 2 filtros e ambos são referentes a IDs *extended* e aplicados ao FIFO0. O primeiro é um filtro que aceita todos os IDs compreendidos entre os valores 0x1000 e 0x10FF. Embora esta gama de valores não seja totalmente utilizada, estes representam os possíveis endereços de mensagens provenientes do controlador secundário para todos os controladores periféricos. O segundo filtro aceita todos os IDs de mensagens direcionadas àquele controlador periférico específico. Por este motivo, este filtro varia consoante o ID do controlador. No caso de o controlador ter o ID 3, o filtro aceita as mensagens com endereço compreendido dentre 0x1300 e 0x13FF. De seguida define-se que todas as mensagens devem ser filtradas, e, as que contenham IDs fora dos filtros aplicados, devem ser ignoradas.

A função seguinte é responsável por configurar o sensor *HTS221*, seguida de um teste, feito através da leitura do registo 0x00 do sensor. Se a resposta a esta leitura for 0xE5, significa que o sensor se encontra ativo e operacional. De seguida repete-se o processo, desta vez para o *ADXL345*, onde se executa a função de configuração deste sensor, e de seguida é testado, através da leitura do registo 0x0F, à qual a resposta deverá ser 0xBC. Depois de

configurados os dois sensores, são iniciados os ADC em modo DMA, é ativada a interrupção do acelerómetro e por fim é enviada uma mensagem CANFD, de modo a informar o controlador secundário que este controlador se encontra ativo e funcional.

Em ambas as funções de configuração dos sensores, são alterados diferentes registos dos dois componentes, consoante as configurações pretendidas. Além disso, em ambas as funções, são obtidos valores iniciais usados para efeitos de calibração. No caso do *HTS221*, são necessários recolher 8 valores constantes, para que posteriormente possam ser calculados os valores de temperatura e humidade. Este cálculo é feito com base nos gráficos da Figura 76.

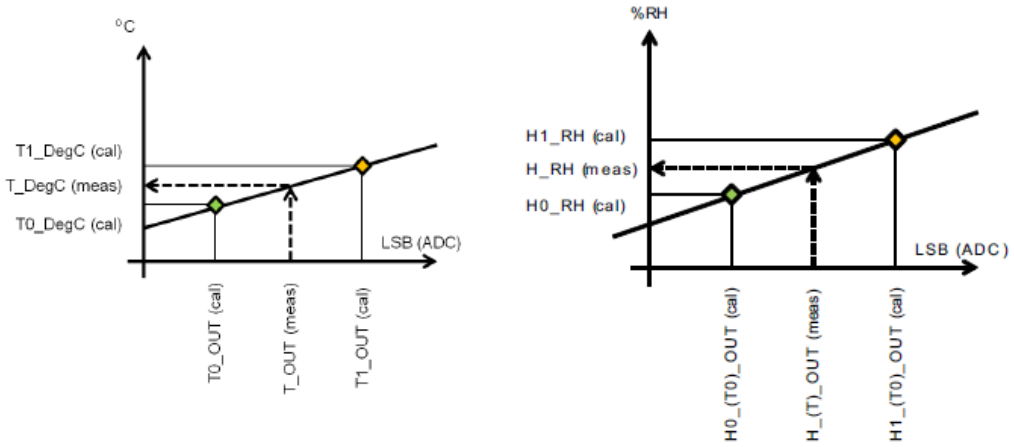


Figura 76 Gráficos de conversão de Temperatura e Humidade do sensor *HTS221* [72]

Obtendo os valores dos registos T0_OUT, T1_OUT, T0_DegC e T1_DegC é possível obter a equação característica da reta de temperatura. Assim, através da leitura do registo com o valor T_OUT, é possível obter o valor real de temperatura, representado por T_DegC. No caso da humidade, o processo é o mesmo, mas para os registos H0_OUT, H1_OUT, H0_RH e H1_RH, com o valor final a ser representado por H_RH.

A finalizar a função de configuração, tal como referido, é enviada uma mensagem CANFD. Este envio é feito através da função que, em primeiro lugar, define a estrutura de dados do pacote e depois coloca a mensagem no *buffer* de envio da interface CAN. Para configurar os parâmetros da mensagem a enviar, é chamada uma função que, através de um ID fornecido, percorre o *switch case* presente no ficheiro *LA_FDCAN_Send*, referido anteriormente. No caso da mensagem de "*Ready*", o ID utilizado é 00, que configura uma mensagem com este mesmo ID e com 0 *bytes* de dados, como se verifica na Figura 77.

```
switch(fdcan_id){

case 00: //Envio do Ready (Conexão à rede)
    TxHeader.Identifier = FDCAN_PER_ID | 0x00;
    TxHeader.DataLength = FDCAN_DLC_BYTES_0;
    break;
```

Figura 77 *Switch case* com mensagem CANFD de "*Ready"*

Este método de configuração dos parâmetros *Identifier*, *Data Lenght* e dados dos pacotes CANFD é utilizado para qualquer mensagem enviada.

7.2.4. FUNÇÃO DE ENVIO DE MENSAGENS

Após a função de configurações, o ciclo do programa entra *loop* principal, começando por executar a função de envio de mensagens de veículo, representada pelo fluxograma da Figura 78.

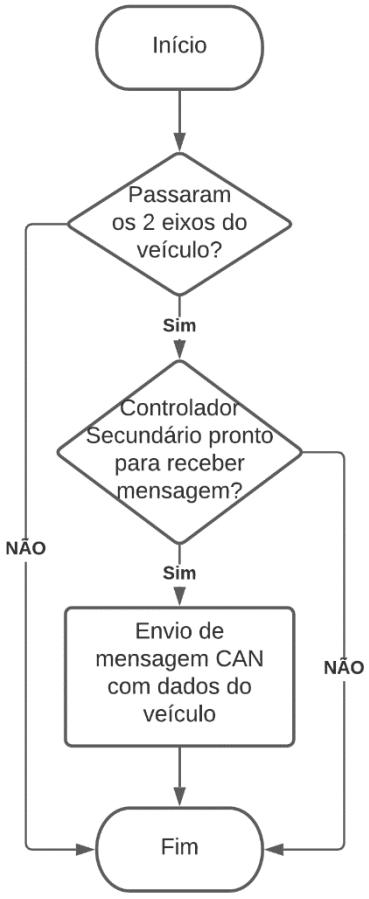


Figura 78 Fluxograma de função de envio de mensagem com dados do veículo

Esta função certifica-se que passaram ambos os eixos do veículo pelo módulo e a todos os dados a enviar estão calculados e carregados no *buffer*. Depois de os dados estarem prontos a ser enviados, o controlador espera que os dados sejam solicitados, por parte do controlador secundário. Todas estas verificações são feitas através de *flags*, controladas por outras funções, descritas de seguida.

7.2.5. FUNÇÃO DE DETEÇÃO DE VEÍCULOS

Seguido da função de envio dos dados para o controlador secundário, o programa entra numa das funções principais do sistema, que monitoriza e controla em que momento da passagem do veículo é que o módulo se encontra, a função de deteção do veículo, representada pelo fluxograma da Figura 79.

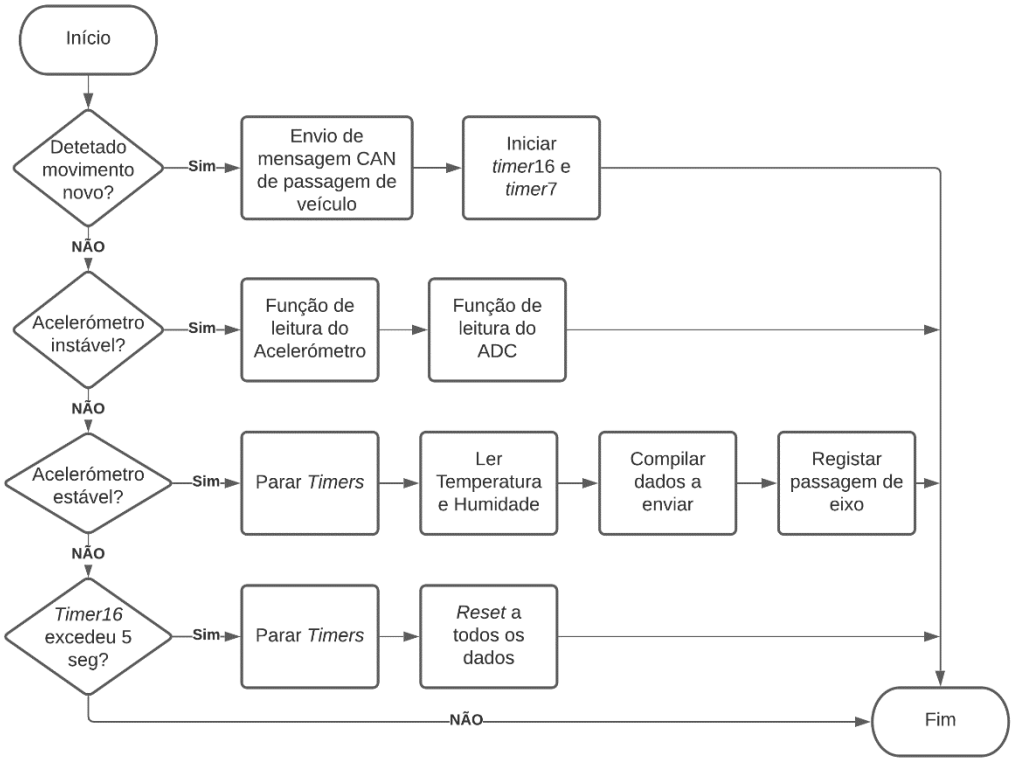


Figura 79 Fluxograma da função de deteção de veículo

Nesta função o sistema passa por diferentes verificações, sendo a primeira a deteção de movimento no módulo, ativada pela interrupção associada ao *ADXL345*. Se se detetar algum movimento, é enviada uma mensagem CANFD de alerta, que avisa o controlador secundário da passagem de um veículo e de seguida são iniciados os *timers*.

Caso o movimento tenha sido detetado, mas o acelerómetro ainda não tenha valores estáveis, significa que a tampa ainda não voltou à sua posição inicial, logo, neste momento vão ser executadas as funções de leitura do acelerómetro e dos ADC.

A função de leitura do acelerómetro tem como objetivo determinar a posição da tampa do módulo e registar os valores máximos de cada um dos seus 3 eixos. De forma a perceber a posição da tampa, relativa à posição inicial, estudou-se o comportamento do acelerómetro e os valores obtidos através do eixo Z.

Através deste estudo, percebeu-se a existência de um padrão nos valores de aceleração de Z sempre que a tampa do módulo fazia um movimento completo. Utilizando uma ferramenta de *plotter*, foi possível desenhar o gráfico representado na Figura 80.

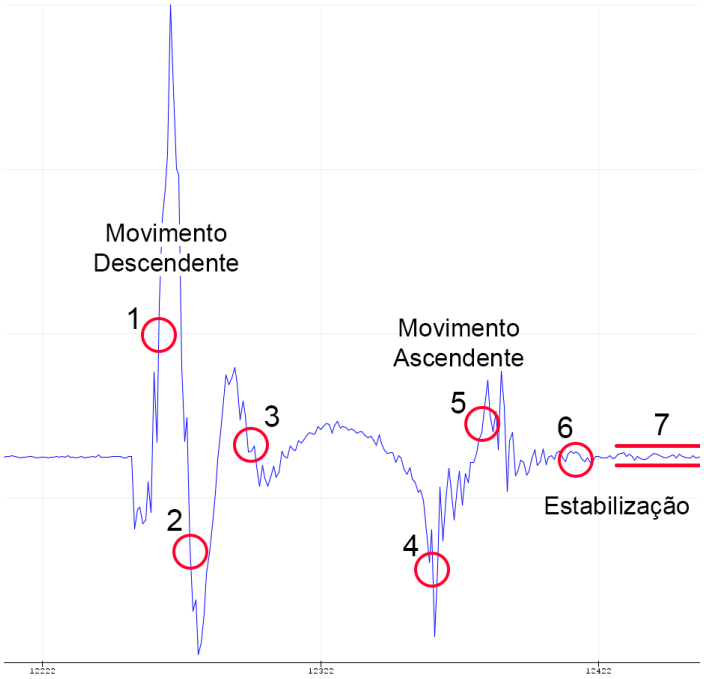


Figura 80 Gráfico dos valores de Z com o movimento da tampa do módulo *NEXT-road*

Com estes dados, desenvolveu-se um algoritmo que através da análise constante dos valores obtidos do eixo Z e comparando com os pontos assinalados na Figura 80, se consegue saber exatamente a posição da tampa do módulo. Os primeiros três pontos, representam o movimento descendente e os últimos 3 pontos representam o movimento ascendente da tampa. Após a passagem por estes 6 pontos, considera-se que o acelerómetro estabiliza assim que os últimos 30 valores obtidos estejam compreendidos num intervalo, representado pelas duas linhas vermelhas da figura.

Após a realização de alguns testes, atribuíram-se os seguintes valores aos diferentes pontos e intervalo:

1. Maior que 2.5 g;
2. Menor que -0.2 g;
3. Maior que 0.85 g;
4. Menor que -0.2 g;
5. Maior que 2.5 g;
6. Menor que 1.1 g;
7. Intervalo de valores entre 0.95 g e 1.05 g.

Se a estabilização do acelerómetro ocorrer até 5 segundos depois da ativação do *timer*, considera-se a correta passagem de um eixo de um veículo. Caso contrário, descartam-se os dados obtidos e o sistema volta ao estado inicial.

Depois da função de análise do acelerómetro, é executada a função de leitura de dados do ADC, cujo funcionamento pode ser representado pelo fluxograma da Figura 81.

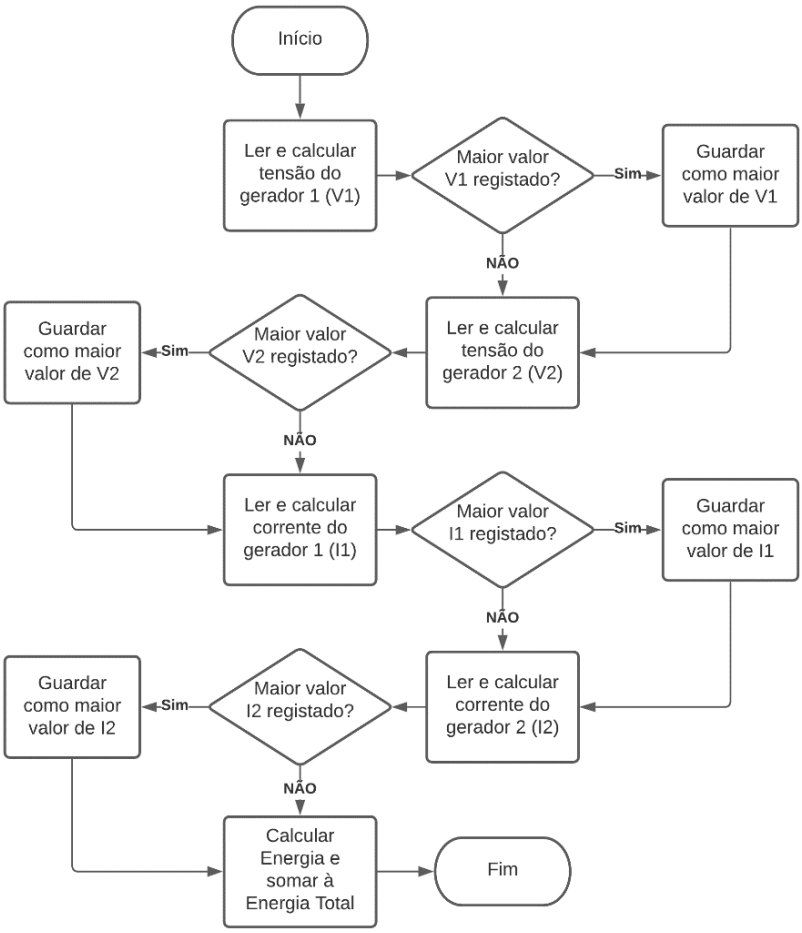


Figura 81 Fluxograma da função de leitura do ADC

Como o ADC está configurado em modo DMA, então a leitura dos valores é feita diretamente através de um registo, facilitando o processo, no entanto, os valores obtidos necessitam de ser convertidos para valores reais. No caso das tensões, os valores reais podem ser obtidos utilizando a equação 3. Para as correntes, é utilizada a equação 4.

$$V = \frac{\left(\frac{(ADC \times 3.3)}{4096} + 0.0787\right)}{0.0876} \quad (3)$$

$$I = \frac{\left(\frac{(ADC \times 3.3)}{4096} - 1.5921\right)}{0.1286} \quad (4)$$

Estas fórmulas foram obtidas através da análise dos valores de tensão nas entradas do ADC, face aos valores de tensão e corrente à entrada do divisor de tensão e sensor de corrente, respetivamente, utilizando um multímetro. Através da ferramenta de cálculo *Excel*, traçou-se os gráficos, apresentados no Anexo C abaixo, e obteve-se a equação das retas. Assim que são calculadas, caso o valor obtido seja mais elevado que o maior valor registado até ao momento, este último é substituído pelo maior valor atual.

Depois das comparações, os valores de tensão e corrente são multiplicados de forma a obter a potência dos dois geradores. Os valores de potência são finalmente multiplicados por um intervalo de tempo, obtido através do *timer7*, correspondente ao tempo entre leituras, que permite obter um valor de energia. Esta energia é, por fim, somada a um registo de energia total de cada gerador.

Se a passagem do eixo do veículo pelo módulo for um sucesso, os *timers* são parados, são obtidos os valores de temperatura e humidade, e finalmente todos os dados são devidamente compilados em uma estrutura, para futuramente serem enviados para o controlador secundário.

7.2.6. FUNÇÃO DE LEITURA DE MENSAGENS CANFD

Na função de leitura de mensagens CANFD, o programa confirma se existe alguma mensagem CANFD à espera de ser lida, através da leitura do registo *FDCAN_RX_FIFO0*, que indica exatamente quantas mensagens estão em fila de espera. Se existir alguma mensagem em espera, esta é lida e de seguida o programa entra no *switch case* do ficheiro *LA_FDCAN_Receive*, para processar a mensagem de forma adequada, consoante o seu ID. Este processo pode ser representado pelo fluxograma da Figura 82.

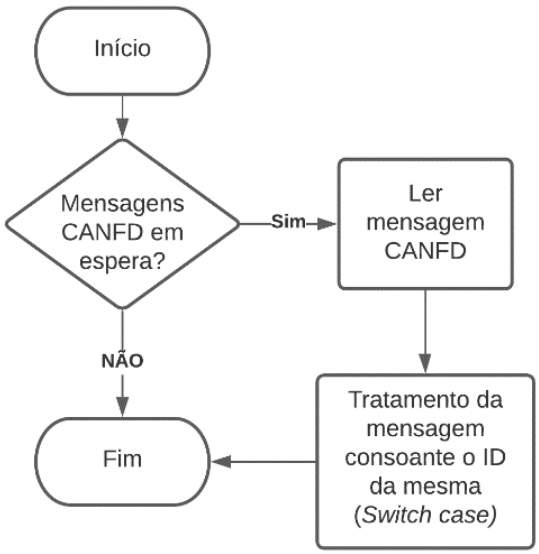


Figura 82 Fluxograma da função de leitura de mensagens CANFD

O controlador periférico pode receber 2 tipos diferentes de mensagens, enviadas por parte do controlador secundário. A primeira é uma mensagem, sem dados, enviada a todos os periféricos, à qual estes devem responder. Desta forma o controlador secundário sabe quais controladores periféricos, do seu barramento, estão ativos.

O segundo tipo de mensagem refere-se ao pedido de envio de dados, após a passagem dos 2 eixos do veículo, referido no subcapítulo 7.2.4

7.2.7. FUNÇÃO DE *TIMEOUT*

Para finalizar o *loop* principal, é chamada a função de *timeout* que tem como objetivo fazer o programa voltar ao estado inicial caso haja algum problema com a passagem dos veículos pelo módulo. No caso de um primeiro eixo ser registado com sucesso, mas não existir um segundo eixo num espaço de 5 segundos, o *timeout* é ativado, e pode ser representado pelo fluxograma da Figura 83.

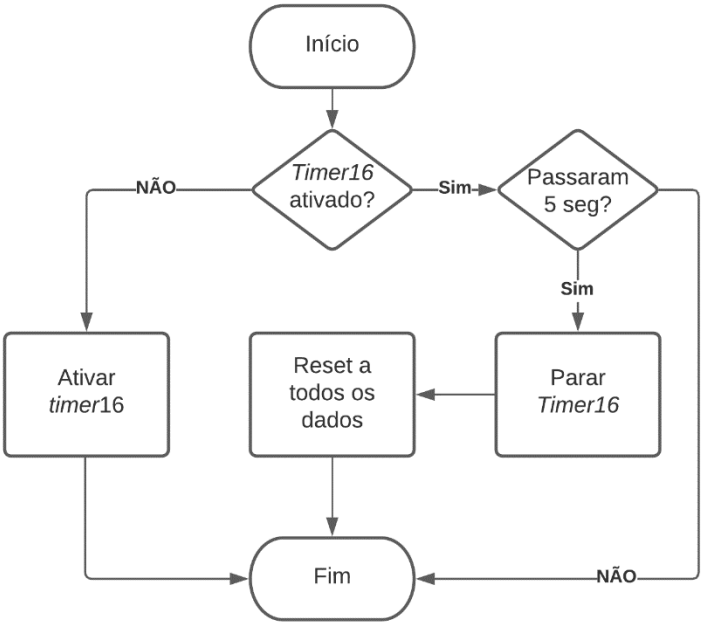


Figura 83 Fluxograma da função de *timeout*

Depois de executada esta função, o *loop* principal volta a executar a sua primeira função, o envio de mensagem para com os dados do veículo.

7.3. CONTROLADOR SECUNDÁRIO

Tal como aconteceu como o *firmware* do controlador periférico, o *firmware* do controlador secundário sofreu inúmeras alterações ao longo de todo o projeto, sendo que o a versão aqui apresentada, refere-se à versão mais atual.

Depois de criado o projeto para o *firmware* do novo controlador, o *STM32Cube* configura automaticamente alguns dos periféricos da *NUCLEO board*, de maneira a ativar funções de *debug*, tais como o periférico LPUART1, usado para o envio de dados por USB para um computador. De seguida, da mesma forma que no *firmware* do controlador periférico, no *firmware* do secundário foram criados 3 novos ficheiros ao projeto: *LA_library*, *LA_FDCAN_Send* e *LA_FDCAN_Receive*, tendo estes, funções semelhantes ao caso anterior. Além dos ficheiros, também as configurações dos periféricos, feitas utilizando a *Device Configuration Tool*, são semelhantes ao controlador periférico. Neste caso, com exceção dos periféricos CANFD, foram ativados e configurados os *timers* 6, 7 e 16, todos com os mesmos parâmetros, que podem ser visualizados na Figura 84.

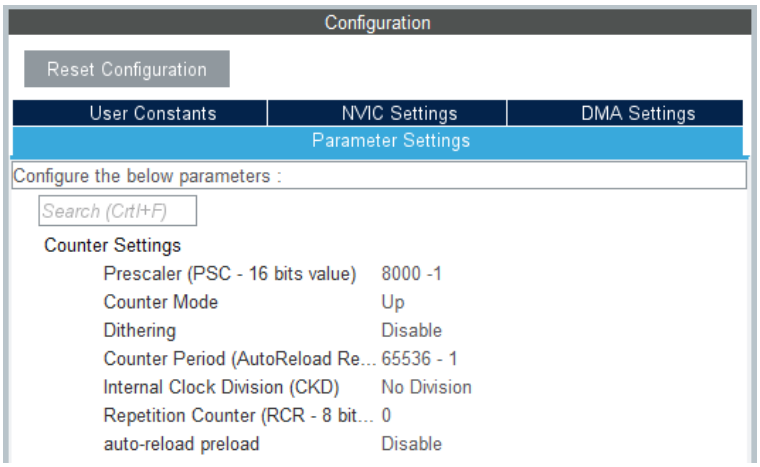


Figura 84 Configurações dos *timers* 6, 7 e 16 do controlador secundário

Com estes valores, os *timers* operam a uma frequência de 10 kHz, ou seja 1 *tick* a cada 0,1 ms. Com um período de 65536, o tempo máximo associado aos seus contadores é de 6,55 s.

7.3.1. CICLO PRINCIPAL

O Ciclo principal do programa segue a lógica do fluxograma apresentado na Figura 85. Primeiramente, tal como no periférico, os GPIO associados aos pinos dos *transceivers* CANFD são colocados a nível lógico "0". De seguida, é enviada uma mensagem de pedido de *status* para o barramento secundário. Com o envio desta mensagem, os controladores periféricos ativos devem responder, de modo que o controlador secundário saiba que se encontram ativos e prontos a recolher dados com a passagem de veículos.

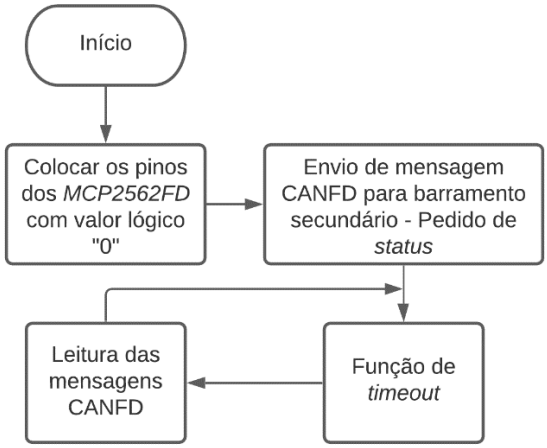


Figura 85 Fluxograma do *loop* principal do ficheiro *main* do controlador secundário

Após a receção das mensagens, o controlador entra no *loop* principal, onde executa a função de *timeout* e de seguida lê ambos os *buffers* de entradas de mensagens CANFD dos periféricos FDCAN1 e FDCAN2.

A função de *timeout* tem com objetivo interromper o funcionamento do programa e colocar todas as variáveis com os valores iniciais, para que o programa não fique preso em algum *loop* que não o principal. Apenas é ativada se o controlador não obtiver novas informações dos periféricos num intervalo de 5 segundos, sendo este tempo gerido pelo *timer* 6.

A leitura das mensagens CANFD é feita da mesma forma que todos os outros controladores, com a exceção de verificar o *buffer* de cada um dos barramentos. Assim, sempre que for recebida uma mensagem, o programa é direcionado para o *switch case* do ficheiro *LA_FDCAN_Receive*.

Toda a lógica deste controlador está associada à receção das mensagens de deteção de movimento de cada um dos controladores periféricos. Embora, num caso ideal, o sistema devesse incluir 10 periféricos por secundário, na prática, apenas foram utilizados 4 controladores periféricos por controlador secundário, logo o *firmware* teve de ser alterado, face ao planeado inicialmente. Os controladores periféricos utilizados foram colocados nas posições 3, 4, 7 e 8. Através da receção destas mensagens, o controlador secundário sabe quais periféricos foram ativos, quantos eixos passaram pelo sistema e, no final, calcula as velocidades de entrada e saída dos veículos. O fluxograma da Figura 86 representa o funcionamento do programa na receção das mensagens de movimento dos controladores periféricos 3 e 4, sendo este semelhante. Se forem recebidas mensagens de deteção de veículo provenientes dos periféricos 7 e 8, o programa tem um funcionamento representado pelo fluxograma da Figura 87. O objetivo final da receção das deteções dos 4 controladores periféricos (combinação dos dois fluxogramas), é executar a função de envio da mensagem do veículo para o controlador principal.

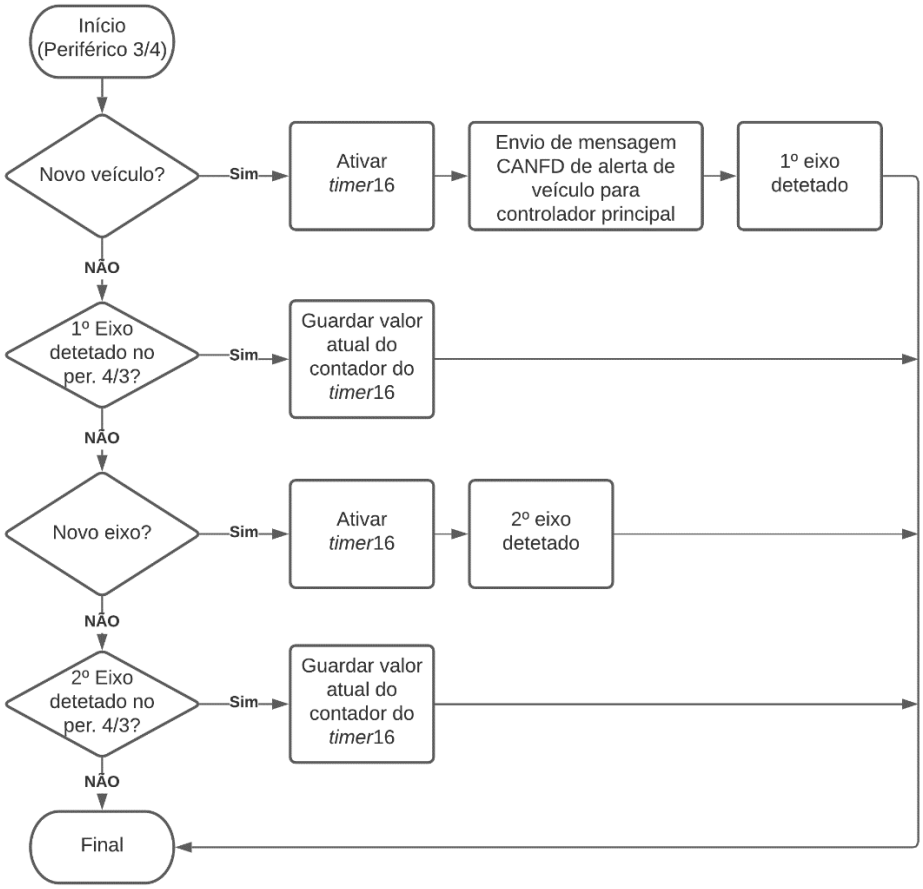


Figura 86 Fluxograma de receção de mensagens de deteção de veículo dos periféricos 3 e 4

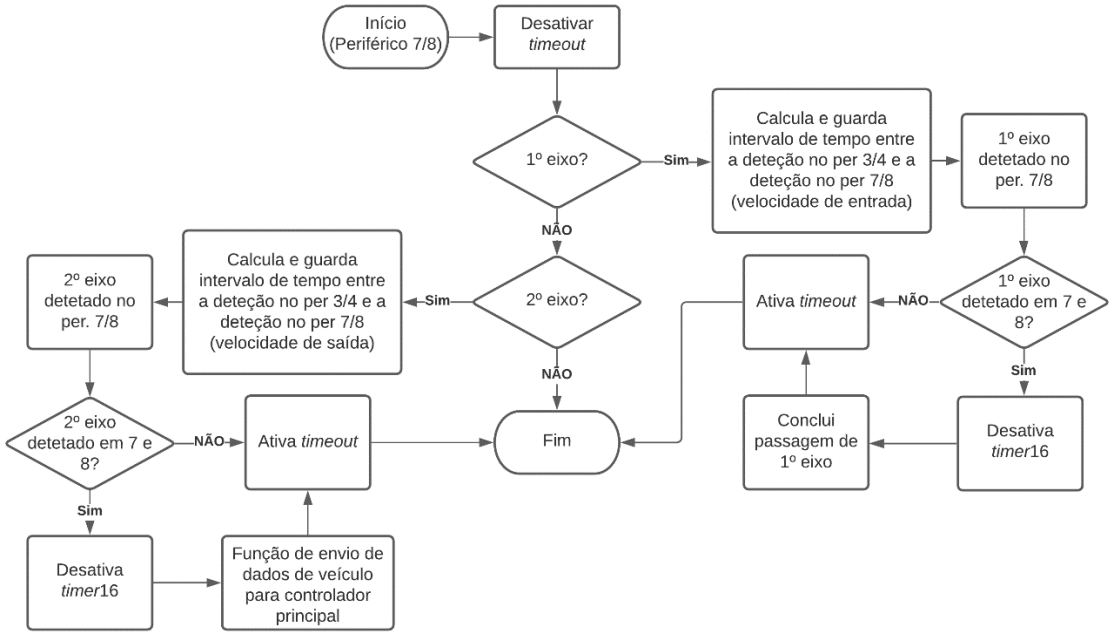


Figura 87 Fluxograma de receção de mensagens de deteção de veículo dos periféricos 7 e 8

Assim que o programa entra na função de envio de dados de veículo para o controlador principal, é enviada uma mensagem CANFD para o barramento secundário a pedir o envio dos dados recolhidos pelos controladores periféricos 3 e 4. Após o envio desta mensagem, é ativado um *timeout* de 5 segundos, controlado pelo *timer*7, para que, caso as mensagens dos periféricos não sejam enviadas, o controlador secundário prossiga o seu funcionamento. Caso sejam recebidas as 2 mensagens ou passem 5 segundos, o *timeout* é desativado e é enviada uma nova mensagem para o barramento secundário a pedir os dados da passagem do veículo aos periféricos 7 e 8. Tal como para o pedido anterior, o *timeout* de 5 segundos é ativado e o controlador espera por receber as 2 respostas. Após receber as respostas ou passarem 5 segundos, o programa executa a função de cálculo e compilação de dados a enviar e de seguida executa a função de verificação de erros. Por fim, envia a mensagem com todos os dados, para o controlador principal, e o programa volta ao seu estado inicial. O funcionamento desta função pode ser visualizado através do fluxograma da Figura 88.

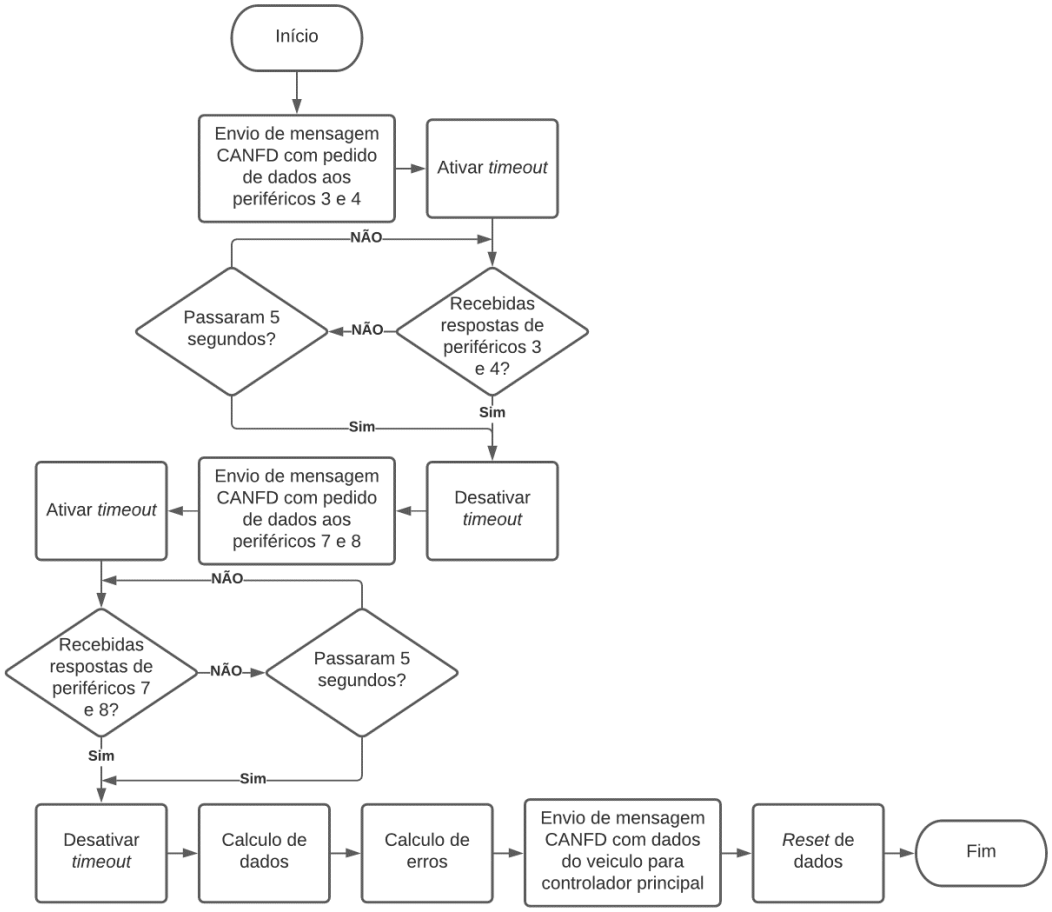


Figura 88 Fluxograma de função de envio de dados para o controlador principal

No cálculo de dados, o programa calcula as velocidades de entrada e saída dos veículos, através da distância entre os módulos, de 70 cm, e o intervalo de tempo que o veículo demora a passar pelos diferentes módulos, ou seja, o tempo que demora a percorrer essa distância. Ao fazer este cálculo, através da equação 5, é também feita a conversão da velocidade obtida para km/h.

$$Velocidade = \left(\frac{70}{\left(\left(\frac{intervalo1 + intervalo2}{2}\right) \times 0.0001\right)}\right) \times \frac{36}{1000} \quad (5)$$

Nesta equação, o intervalo 1 refere-se ao intervalo de tempo que um eixo demora a passar nos módulos 3 e 7, e o intervalo 2 ao tempo que demora a passar nos módulos 4 e 8.

A função de calculo de erros é constituída por um conjunto de condições que verificam se algum dos dados recebido não está dentro dos valores aceitáveis. Para que seja possível enviar os erros para o controlador principal, juntamente com os restantes dados, utilizaram-se variáveis de 8 *bits* para cada controlador periférico, nas quais cada *bit* representa o erro em um tipo de dados. Caso a condição de erro se confirme, o *bit* correspondente é colocado a "1", seguindo o seguinte formato:

- *Bit* 0 – Corrente ou tensão do gerador 1 igual a 0;
- *Bit* 1 – Corrente ou tensão do gerador 2 igual a 0;
- *Bit* 2 – Temperatura menor que 0 ºC ou maior que 70 ºC;
- *Bit* 3 – Humidade menor que 50 % ou maior que 90 %;
- *Bit* 4 – Valores anormais na aceleração do eixo Z;
- *Bit* 5 – Valores anormais na aceleração do eixo Y;
- *Bit* 6 – Valores anormais na aceleração do eixo X.

7.4. CONTROLADOR PRINCIPAL

A parte do *firmware* do controlador principal, correspondente a este projeto, foi desenvolvida em um ficheiro separado (*LA_FDCAN*) de forma a facilitar a integração com o restante *firmware* já desenvolvido em outros projetos. Antes do *loop* principal do programa, o dispositivo envia uma mensagem CANFD para todos os controladores secundários, com o intuito de saber quais se encontram ativos, juntamente com os respetivos controladores periféricos. Esta informação é guardada em uma matriz criada para 3 faixas, 10 controladores secundários por faixa e 10 controladores periféricos por secundário. No

loop principal de funcionamento do programa, foi introduzida a função de leitura de mensagens CANFD, tal como ilustra o fluxograma da Figura 89. Desta forma, tal como nos restantes controladores, caso exista alguma mensagem CANFD em espera, a mensagem é lida e os dados recebidos são processados tendo em conta o ID da mensagem.

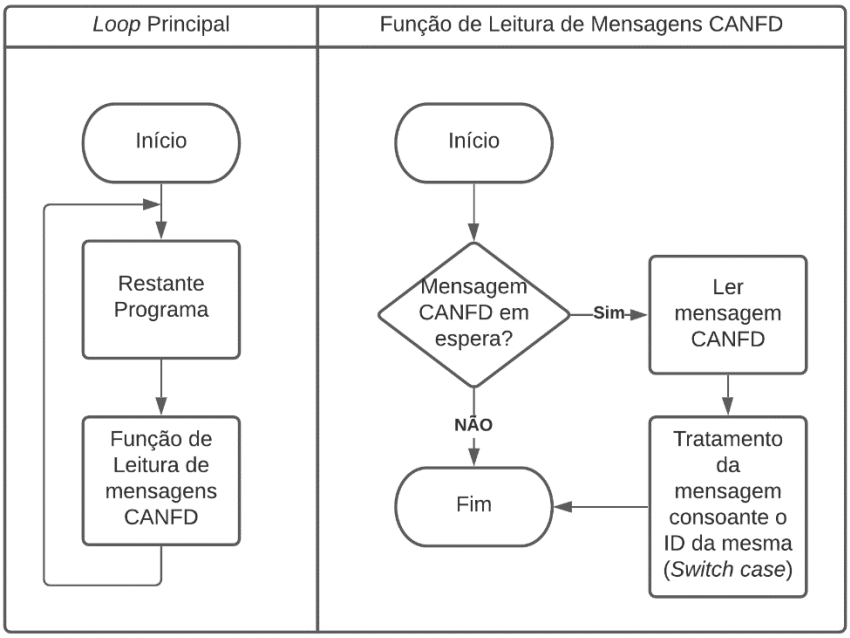


Figura 89 Fluxograma de função de leitura de mensagens CANFD inserida no controlador principal

Além desta função de leitura, está também presente no ficheiro, a função de envio de mensagens CANFD, necessária para transmitir informação para o barramento principal. A sua utilização, atualmente, limita-se ao envio de uma mensagem, antes do *loop* principal, para determinar quais controladores periféricos estão conectados ao barramento.

8. RESULTADOS

Neste capítulo, serão apresentados os vários testes e respetivos resultados em diferentes ambientes e etapas do projeto. Começando pelos primeiros testes realizados no decorrer da implementação de *hardware* e *firmware*, onde o objetivo foi perceber como configurar da melhor maneira os periféricos, sensores e restantes algoritmos do programa. Como a grande maioria deste tipo de testes já foram expostos em capítulos anteriores, neste subcapítulo, serão apenas referidos mais alguns testes. De seguida serão apresentados os testes realizados com o *hardware* instalado nos módulos *NEXT-road*, em ambiente de laboratório. Por fim, serão apresentadas algumas fotografias e alguns resultados dos testes realizados em ambiente real, com os módulos *NEXT-road* instalados em uma estrada em Matosinhos.

8.1. TESTES INICIAIS

Tal como descrito anteriormente, os primeiros testes foram realizados de maneira a configurar corretamente todos os componentes do sistema. Além dos previamente apresentados, um dos primeiros e principais testes realizado foi o teste do protocolo CANFD e o envio de mensagens entre duas *NUCLEO-G474RE*, sem a utilização de um *transceiver* e, consequentemente, barramento CAN. Para tal, utilizaram-se dois fios que se ligaram,

alternadamente, aos terminais TX e RX, do protocolo CAN. Isto significa que o pino PA12 (TX) da primeira placa foi ligado ao pino PA11 (RX) da segunda placa, e vice-versa. Desta forma, e utilizando os diferentes modos de teste, apresentados no subcapítulo 7.1, estabeleceu-se a primeira comunicação CANFD.

De seguida, com o auxílio de uma *breadboard*, ao circuito anterior, foram adicionados os dois *transceivers*, criando assim o primeiro barramento CANFD. Utilizando um osciloscópio digital, ligado aos dois terminais do barramento, foi possível visualizar a passagem do pacote de dados, tal como se verifica na Figura 90.

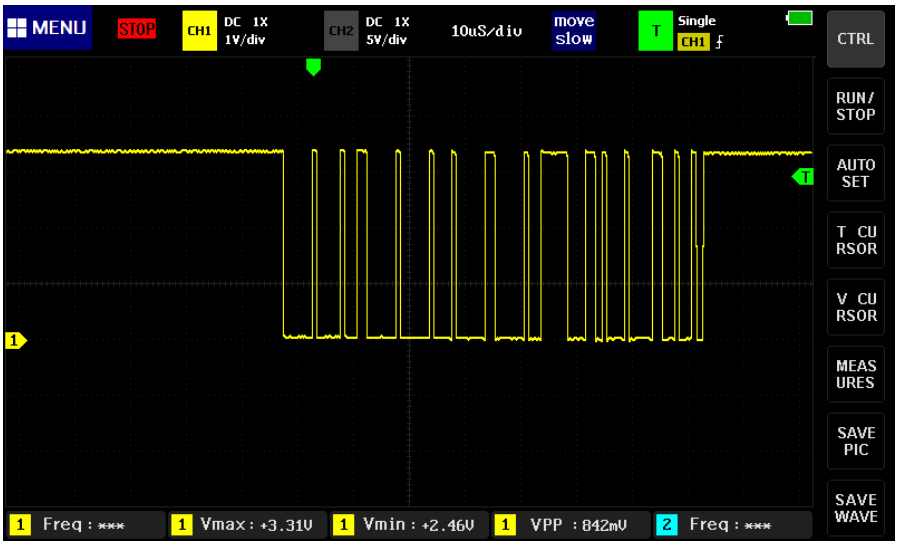


Figura 90 Pacote de dados CANFD visualizado no barramento utilizando um osciloscópio

Após o sucesso deste teste, acrescentou-se um conjunto *HTS221* + *ADXL345* ao circuito, com o intuito de enviar os captar valores de temperatura, em uma placa de testes e enviar esses dados para a segunda placa através do barramento CANFD. Na Figura 91 é possível visualizar as duas placas de teste, ligadas por USB a um computador, e também ligadas entre si através dos *transceivers* e barramento. No monitor, são apresentadas as duas consolas, sendo que a da esquerda representa o microcontrolador ao qual está ligado o sensor, e a da direita o microcontrolador que recebe as temperaturas e as imprime na consola.

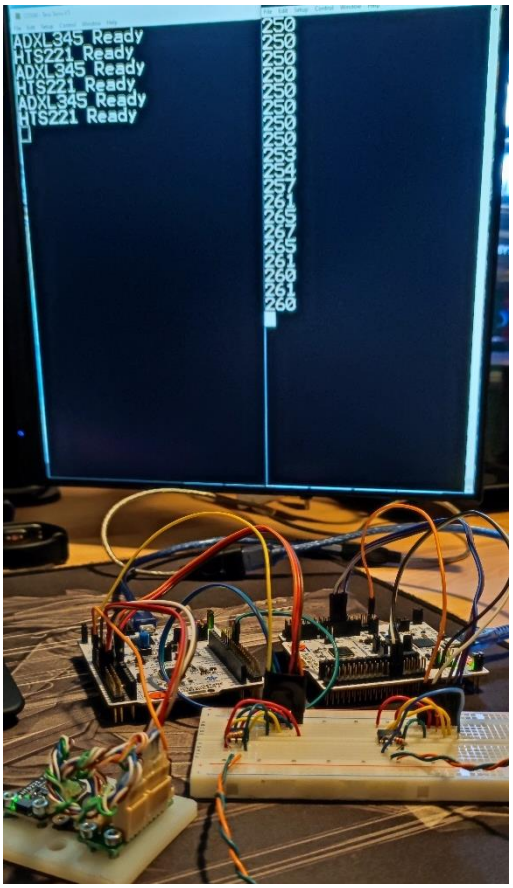


Figura 91 Fotografia do teste de envio de temperaturas entre microcontroladores

8.2. TESTES EM LABORATÓRIO COM MÓDULOS *NEXT-ROAD*

Assim que foram finalizadas as primeiras versões das PCB dos controladores periféricos e secundários, iniciaram-se os primeiros testes utilizando os módulos *NEXT-road*.

O primeiro teste efetuado, foi feito apenas com um módulo, ao qual, à sua tampa, se acoplou um conjunto *HTS221 + ADXL345* e ligou um dos geradores ao *PAC1934*, com o auxílio de uma carga resistiva. Este teste foi importante para calibrar o acelerómetro e obter os primeiros valores de aceleração utilizando o *NEXT-road*, ao mesmo tempo que se obteve os primeiros valores de energia do gerador, através do *PAC1934*. Foi também através deste teste que se criou o algoritmo para deteção da posição da tampa, tal como referido no subcapítulo 7.2.5 e na Figura 80.

Nas figuras seguintes (Figura 92 e Figura 93) é possível visualizar a montagem efetuada e a consola com os dados de aceleração, energia e passagem da tampa pelos 6 pontos de estabilização da tampa.

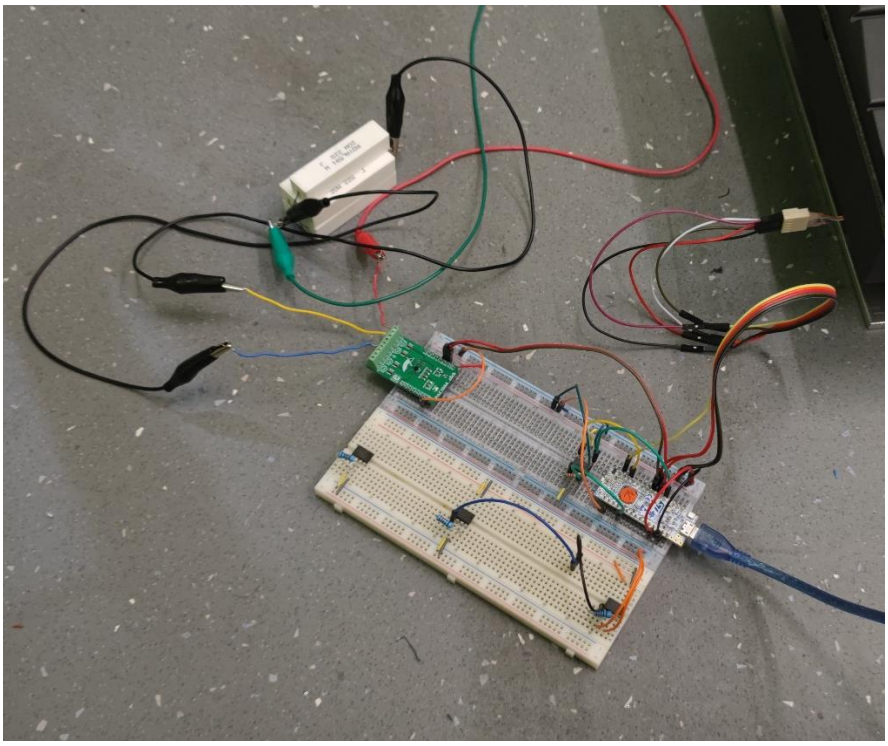


Figura 92 Fotografia da primeira montagem utilizando o módulo *NEXT-road*

```
flag = 6
x/y/z max: 2.720000 / 3.336000 / 6.896000
V_max: 11.020187 / I_max: 2188.143799 / E_total: 0.056268
Interrupt Activated
flag = 1
flag = 2
flag = 3
flag = 4
flag = 5
flag = 6
x/y/z max: 2.808000 / 3.888000 / 5.680000
V_max: 7.800900 / I_max: 1521.705933 / E_total: 0.015069
Interrupt Activated
flag = 1
flag = 2
flag = 3
flag = 4
flag = 5
flag = 6
x/y/z max: 1.680000 / 2.528000 / 6.224000
V_max: 11.052415 / I_max: 2251.468750 / E_total: 0.027085
Interrupt Activated
flag = 1
flag = 2
flag = 3
flag = 4
flag = 5
flag = 6
x/y/z max: 0.928000 / 2.488000 / 5.008000
V_max: 4.743725 / I_max: 981.918030 / E_total: 0.005532
```

Figura 93 Consola de *debug* do primeiro teste com o módulo *NEXT-road*

O próximo teste apresentado foi um dos últimos feitos em laboratório, que permitiu finalizar o *firmware* e simular a passagem de um veículo pelos vários módulos. A montagem utilizou 4 controladores periféricos, cada um com microcontrolador, acelerómetro e sensor de temperatura e humidade, ligados a um controlador secundário, através do barramento CANFD. Além disso, os geradores foram ligados a uma carga ao mesmo tempo que as tensões e correntes à saida de cada controlador periférico eram captadas. Uma diferença para o teste anterior é que neste utilizaram-se as versões mais recentes das PCB (V2) e a alimentação de energia do sistema foi feita através de uma fonte de alimentação regulável.

Por fim, o controlador secundário foi ligado ao controlador principal, através do barramento principal CANFD, para assim criar o primeiro sistema completo, com a presença dos três tipos de controladores. Desta forma, o controlador principal pacotes de dados incluindo: velocidade inicial, velocidade final, temperaturas, humidades, energias geradas e erros. Na Figura 94 é apresentada uma fotografia da montagem, onde se visualiza o controlador secundário, rodeado dos módulos *NEXT-road*.

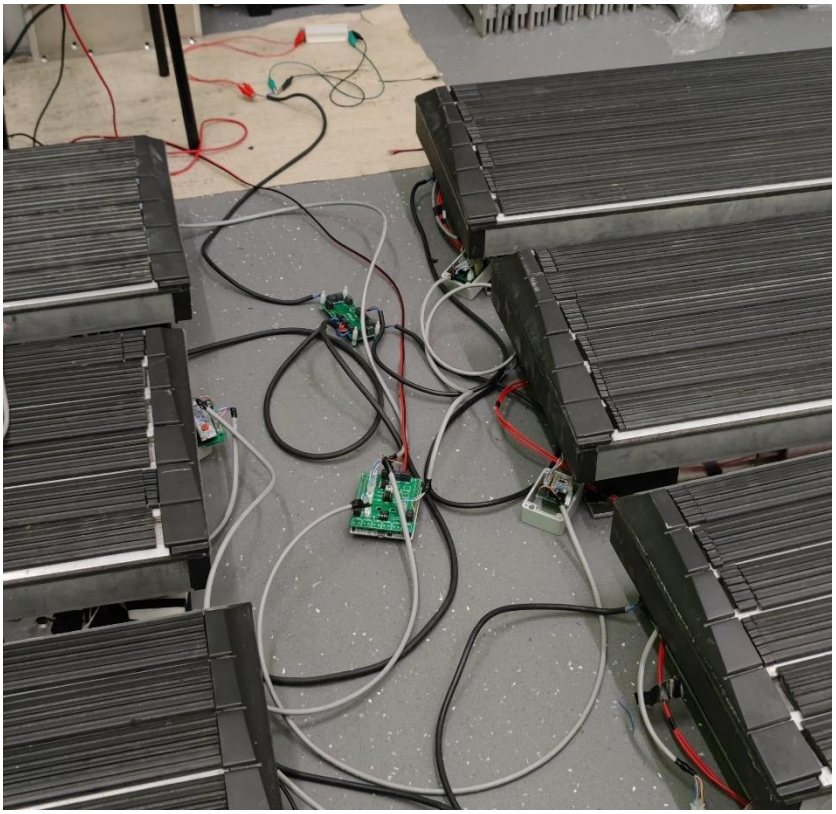


Figura 94 Fotografia da montagem de teste com 4 módulos *NEXT-road*, em laboratório

8.3. TESTES EM AMBIENTE REAL

Finalmente, após terminados todos os testes em ambiente de laboratório, realizaram-se os testes em ambiente real, através da montagem de todo o sistema, em uma estrada em Matosinhos. Com a ajuda de toda a equipa, e após algumas horas, finalizou-se a montagem, e, no dia seguinte, abriu-se a estrada ao trânsito. O sistema montado era constituído por 2 controladores secundários, com 4 controladores periféricos conectados a cada um, ou seja, 8 módulos *NEXT-road*. Na Figura 95, é possível visualizar uma fotografia dos trabalhos de montagem do sistema no pavimento.

Figura 95 Fotografia da montagem do sistema numa estrada em Matosinhos

Estando montado no pavimento, o sistema torna-se apenas acessível através do controlador principal, instalado em um caixa fora da estrada. Por este motivo, os dados produzidos pela passagem dos veículos apenas conseguem ser visualizados neste componente. Sendo o desenvolvimento deste controlador da responsabilidade de vários projetos, os dados provenientes do pavimento foram enviados diretamente para um servidor, e disponibilizados através de um *website*, tal como se verifica pelo painel apresentado na Figura 96.

Figura 96 Dados do sistema armazenados em um servidor

Embora o sistema tenha sido montado e tenham sido obtidos alguns dados, durante os testes surgiram alguns problemas. O primeiro esteve associado ao cabo do barramento primário que interliga os dois controladores secundários. Este danificou-se no momento da montagem, e, por consequência, introduzia ruído no barramento primário, impossibilitando o envio de dados. Assim, o primeiro controlador secundário foi desconectado do resto do sistema, tendo este sido reduzido a 4 controladores periféricos.

O segundo problema foi detetado quando, ao analisar os dados recebidos, o número da contagem de veículos não correspondia ao real. Não foi possível perceber o motivo nem resolver o problema devido ao tempo escasso para o final do prazo de projeto.

9. CONCLUSÃO

O desenvolvimento deste projeto permitiu, não só adquirir e aprofundar os conhecimentos na área da eletrónica digital, projeto de PCB e desenvolvimento de *firmware*, mas também perceber como a evolução dos projetos *IoT*, ao longo dos últimos anos, está a guiar o desenvolvimento das *smart cities*. Assim, será, nesta última secção, realizada uma síntese de todo o projeto: conclusões, relevância do trabalho e possíveis melhoramentos ao sistema desenvolvido.

Após realizado o estudo inicial aos diferentes microcontroladores, sensores e restantes componentes eletrónicos existentes, foi possível aferir que o mercado está preparado para que as cidades inteligentes deixem de ser um tema do futuro, e passem, cada vez mais, a ser um tema do presente. O desenvolvimento deste projeto é a prova disso. Para além de tudo, concluiu-se que a área do controlo de tráfego automóvel é uma das mais problemáticas e que mais tirará proveito desta evolução das cidades.

Em termos do projeto desenvolvido, os requisitos propostos foram todos alcançados, no entanto, reconhece-se que com mais tempo de trabalho, este projeto poderia evoluir exponencialmente. Além do proposto, o sistema foi desenvolvido de maneira a ser modular e escalável, permitindo uma fácil manutenção e troca de componentes, assim como alteração do seu tamanho, sem serem necessárias alterações significativas ao *firmware*. Embora na prática só tenham sido utilizados 8 controladores periféricos, 2 controladores secundários e

1 controlador principal, em teoria, o sistema de transmissão de dados foi desenvolvido para suportar até 3375 controladores periféricos, distribuídos por diferentes controladores secundários e faixas. Os sensores de temperatura e humidade e acelerómetro escolhidos, permitiram cumprir os objetivos propostos. No caso do cálculo da energia gerada, devido a problemas de disponibilidade de material, o sensor de monitorização de energia escolhido teve de ser substituído por um sensor de corrente e um divisor de tensão. Desta forma, foram utilizados recursos do microcontrolador, como *timers*, ADC e tempo de processador, para calcular os valores de energia, que poderiam ser obtidos automaticamente com o uso do sensor *PAC1934*.

Com os testes no terreno, realizados em Matosinhos, provou-se que estes tipos de testes são necessários, num projeto desta dimensão, para perceber que tipo de erros e problemas poderão surgir quando o sistema é implementado em ambiente real, e não exclusivamente em laboratório. Apesar de nos ensaios em tempo real nem todos os objetivos terem sido alcançados, foi possível obter alguns valores, que validaram a lógica do programa e o funcionamento e robustez do sistema de comunicação de dados implementado.

Numa futura nova versão do sistema, podem ser feitos melhoramentos no que toca a *firmware*, de modo que este aceite diferentes tipos de veículos, além de veículos de 4 rodas. Não só, mas também tentar que o sistema não seja totalmente dependente de certos controladores, criando alternativas e sistemas redundantes, para que, no caso de uma falha, o sistema como um todo não deixe de funcionar corretamente. Poderá também ser tido em conta o consumo energético do sistema, através da criação de um modo de *standby*. Por fim, caso haja possibilidade, implementar o sensor *PAC1934* em todos os controladores periféricos e abdicar do sistema de leitura de energia por ADC.

Referências Documentais

[1] H. Ritchie, “Energy - Our World in Data,” *OurWorldInData.org*, 2020. https://ourworldindata.org/urbanization (accessed Feb. 12, 2021).

[2] UNITED NATIONS - DEPARTMENT OF ECONOMIC AND SOCIAL AFFAIRS, “68% of the world population projected to live in urban areas by 2050, says UN | UN DESA | United Nations Department of Economic and Social Affairs,” *United Nations News*. 2018, Accessed: Feb. 11, 2021. [Online]. Available: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html.

[3] C. Harrison and I. A. Donnelly, “A theory of smart cities,” in *55th Annual Meeting of the International Society for the Systems Sciences 2011*, Sep. 2011, pp. 521–535, Accessed: Feb. 23, 2021. [Online]. Available: https://journals.isss.org/index.php/proceedings55th/article/view/1703.

[4] M. Batty *et al.*, “Smart cities of the future,” *Eur. Phys. J. Spec. Top.*, vol. 214, no. 1, pp. 481–518, Dec. 2012, doi: 10.1140/epjst/e2012-01703-3.

[5] European Commission, “Smart Cities / European Commission,” *Digital Agenda for Europe*, 2015. https://ec.europa.eu/info/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities_en (accessed Feb. 24, 2021).

[6] European Commission, “Home | Smart Cities Marketplace,” *Web*. https://smart-cities-marketplace.ec.europa.eu/ (accessed Feb. 24, 2021).

[7] J. Glasco, “Smart Mobility: Challenges and Solutions in Smart Cities,” *Bee Smart City*, 2019. https://hub.beesmart.city/en/solutions/smart-mobility/smart-mobility-challenges-and-solutions-in-smart-cities (accessed Feb. 25, 2021).

[8] Y. Zhang, R. Su, C. Sun, and Y. Zhang, “Modelling and traffic signal control of a heterogeneous traffic network with signalized and non-signalized intersections,” *1st*

Annu. IEEE Conf. Control Technol. Appl. CCTA 2017, vol. 2017-Janua, no. 12, pp. 1581–1586, 2017, doi: 10.1109/CCTA.2017.8062682.

[9] "Traffic control - Road traffic control | Britannica." https://www.britannica.com/technology/traffic-control/Road-traffic-control (accessed Mar. 22, 2021).

[10] P. Balendra, "Vehicle Speed Compliance Statistics, Great Britain: January - June 2020," no. September, p. 7, 2020, Accessed: Sep. 01, 2021. [Online]. Available: https://www.gov.uk/government/statistics/vehicle-speed-compliance-statistics-for-great-britain-january-to-june-2020.

[11] J. M. Breen, P. A. Næss, T. B. Hansen, C. Gaarder, and A. Stray-Pedersen, "Serious motor vehicle collisions involving young drivers on Norwegian roads 2013–2016: Speeding and driver-related errors are the main challenge," *Traffic Inj. Prev.*, vol. 21, no. 6, pp. 382–388, 2020, doi: 10.1080/15389588.2020.1770237.

[12] P. Marwedel, *Embedded system design*. Cham: Springer International Publishing, 2006.

[13] J. Davies, *MSP430 Microcontroller Basics - John H. Davies*. 2008.

[14] Omnisci, "What is Graphical Representation? Definition and FAQs | OmniSci," 2021. https://www.omnisci.com/technical-glossary/embedded-systems (accessed May 31, 2021).

[15] Tutorialspoint, "Embedded Systems - Overview - Tutorialspoint." Accessed: May 31, 2021. [Online]. Available: https://www.tutorialspoint.com/embedded_systems/es_overview.htm.

[16] Geneva, "Microcontrollers (MCU) and Microprocessors (MPU) - STMicroelectronics." https://www.st.com/en/microcontrollers-microprocessors.html (accessed Jun. 01, 2021).

[17] P. Hughes, "The Arm ecosystem ships a record 6.7 billion Arm-based chips – Arm," 2021. https://www.arm.com/company/news/2021/02/arm-ecosystem-ships-record-6-billion-arm-based-chips-in-a-single-quarter (accessed Jun. 01, 2021).

[18] S. Segars, “Enabling Mass IoT connectivity as ARM partners ship 100 billion chips,” *community.arm.com*, 2017, Accessed: Jun. 01, 2021. [Online]. Available: https://community.arm.com/iot/b/internet-of-things/posts/enabling-mass-iot-connectivity-as-arm-partners-ship-100-billion-chips.

[19] D. I. Cutress, “New #1 Supercomputer: Fujitsu’s Fugaku and A64FX take Arm to the Top with 415 PetaFLOPs,” *www.anandtech.com*, 2020, Accessed: Jun. 01, 2021. [Online]. Available: https://www.anandtech.com/show/15869/new-1-supercomputer-fujitsus-fugaku-and-a64fx-take-arm-to-the-top-with-415-petaflops.

[20] “CPU Architecture,” 2017, pp. 7–24.

[21] “Cortex-M – Arm Developer.” https://developer.arm.com/ip-products/processors/cortex-m (accessed Jun. 03, 2021).

[22] “The Apple M1 is the first ARM-based chipset for Macs with the fastest CPU cores and top iGPU - GSMArena.com news,” 2020. https://www.gsmarena.com/the_apple_m1_is_the_first_armbased_chipset_for_macs_with_the_fastest_cpu_cores_and_top_igpu-news-46222.php (accessed Jun. 03, 2021).

[23] C. Pan, “British firm Arm says new chip tech could be licensed to Huawei, potentially easing the telecoms giant’s supply chain woes | South China Morning Post,” 2021. https://www.scmp.com/tech/tech-trends/article/3127782/british-chip-design-firm-arm-takes-aim-intel-biggest-tech-overhaul (accessed Jun. 03, 2021).

[24] “32-Bit Microcontrollers (MCUs) | Microchip Technology.” https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus (accessed Jun. 03, 2021).

[25] Qualcomm, “Qualcomm Application Processors,” Accessed: Jun. 03, 2021. [Online]. Available: https://www.qualcomm.com/products/application-processors.

[26] STMicroelectronics, “STM32 32-bit ARM Cortex MCUs - STMicroelectronics,” 2014. https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html (accessed Jun. 03, 2021).

[27] Monpeco, “real_time_bn/intro_rtos.md at master · monpeco/real_time_bn · GitHub.” https://github.com/monpeco/real_time_bn/blob/master/lab1/intro_rtos.md (accessed Jun. 03, 2021).

[28] Cadence, “Customizable processors that scale from small efficient controllers up to compute-intensive data processing engines.” https://ip.cadence.com/ipportfolio/tensilica-ip/xtensa-customizable (accessed Jun. 14, 2021).

[29] Espressif, “Modules | Espressif Systems,” *Espressif-WikiDevi*, 2018. https://www.espressif.com/en/products/socs (accessed Jun. 14, 2021).

[30] W. Wong, “Flash-based microcontrollers are rapidly taking charge,” *Electron. Des.*, vol. 50, no. 25, pp. 53–56, 2002, Accessed: Jun. 14, 2021. [Online]. Available: https://www.electronicdesign.com/technologies/digital-ics/article/21771889/flashbased-microcontrollers-are-rapidly-taking-charge.

[31] Infineon, “DAVE™ (Version 4)-Introduction,” 2018.

[32] Microchip Technology, “MPLAB X IDE,” 2013. https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-x-ide (accessed Mar. 11, 2021).

[33] STMicroelectronics, “STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics.” https://www.st.com/en/development-tools/stm32cubeide.html (accessed Mar. 11, 2021).

[34] Microchip Technology Inc., “MPLAB Code Configurator | Microchip Technology Inc.” https://www.microchip.com/mplab/mplab-code-configurator (accessed Mar. 12, 2021).

[35] Microchip Technology Inc., “MPLAB® Data Visualizer | Microchip Technology.” https://www.microchip.com/en-us/development-tools-tools-and-software/embedded-software-center/mplab-data-visualizer (accessed Mar. 12, 2021).

[36] Laura Mellon, “Data Transmission - Parallel vs Serial Transmission,” *QUANTIL*, 2019. https://www.quantil.com/content-delivery-insights/content-acceleration/data-transmission/# (accessed Nov. 10, 2020).

[37] US Micro Products, “Understanding Display Interfaces | US Micro Products,” 2019. https://www.usmicroproducts.com/blog/understanding-display-interfaces (accessed Jan. 05, 2021).

[38] “Difference between Synchronous and Asynchronous Transmission - GeeksforGeeks.” https://www.geeksforgeeks.org/difference-between-synchronous-and-asynchronous-transmission/ (accessed Nov. 10, 2020).

[39] Z. W. Hu, “I2C protocol design for reusability,” in *Proceedings - 3rd International Symposium on Information Processing, ISIP 2010*, Oct. 2010, pp. 83–86, doi: 10.1109/ISIP.2010.51.

[40] J. Mankar, C. Darode, K. Trivedi, M. Kanoje, and P. Shahare, “Review of I2C Protocol,” *Int. J. Res. Advent Technol.*, vol. 2, no. 1, pp. 2321–9637, 2014, Accessed: Nov. 10, 2020. [Online]. Available: http://www.ijrat.org.

[41] Hélio Sousa Mendonça, “SPI e I2C,” 2016. https://paginas.fe.up.pt/~hsm/docencia/comp/spi-e-i2c/ (accessed Nov. 10, 2020).

[42] NXP Semiconductors, “I2C-Bus Specification Rev6.” Accessed: Nov. 10, 2020. [Online]. Available: http://www.nxp.com.

[43] Philips Semiconductors, “I2C-Bus Specification Rev2.1,” *Philips Semicond.*, no. January, pp. 1–46, 2000, [Online]. Available: http://www.classic.nxp.com/acrobat_download2/literature/9398/39340011.pdf.

[44] NXP Semiconductors, “I2C-Bus Specification Rev3,” no. April, pp. 1–50, 2007, [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf%0Ahttps://www.nxp.com/docs/ja/user-guide/UM10204.pdf.

[45] NXP Semiconductors, “I2C-Bus Specification Rev4,” no. April, pp. 1–50, 2012, [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf%0Ahttps://www.nxp.com/docs/ja/user-guide/UM10204.pdf.

[46] NXP Semiconductors, “I2C-Bus Specification Rev5,” *Semiconductors*, vol. 3, no. June, pp. 1–50, 2012, [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:I2C-

bus+specification+and+user+manual#0.

[47] D. Astels, “Open Collector,” pp. 1–4, 2014, Accessed: Dec. 02, 2020. [Online]. Available: https://learn.adafruit.com/transistors-101/open-collector.

[48] F. Wikipedia, “Open collector,” pp. 1–4, 2014, Accessed: Oct. 15, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Open_collector#/media/File:OpencollectorV3.png.

[49] “Introduction to I^2C and SPI protocols – Byte Paradigm – Speed up embedded system verification,” *Byte Paradigm*, 2013. https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/ (accessed Dec. 09, 2020).

[50] I. Motorola, “SPI Block Guide,” pp. 1–38, 2003.

[51] Corelis, “SPI Tutorial – Serial Peripheral Interface Bus Protocol Basics,” 2018. https://www.corelis.com/education/tutorials/spi-tutorial/ (accessed Dec. 09, 2020).

[52] Texas Instruments, “TMS320DM36x Digital Media System-on-Chip (DMSoC) Serial Peripheral Interface (SPI) User’s Guide,” 2009.

[53] Maxim Integrated, “Daisy-Chaining SPI Devices,” 2006. Accessed: Dec. 18, 2020. [Online]. Available: https://www.maximintegrated.com/en/design/technical-documents/app-notes/3/3947.html.

[54] E. Peña and M. G. Legaspi, “UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter,” 2020.

[55] Microchip Technology Inc., “TB3216 Getting Started with USART,” 2018.

[56] Sparkfun, “Serial Communication - learn.sparkfun.com,” *https://learn.sparkfun.com*, 2017. https://learn.sparkfun.com/tutorials/serial-communication/all (accessed Mar. 10, 2021).

[57] Hélio Sousa Mendonça, “UART.” https://paginas.fe.up.pt/~hsm/docencia/comp/uart/ (accessed Nov. 10, 2020).

[58] R. Bosch, “CAN Specification Version 2.0,” *Rober Bousch GmbH, Postfach*, vol.

300240, p. 72, 1991, [Online]. Available: http://esd.cs.ucr.edu/webres/can20.pdf.

[59] T. Alferink, “Practical tips: CAN-Bus – KMP Drivetrain Solutions,” *KMP Drivetrain Solutions*, 2017. https://www.kmpdrivetrain.com/paddleshift/practical-tips-can-bus/ (accessed Mar. 01, 2021).

[60] Pico Technology, “CAN & CAN FD - serial protocol decoding - PicoScope from A to Z.” https://www.picotech.com/library/oscilloscopes/can-bus-serial-protocol-decoding (accessed Mar. 02, 2021).

[61] MIKROE, “CAN Bus,” no. 243, pp. 1–16, Jun. 2016, Accessed: Mar. 01, 2021. [Online]. Available: https://www.mikroe.com/blog/can-bus.

[62] CSS Electronics, “CAN Bus Explained - A Simple Intro (2019),” *CCS Electronics*, 2019. https://www.csselectronics.com/screen/page/simple-intro-to-can-bus (accessed Mar. 01, 2021).

[63] Bosch, “CAN with Flexible Data-Rate Specification Version 1.0,” vol. 0, p. 32, 2012.

[64] CSS Electronics, “CAN FD Explained - A Simple Intro (2019),” *CCS Electronics*, 2019. https://www.csselectronics.com/screen/page/can-fd-flexible-data-rate-intro (accessed Mar. 03, 2021).

[65] A. Mutter, R. B. Gmbh, F. Hartwich, and R. B. Gmbh, “Advantages of CAN FD Error detection mechanisms compared to Classical CAN,” no. October, pp. 1–18, 2015.

[66] STMicroelectronics, “STM32G4 Nucleo-32 Board User Manual,” 2019.

[67] Microchip Technology, “MCP2561/2FD - High-Speed CAN Flexible Data Rate Transceiver,” p. 32, 2014, [Online]. Available: http://www.microchip.com/wwwproducts/Devices.aspx?product=MCP2561FD.

[68] Allegro microsystems, “ACS725 Datasheet,” 2020.

[69] Allegro microsystems, “ACS723 Datasheet,” pp. 1–26, 2019.

[70] Microchip Technology Inc., “PAC1931/2/3/4 Datasheet,” pp. 1–64, 2019.

[71] MikroElektronika, “PAC1934 click - board with PAC1934 four channel energy

monitor | MikroElektronika.” https://www.mikroe.com/pac1934-click (accessed Oct. 18, 2021).

[72] STMicroelectronics, “HTS221 - Capacitive digital sensor for relative humidity and temperature,” no. April, pp. 1–31, 2015, [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00116291.pdf.

[73] Microchip Technology Inc., “MCP9808 Datasheet,” *Microchip Technol. Inc.*, vol. 25, pp. 1–54, 2011, [Online]. Available: ww1.microchip.com/downloads/en/DeviceDoc/25095A.pdf.

[74] G. Halliday, E. H. Bigio, N. J. Cairns, M. Neumann, I. R. A. MacKenzie, and D. M. A. Mann, “BME280 datasheet,” *Acta Neuropathol.*, vol. 124, no. 3, pp. 373–382, 2012, [Online]. Available: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280-DS002.pdf%0Ahttps://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf.

[75] Analog Devices, “Digital Accelorometer ADXL345 Data Sheet,” p. 40, 2016, [Online]. Available: www.analog.com.

[76] STMicroelectronics, “LIS3DH Datasheet,” no. May, pp. 1–48, 2015.

[77] STMicroelectronics, “STM32G4 Nucleo-64 Board User Manual,” no. February, pp. 1–44, 2021.

[78] “Altium Designer - PCB Design Software.” https://www.altium.com/altium-designer (accessed Oct. 22, 2021).

[79] Broadcom, “ACHS-7124/7125: Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 3 kVRMS Isolation and Low-Resistance Current Conductor Data Sheet,” 2020.

[80] STMicroelectronics, “Positive voltage regulator ICs,” no. September, pp. 1–55, 2018.

[81] ST, “RM0440 - Reference Manual - STM34G4 Series advanced Arm-based 32-bit MCUs,” no. April, p. 2127, 2020.

[82] “ST Community Question FDCAN Communication.” https://community.st.com/s/question/0D53W00000iEGAOSA4/helloi-am-trying-to-set-up-a-fdcan-communication-std-id-at-a-baud-rate-of-250-kbits-in-stm32g473rbt-the-tx-part-seems-to-work-perfectly-but-i-have-a-problem-at-the-level-of-the-reception (accessed Oct. 25, 2021).

[83] “CAN Bit Time Calculation.” http://www.bittiming.can-wiki.info/#bxCAN (accessed Oct. 25, 2021).

[84] STMicroelectronics, “STM32G0 - DMA.”

Anexo A. Tabelas referentes ao estudo das frequências de aquisição de dados para o Controlador Periférico

VELOCIDADE		TEMPO ENTRE ATIVAÇÃO DAS DUAS TAMPAS		TEMPO ENTRE 2 EIXOS POR UMA TAMPA					
				V1 (1m)		V2 (2m)		V3 (3m)	
KM/h	m/s	s	ms	s	ms	s	ms	s	ms
5	1,389	0,252	252,000	0,720	720,000	1,440	1440,000	2,160	2160,000
10	2,778	0,126	126,000	0,360	360,000	0,720	720,000	1,080	1080,000
15	4,167	0,084	84,000	0,240	240,000	0,480	480,000	0,720	720,000
20	5,556	0,063	63,000	0,180	180,000	0,360	360,000	0,540	540,000
25	6,944	0,050	50,400	0,144	144,000	0,288	288,000	0,432	432,000
30	8,333	0,042	42,000	0,120	120,000	0,240	240,000	0,360	360,000
35	9,722	0,036	36,000	0,103	102,857	0,206	205,714	0,309	308,571
40	11,111	0,032	31,500	0,090	90,000	0,180	180,000	0,270	270,000
45	12,5	0,028	28,000	0,080	80,000	0,160	160,000	0,240	240,000
50	13,889	0,025	25,200	0,072	72,000	0,144	144,000	0,216	216,000
55	15,278	0,023	22,909	0,065	65,455	0,131	130,909	0,196	196,364
60	16,667	0,021	21,000	0,060	60,000	0,120	120,000	0,180	180,000
65	18,056	0,019	19,385	0,055	55,385	0,111	110,769	0,166	166,154
70	19,444	0,018	18,000	0,051	51,429	0,103	102,857	0,154	154,286
75	20,833	0,017	16,800	0,048	48,000	0,096	96,000	0,144	144,000
80	22,222	0,016	15,750	0,045	45,000	0,090	90,000	0,135	135,000
85	23,611	0,015	14,824	0,042	42,353	0,085	84,706	0,127	127,059
90	25,000	0,014	14,000	0,040	40,000	0,080	80,000	0,120	120,000
95	26,389	0,013	13,263	0,038	37,895	0,076	75,789	0,114	113,684
100	27,778	0,013	12,600	0,036	36,000	0,072	72,000	0,108	108,000
105	29,167	0,012	12,000	0,034	34,286	0,069	68,571	0,103	102,857
110	30,556	0,011	11,455	0,033	32,727	0,065	65,455	0,098	98,182

Cálculos para V1 a 80 km/h:

Nº de leituras	**Frequência de aquisição**	
	Hz	kHz
1	63,49206349	
10	634,9206349	
100	6349,206349	6,349206349
1000	63492,06349	63,49206349
10000	634920,6349	634,9206349

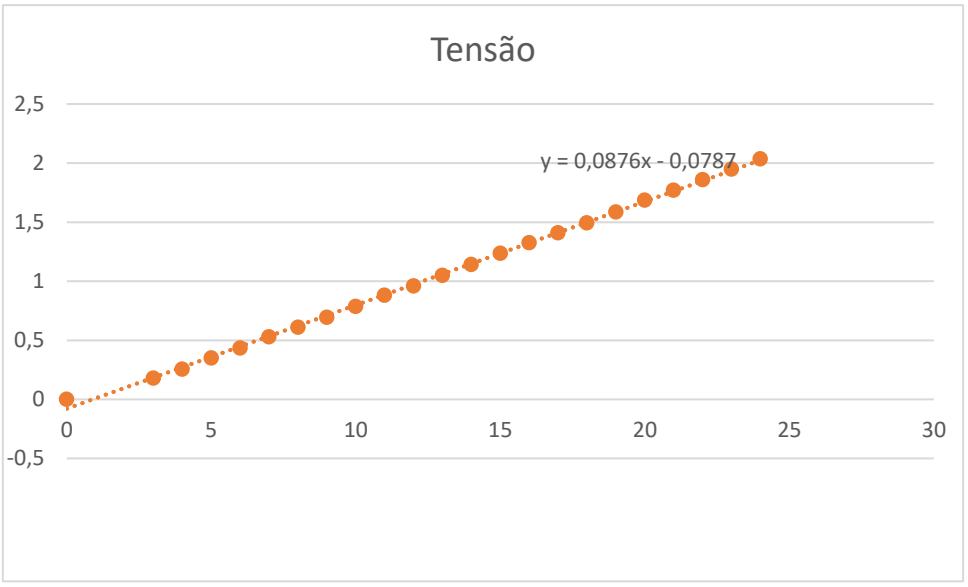
Anexo B. Tabela Comparativa de Microcontroladores

Nome	Marca	CPU	Freq. (MHz)	Flash ROM (KB)	SRAM (KB)	GPIO	USART	SPI	I2C	CAN	ADC
ESP32-S3	Espressif	Xtensa LX7	240	384	512	44	3	4	2	N/A	2x12-bit
ESP32-S2	Espressif	Xtensa LX7	240	128	320	43	2	3	1	N/A	2x12-bit
ESP32-S2F	Espressif	Xtensa LX8	240	128	320	43	2	3	1	N/A	2x12-bit
ESP32-C3	Espressif	RISC-V	160	384	400	22	2	3	1	N/A	2x12-bit
STM32F302	ST	Cortex-M4	72	32 ~ 512	16 ~ 40	-	2 ~ 3	2 ~ 3	2 ~ 3	1	1/2x12-bit
STM32F303	ST	Cortex-M4	72	32 ~ 512	16 ~ 80	37 ~ 86	2 ~ 3	1 ~ 4	1 ~ 3	1	2/4x12-bit
STM32G4	ST	Cortex-M4	170	32 ~ 512	~ 32	-	Yes	Yes	Yes	Can-Fd	3/5x12-bit
STM32G431KB	ST	Cortex-M4	170	128	32	-	3	3	3	1	2x12-bit
STM32F4X	ST	Cortex-M4	180	64 ~ 2056	32 ~ 384	-	Yes	Yes	Yes	~ 2	12-bit
STMF4F7X	ST	Cortex-M7	216	64 ~ 2048	256 ~ 512	-	Yes	Yes	Yes	~ 3	3x12-bit
STM32L4	ST	Cortex-M4	80	128 ~ 1024	40 ~ 320	-	Yes	Yes	Yes	2	2x12-bit
STM32L4+	ST	Cortex-M4	120	512 ~ 2048	320 ~ 640	-	Yes	Yes	Yes	1	1/2x12-bit
XMC4100	Infineon	Cortex-M4	80	64 ~ 128	20	30 ~ 45	~ 4	~ 4	~ 4	2	4x12-bit
XMC4100	Infineon	Cortex-M4	80	256	40	30 ~ 45	~ 4	~ 4	~ 4	2	4x12-bit
XMC4300	Infineon	Cortex-M4	144	256	128	75	~ 4	~ 4	~ 4	2	4x12-bit
XMC4400	Infineon	Cortex-M4	120	256 ~ 512	80	41 ~ 75	~ 4	~ 4	~ 4	4	4x12-bit
XMC4500	Infineon	Cortex-M4	120	512 ~ 1024	128 ~ 160	55 ~ 91	~ 6	~ 6	~ 6	3	4x12-bit
XMC4700	Infineon	Cortex-M4	144	1536 ~ 2048	276 ~ 352	75 ~ 155	~ 6	~ 6	~ 6	6	4x12-bit
SAM4E	Microchip	Cortex-M4	120	~ 1024	~ 128	79 ~ 117	Yes	Yes	N/A	2	2x12-bit
SAME54	Microchip	Cortex-M4F	120	1024	256	48 ~ 128	~ 8	~ 8	~ 8	2	2x12-bit
SAMG55	Microchip	Cortex-M4	120	512	176	48	~ 8	~ 8	~ 8	N/A	8x12-bit
SAMD5X	Microchip	Cortex-M4F	120	~ 1024	~ 256	48	~ 8	~ 8	~ 8	N/A	2x12-bit
NUCLEO-F7429ZI	ST	Cortex-M4	180	2048	256	114	4	6	3	2	3x12-bit
NUCLEO-F746ZG	ST	Cortex-M7	216	1024	320	114	4	6	4	2	3x12-bit
NUCLEO-L496ZG-P	ST	Cortex-M4	80	1024	320	136	5	3	4	2	3x16-bit
NUCLEO-L4R5ZI-P	ST	Cortex-M4	120	2048	640	110	3	3	4	1	1x12-bit
NUCLEO-G431RB	ST	Cortex-M4	170	128	32	52	3	3	3	1xCAN-FD	2x12-bit
NUCLEO-G474RE	ST	Cortex-M4	170	512	128	52	3	2	4	3xCAN-FD	5x12-bit
NUCLEO-G491RE	ST	Cortex-M4	170	512	112	52	3	3	3	2xCAN-FD	1x12-bit

Nome	Timers	Size	Preço	Extras	Software
ESP32-S3	-	QFN56 (7*7)	-	Wi-Fi; Bluetooth	Eclipse IDF
ESP32-S2	-	QFN56 (7*7)	desde 0,839€	-	Eclipse IDF
ESP32-S2F	-	QFN56 (7*7)	desde 1,08€	-	Eclipse IDF
ESP32-C3	-	QFN32(5*5)	-	Wi-Fi; Bluetooth	Eclipse IDF
STM32F302	1x32-bit / 7x16-bit	QFN32 ~ LQFP144	desde 3,62€	Watchdog; RTC; DAC 12-bit	STM32Cube
STM32F303	1x32-bit / 10x16-bit	QFN32 ~ LQFP144	desde 3,83€	DAC 12-bit	STM32Cube
STM32G4	Yes	-			STM32Cube
STM32G431KB	1x32-bit / 7x16-bit	-	desde 4,19€	FPU e DPS; DAC 12-bit; Low-Power RTC; 1x16-bit Low-Power Timer	STM32Cube
STM32F4X	16-bit / 32-bit	-			STM32Cube
STMF4F7X	16-bit / 32-bit	-			STM32Cube
STM32L4	Yes	-		Low-Power, FPU, DSP	STM32Cube
STM32L4+	Yes	-		Low-Power, FPU, DSP	STM32Cube
XMC4100	-	VQFN48/LQFP64	desde 2,63€	LIN	DAVE 4
XMC4100	-	VQFN48/LQFP64	desde 2,72€	LIN	DAVE 4
XMC4300	-	LQFP100	desde 8,38€	LIN; EtherCAT	DAVE 4
XMC4400	-	LQFP64/LQFP100	desde 4,05€	LIN; Ethernet 10/100	DAVE 4
XMC4500	-	LQFP144;LQFP100;LFBGA144	desde 5,79€	LIN; Ethernet 10/100; Watchdog Timer; FPU	DAVE 4
XMC4700	-	LQFP144;LQFP100;LFBGA196	desde 7,47€	LIN; Ethernet 10/100	DAVE 4
SAM4E	-		desde 3,20€	Ethernet 10/100	Microchip Studio
SAME54	-		desde 4,42€	E51 2xCAN.FD / E53 Ethernet / E54 2xCAN+Ethernet	Microchip Studio
SAMG55	6x16-bit		desde 3,21€		Microchip Studio
SAMD5X	-		desde 2,77€		Microchip Studio
NUCLEO-F7429ZI	12x16-bit/2x32-bit		24,47$	Dual Watchdog, RTC, Systick, FPU	STM32Cube
NUCLEO-F746ZG	12x16-bit/2x32-bit		24,47$	Dual Watchdog, RTC, Systick, FPU	STM32Cube
NUCLEO-L496ZG-P	7x16-bit/2x32-bit		21,28$		STM32Cube
NUCLEO-L4R5ZI-P	11x16-bit/2x32-bit		21,28$		STM32Cube
NUCLEO-G431RB	10x16-bit/1x32-bit		15,96$		STM32Cube
NUCLEO-G474RE	11x16-bit/2x32-bit		15,96$		STM32Cube
NUCLEO-G491RE	11x16-bit/1x32-bit		15,96$		STM32Cube

Anexo C. Conversão dos valores de tensão e corrente no ADC

0	0
3	0,18
4	0,254
5	0,35
6	0,435
7	0,53
8	0,61
9	0,695
10	0,785
11	0,88
12	0,96
13	1,05
14	1,14
15	1,235
16	1,325
17	1,41
18	1,495
19	1,585
20	1,685
21	1,77
22	1,86
23	1,95
24	2,035



0	1,59
0,2	1,622
0,4	1,642
0,45	1,648
0,5	1,656
0,6	1,671
0,7	1,681
0,8	1,699
0,9	1,707
1	1,719