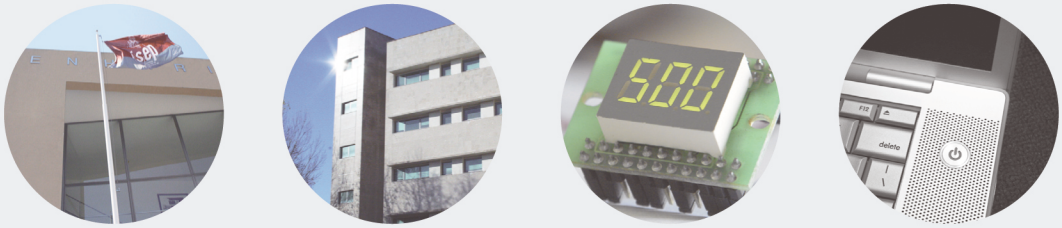
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Rede de sensores Bluetooth de baixo Consumo Energético com Interface CAN

FLÁVIO LOPES DE VASCONCELOS
novembro de 2017

POLITÉCNICO
DO PORTO

Rede de sensores Bluetooth de baixo Consumo Energético com Interface CAN

Flávio Lopes de Vasconcelos

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2017

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, ramo Automação e Sistemas

Candidato: Flávio Lopes de Vasconcelos, Nº 1100419, 1100419@isep.ipp.pt

Orientação científica: Lino Figueiredo , lbf@isep.ipp.pt

Coorientador: João Ferreira , joao.carlos.ferreira@iscte-iul.pt

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

11 de Novembro de 2017

Agradecimentos

Desejo expressar os meus agradecimentos ao meu orientador Lino Figueiredo pela sua orientação e conselhos preciosos e pelas enriquecedoras discussões mantidas. Não deixando o coorientador João Ferreira de fora, quero agradecer o seu envolvimento nesta tese, quer pela discussão, quer pela ajuda prestada e até mesmo pela possibilidade de levar esta ideia até Itália.

Por fim, desejo expressar os meus maiores agradecimentos aos meus pais, pelo apoio e paciência que tiveram comigo. Agradeço sobretudo por me proporcionarem a oportunidade de estudar. Agradeço também à minha namorada, pela compreensão transmitida nos momentos mais complicados.

A todos um bem haja por permitirem que esta tese seja uma realidade.

Flávio Lopes Vasconcelos

Esta página foi intencionalmente deixada em branco.

Resumo

Este projeto tem como objetivo desenvolver um sistema que permita monitorizar a viagem de um veiculo no transporte de mercadorias, permitindo assim monitorizar, por exemplo, da temperatura ou da velocidade do veiculo.

Para isso realizou-se uma pesquisa sobre as várias soluções, nomeadamente redes de sensores sem fios, tecnologias *low power* existentes, com a finalidade de cumprir os objetivos do projeto. A escolha caiu sobre uma rede de sensores sem fios, utilizando a tecnologia *Bluetooth Low Energy* associando o sistema ao conceito *Internet of Things*.

O sistema é composto por 5 *Slaves*, 1 *Master* e uma Unidade Central. Cada um dos *Slaves* contém um sensor que mede uma grandeza diferente, como por exemplo, temperatura, aceleração, pressão atmosférica, entre outros. O *Master* recolhe essa informação toda e reencaminha para a Unidade Central. A Unidade Central está conectada ao veículo por CAN onde é possível obter a velocidade e as rotações do motor, esta contém também um sensor GPS e uma ligação serie para que os dados possam ser enviados para o servidor. Os *Slaves* e o *Master* tem o SoC nrf51822 como microcontrolador e radio *bluetooth*. Por sua vez, a Unidade Central dispõe o STM32f103 como unidade de processamento. Ambos os microcontroladores são de arquitetura ARM.

Palavras-chave

Rede de Sensores, *Bluetooth Low Energy*, *Master*, *Slave*, GPS, ARM, CAN, IoT, nrf51822, stm32f103

Esta página foi intencionalmente deixada em branco.

Abstract

This project aims to develop a system that allows to monitor the trip of a vehicle in the transport of goods, thus allowing control variables like temperature and speed in goods transport vehicle trips.

There were thought into consideration many solutions to accomplish the objectives of this project, namely wireless sensor networks, existing technologies, in order to fulfill the objectives of the project. The choice fell on a wireless sensor network, using technology Bluetooth Low Energy associating the system with the concept Internet of Things.

The system has 5 slaves, 1 master and a 1 central unit. Each of the slaves contains a sensor which measures a different magnitude, such as temperature, acceleration, atmospheric pressure, and more. The master collects this whole information and forwards it to the central unit. The central unit is connected to the vehicle by CAN-BUS where it is possible to obtain vehicle speed and RPM of the engine, the central unit also contains a GPS and a serial link so that the data can be sent to the server. The slaves and the master have the SoC nrf51822 as microcontroller and bluetooth radio. On the other hand the central unit has a stm32f103. Both microcontrollers are of ARM architecture.

Keywords

Wireless Sensor Network, Bluetooth Low Energy, master, slave, GPS, ARM, CAN, IoT, nrf51822, stm32f103

Esta página foi intencionalmente deixada em branco.

Conteúdo

Lista de Figuras

Esta página foi intencionalmente deixada em branco.

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Excertos de Código

Esta página foi intencionalmente deixada em branco.

Lista de Acrónimos

ADC *Analog to Digital Converter*

API *Application Programming Interface*

BLE *Bluetooth Low Energy*

BPSK *Binary Phase Shift Keying*

CAN *Controller Area Network*

CMG *Course Made Good*

CRC *Cyclic Redundancy Check*

CSRK *Connection Signature Resolving Key*

DMP *Digital Motion Processor*

EDIV *Encrypted Diversifier*

FIFO *First In, First Out*

GFSK *Gaussian Frequency Shift Keying*

GPIO *General Purpose Input/Output*

GPRS *General Packet Radio Service*

GPS *Global Positioning System*

I2C *Inter-Integrated Circuit*

IEEE *Institute of Electrical and Electronics Engineers*

IoT *Internet of Things*

IRK *Ientity Resolution Key*

ISM *Industrial Scientific and Medical*

LCD *Liquid Crystal Display*

ISO *International Organization for Standardization*

IIoT *Industrial Internet of Things*

IoT *Internet of Things*

LDR *Light Dependent Resistor*

LED *Light Emitting Diode*

LGA *Land Grid Array*

LiPo *Lithium-ion Polymer Battery*

LTK *Long Term Key*

MEEC *Mestrado em Engenharia Eletrotécnica e de Computadores*

MEMS *Micro Electro Mecanical System*

MIC *Message Integrity Check*

NFC *Near Field Communication*

OBD-II *On-Board Diagnostic*

OQPSK *Offset Quadrature Phase Shift Keying*

OSI *Open Systems Interconnection*

OTA *Over the air*

PCB *Printed circuit board*

PDU *Protocol Data Unit*

PWM *Pulse Width Modulation*

RAM *Random Access Memory*

Rand *Random Number*

RFID *Radio-Frequency IDentification*

RFU *Reserved Future Use*

RPM Rotações Por Minuto

RSSF Rede de Sensores Sem Fios

SoC *System On Chip*

SMD *Surface Mounted Components*

SPI *Serial Peripheral Interface*

SRAM *Static Random Access Memory*

STK *Short-Term Key*

TDMA *Time Division Multiple Access*

TEDI Unidade Curricular Tese/Dissertação

UART *Universal Asynchronous Receiver/Transmiter*

USART *Universal Synchronous Asynchronous Receiver/Transmiter*

USB *Universal Serial Bus*

UUID *Universally Unique Identifier*

VHF *Very High Frequency*

Esta página foi intencionalmente deixada em branco.

Capítulo 1

Introdução

Os avanços nas áreas de micro processamento, *Micro Electro Mecanical System* (MEMS) e comunicação sem fios, estimulam o desenvolvimento de equipamentos sensoriais que podem ser utilizados em diversas áreas de aplicação.

Representando uma subclasse das redes *ad hoc* sem fios, as Rede de Sensores Sem Fios (RSSF) são consideradas a nova geração dos sistemas embebidos de tempo real com recursos computacionais, energia e memória limitados. Estas redes podem apresentar grande número de nós sensoriais sem fios (também conhecido por *motes* [1]) e têm como objetivo recolher e transmitir dados.

Normalmente os nós sensoriais têm pouco poder de processamento de modo a aumentar a sua autonomia. O facto de serem sem fios impõe a utilização de baterias para a alimentação. Vários fatores são culminantes para o desgaste da bateria dos nós, sendo o módulo de rádio um dos principais consumidores de energia dos nós sensoriais no processo de transmissão de dados para a rede. Um dos maiores desafios nas RSSF passa por criar mecanismos de cooperação entre os nós da rede de forma a permitir a esta tenha capacidade de resolver problemas de forma eficiente.

Este tipo de solução está muito em voga em vários cenários devido a ser uma solução de baixo custo para diversas aplicações. Outros fatores que

favorecem a utilização de RSSF são a auto-organização, mobilidade dos nós no espaço geográfico, operação autónoma, capacidade de suportar as más condições ambientais, escalabilidade, facilidade de uso, entre outros [2].

1.1 Enquadramento e Motivação

A implementação de sistemas computacionais confiáveis, tolerantes a falhas e robustos é difícil devido a fatores externos que não são possíveis controlar ou prever.

Com o avanço da tecnologia existe necessidade da criação de sistemas que possam monitorizar uma grande área, mas com um custo relativamente baixo. Para colmatar essa lacuna, o desenvolvimento da microeletrónica proporciona o desenvolvimento de componentes com dimensões cada vez mais reduzidas, preços menores, maior capacidade funcional e com consumo de energia inferior. Essa tendência possibilita o desenvolvimento de aplicações mais complexas e viáveis economicamente [3].

1.2 Cenários de aplicação

As RSSF, são adequadas para situações onde não se pode implementar uma solução com fios e onde se pretende um acesso instantâneo à informação. Os nós sensoriais podem mover-se juntamente com o fenómeno observado. Como exemplo disso temos os sensores que são colocados em animais para observar o seu comportamento.

As redes de sensores podem ser utilizadas nas mais diferentes áreas [4]:

- Na área militar onde por exemplo podem detetar movimentos dos inimigos, a presença de materiais perigosos ou minas escondidas;
- Na medicina/biologia onde podem analisar o funcionamento dos diversos órgãos existentes no corpo humano e detetar a presença de

substâncias que poderão causar problemas ao organismo;

- Na área do turismo, da educação, na descoberta de desastres ecológicos;
- Na industria de produção (monitorização de fluxos, pressão, temperatura, etc.).

1.3 Objetivos

O principal objetivo passa pela monitorização dos transportes de mercadorias. Hoje em dia existem sistemas de localização para frotas. Este sistema tem como objetivo ser mais que isso, além de conter GPS este sistema contém vários sensores que permitem monitorizar a carga e o veiculo.

A implementação de uma rede local sem fios deve ser levada em conta de forma e evitar a passagem de cabos. A solução deverá ter um consumo baixo de energia e um custo igualmente baixo.

A solução terá que incorporar sensores numa viatura de modo a recolher dados sobre a viagem, sensores inerciais, sensores de temperatura, entre outros. Terá que ser uma solução de baixo consumo energético a qual permita transmitir, sem fios, a informação para um sistema central próximo.

O sistema deverá possuir conectividade com o veiculo, para tal deve ser utilizado o protocolo CAN. Agregado a isto tudo deve também possuir a capacidade de monitorização à distancia. Uma vez que o sistema está dentro do veículo , este deverá disponibilizar os dados numa *cloud* por forma a serem consultados de forma remota.

A isto soma-se o conceito de *Internet of Things* (IoT), ou seja, o sistema deve ofertar a intercomunicação entre si, trocando informações sobre determinadas grandezas, localização, estado de variáveis relevantes, entre outros, de forma a gerar uma grande quantidade de dados armazenados em servidores.

Existe outros objetivos que não devem ser esquecidos, dos quais se destacam:

- O sistema deve ser flexível o suficiente para que a adição ou remoção de algum elemento seja sem intervenção do utilizador;
- O sistema deve possuir um GPS de forma a guardar o percurso do veículo;
- Os sensores devem ser precisos o suficiente de forma a não comprometer o sistema;
- A rede deverá ter no mínimo 3 nós de forma a evidenciar as vantagens deste tipo de rede sem fios.

1.4 Calendarização

Na figura 1.1 está apresentado a distribuição das tarefas realizadas ao longo destes dois anos de trabalho neste projeto.

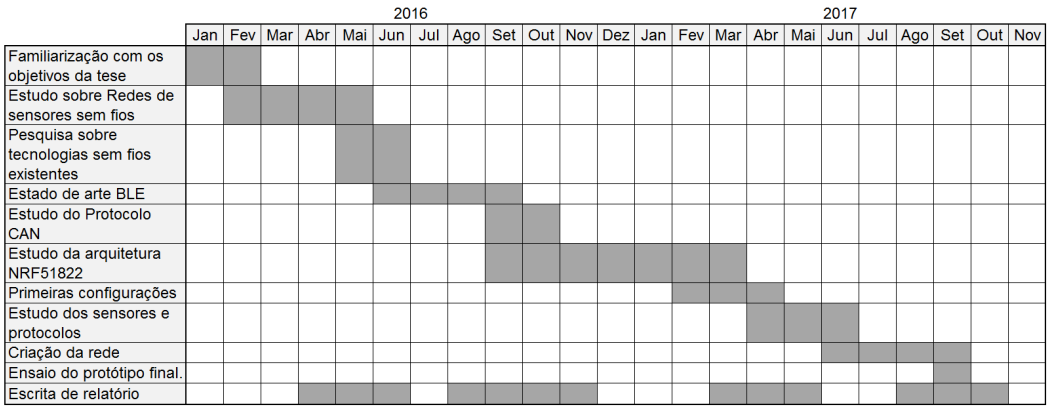


Figura 1.1: Calendarização

1.5 Estrutura do relatório

O presente relatório está dividido em 8 capítulos. No primeiro capitulo é apresentado de forma introdutória o tema do trabalho. No segundo capitulo é apresentado o estado da arte de redes de sensores sem fios onde é abordado as principais características bem como os fatores que determinam uma rede de sensores sem fios, é apresentado algumas aplicações e por fim é explicado o conceito IoT. No terceiro capitulo é apresentado a tecnologias existentes referente às redes de sensores sem fios, como o *Bluetooth Low Energy*, o ZigBee, Z-Wave e ANT. Neste capitulo é ainda apresentado os motes com mais conhecidos. O quarto capítulo é dedicado ao Bluetooth, onde é explicado a diferença entre o *Bluetooth* classico e o *Bluetooth Low Energy*. É apresentado a a arquitetura da *stack* do *Bluetooth Low Energy* bem como os pacotes que são transmitidos. O quinto capitulo retrata as redes de sensores com fios utilizadas neste projeto, como o I2C, o OneWire e o protocolo CAN. No sexto capitulo é apresentado a arquitetura e o projeto do sistema, primeiramente é feito uma analise de requisitos, posteriormente é descrito a arquitetura do sistema e por ultimo é feito a escolha da eletrónica a utilizar neste projeto. No sétimo capitulo é descrita a implementação tanto em *hardware* como em *software*. O oitavo capitulo tem como objetivo expor alguns testes e resultados do sistema implementado. Por fim, o nono capitulo é feito a conclusão do projeto e indicado alguns aspetos a melhorar.

Esta página foi intencionalmente deixada em branco.

Capítulo 2

Rede de Sensores sem Fios

Uma rede de sensores caracteriza-se pela capacidade de monitorizar uma ou mais variáreis de interesse num determinado evento, tais como: distancia, direção, velocidade, humidade, velocidade do vento, temperatura, movimento, vibração, intensidade luminosa, atividade sísmica, som, peso, pressão, dentre outras [5]. Seguidamente é apresentado algumas características de uma RSSF bem como alguns fatores que determinam ou influenciam a implementação de uma RSSF, algumas aplicações e alguns casos práticos.

2.1 Características

As RSSF são bastante abrangentes e dinâmicas. A flexibilidade de instalação e configuração dessas redes fazem que sua utilização apresente resultados bastante satisfatórios. Este elevado nível de flexibilidade necessita de mecanismos com grande capacidade de adaptação. De modo a colmatar é utilizado a arquitetura de *clusters* [4]. *Cluster* por definição na área computacional, é um sistema que consiste num conjunto de computadores ou de unidades de processamento autónomos interligados entre si de forma a trabalhar juntos para uma finalidade [6]. Os *clusters* são ideais para or-

ganizar grandes quantidades de elementos que estão localizados numa área extensa, proporcionando assim uma excelente flexibilidade. Normalmente numa RSSF existem 3 tipos de elementos [4]:

- Escravos
- Mestres
- *Sink*

Os elementos ou nós, escravos (*slave*) tem como objetivo fazer a recolha de dados provindos dos sensores e a transmissão desses dados para a rede. Poderá haver momentos que apenas fazem a aquisição dos dados dos sensores de modo a criar algum tipo de historial e só passado um determinado tempo é que a transmissão será feita para a rede.

Os nós mestres ou *master* tem a função de monitorizar todos os nós escravos existentes. A grande característica destes elementos é a capacidade de cruzamento de dados provindos dos vários sensores. Estes nós podem receber uma quantidade massiva de informações provinda dos seus escravos o que originará uma quebra significativa no desempenho da rede. Desta forma implica que o cruzamento de dados terá de ser feito de forma otimizada [7].

O nó *sink* é um elemento que faz a ligação de uma outra rede com a RSSF. Deste modo, o *sink* é um elemento de grande importância pois caso este nó pare, toda a rede fica incomunicável com o exterior [4].

2.2 Fatores para escolha de um RSSF

O projeto de uma rede de sensores é influenciado por vários fatores [4]:

- Tolerância a falhas;
- Escalabilidade;
- Custos de produção;

- Ambiente operacional;
- Tipologia da rede;
- Restrições de *hardware*;
- Meios de Transmissão;
- Consumo de energia;
- Interferência.

2.2.1 Tolerância a falhas

Alguns nós da RSSF podem falhar, quer por falta de energia, quer por algum tipo de dano físico ou simplesmente devido ao ruído no meio envolvente. Essas falhas não podem de maneira alguma afetar o funcionamento da rede em geral. A rede terá que ser suficientemente robusta de modo a manter o seu funcionamento, para isso poderá ser necessário a reorganização da rede ou a até mesmo mudar o esquema de endereçamento de mensagens [8] [9].

2.2.2 Escalabilidade

O número de nós presentes nestas redes, dependendo da aplicação, podem ser centenas ou mesmo milhares. A rede tem de ser capaz de operar com esse número de nós. Os esquemas adotados devem ser flexíveis o suficiente para serem capazes de suportar grandes escalabilidades de nós [4].

2.2.3 Custos de produção

A partir do momento que uma RSSF é composta por vários nós sensoriais, o custo de cada nó é muito importante para justificar o preço de uma RSSF. O custo de cada nó tem de ser o mínimo possível. Atualmente a tecnologia disponível já permite ter um sistema de radio eficiente a preço baixo. Cada nó sensorial tem outros periféricos alem do sistema radio, dependendo da

aplicação este pode ter um sistema de localização. O custo total de cada nó é uma questão desafiadora, dada a quantidade de funcionalidades a um preço muito reduzido.

2.2.4 Ambiente operacional

Os nós sensoriais podem estar localizados de forma mais densa ou mais dispersa. Geralmente as RSSF são aplicadas em ambientes de difícil acesso. Exemplos onde as RSSF são aplicadas [4]:

- Interior de grandes maquinas;
- Áreas com contaminação biológica ou química;
- Superfície e no fundo do oceano;
- Interior de casas;
- Veículos;
- Campo de batalha;
- Animais;
- Aviação.

Esta pequena lista dá a ideia das condições adversas que este tipo de redes podem estar sujeitas. Estas podem funcionar em ambientes de alta pressão como é o caso do fundo do oceano. Ou ainda em ambientes com diversas temperaturas e níveis de ruído.

2.2.5 Topologia da rede

Com um número tão elevado de nós sensoriais, que devem funcionar sem intervenção e sujeitos a falhas frequentes, a manutenção da topologia da rede é algo fundamental para o seu funcionamento. O facto da densidade dos nós

numa RSSF ser variável, dependendo da aplicação, requer um tratamento cuidadoso na manutenção da topologia.

Numa primeira fase, os nós sensoriais terão de ser colocados no ambiente mas independentemente do método, deve-se sempre procurar reduzir o custo de instalação, incrementar a flexibilidade da reorganização da rede e promover tolerância a falhas [4].

Após a rede já estar no meio, a topologia da rede pode sofrer alterações devido a uma serie de fatores[4]:

- Posição;
- Acessibilidade (ruído, obstáculos, etc.);
- Energia disponível;
- Mau funcionamento;
- Requisitos das tarefas.

Os nós sensoriais podem ser estaticamente implantados. No entanto, a falha do nó é regular devido à falta de energia. Também é possível ter nós sensoriais completamente moveis mas ambos têm que ter a capacidade de adaptação.

Depois da rede já estar devidamente implantada, esta pode ainda assim, sofrer a necessidade de ter algumas alterações como a adição de mais nós à rede ou simplesmente fazer alguma substituição a algum nó inoperacional. A adição de novos nós representa uma necessidade de reorganizar a rede. Além disso, estes podem ser um alvo de algum tipo interferência. Portanto, a topologia da rede está sujeita mudanças frequentes após a implantação.

2.2.6 Restrições de *Hardware*

Um nó sensorial é feito por quatro componentes básicos, como mostra a figura 2.1, uma unidade de sensoriamento, unidade de processamento, modulo

de conectividade sem fios e a unidade de energia.

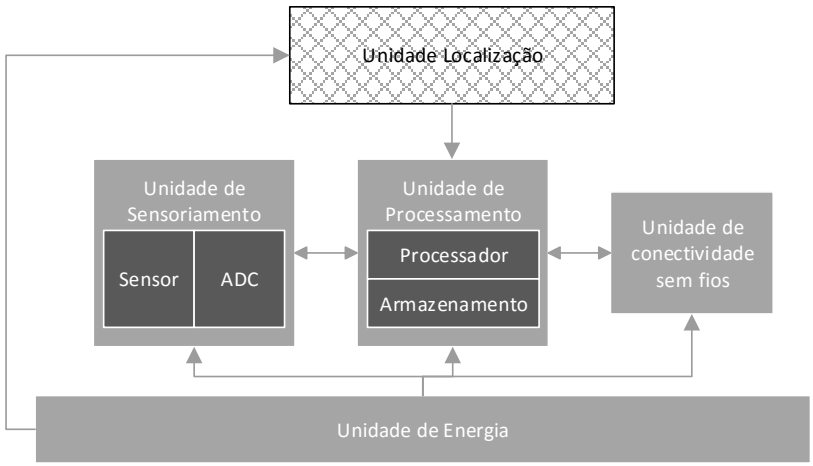


Figura 2.1: Componentes de um nó sensorial (adaptado de [4])

Na unidade de sensoriamento pode ou não ser necessário a utilização de algum tipo de conversor ou algum tipo de condicionamento de sinal de modo que a unidade de processamento possa interpretar esses sinais provindos da unidade de sensoriamento.

A unidade de processamento pode ter uma sub-unidade de armazenamento para que os dados processados possam ser armazenados até à próxima transmissão.

O objetivo da unidade de conectividade sem fios é conectar o nó à rede e tem como tarefa enviar os dados para a rede.

Por ultimo e não menos importante, a unidade de energia que gera energia para alimentar todos os outros elementos do nó sensorial.

Na maioria das técnicas de encaminhamento de mensagens numa RSSF é necessário o conhecimento da localização por forma a aumentar a eficiência na troca de mensagens, para isso é comum um nó sensorial ter uma unidade de localização [4].

Uma vez que os nós sensoriais estão muitas vezes inacessíveis, o tempo

de vida de uma rede de sensores depende dos recursos de energia dos nós. A energia é também um recurso escasso devido às limitações de tamanho [10].

O modulo de conectividade sem fios do nó sensorial é também dos mais importantes, pois é este que envia a informação para a rede. A transmissão de dados pode ser realizada por meio ótico, o que torna o nó mais económico, consome menos energia que a rádio frequência mas requer que os nós sensoriais estejam alinhados de determinada maneira de modo que a comunicação esteja estabelecida [11]. Este cenário é extremamente difícil cumprir numa RSSF. É usual utilizar rádio frequência numa RSSF, mas existem requisitos, modulação, filtragem, desmodulação [12], mas apesar disso ainda tornam esta solução mais flexível face à transmissão ótica. Nas RSSF atuais é utilizada a norma IEEE 802.15.4, ou seja, a transmissão pode ser feita a 2.4 GHz, 915 MHz ou 868 MHz oferecendo uma largura de banda de 250 kbps 40 kbps e 20 kbps, respetivamente [13].

Relativamente ao processamento associa-se um nó sensorial a uma unidade fraca em termos de capacidade de processamento, mesmo apesar de hoje em dia, a tecnologia proporcionar microcontroladores mais poderosos e cada vez mais pequenos. Um exemplo disso é o *Mica* desenvolvido pela Universidade da Califórnia em Berkeley [14], este conta com um microcontrolador de 8 bits da Atmel *clock* de 4Mhz, 128KB de memoria *flash*, 4KB de *Random Access Memory* (RAM). Quanto ao sistema operativo, este nó sensorial utiliza o *TinyOS*, também desenvolvido pela Universidade da Califórnia em Berkeley [14]. Mas existem outros *motes* alem deste, de forma a expor outras alternativas é apresentado tabela 2.1.

Pode-se ver que existe uma vasta gama de *motes* disponíveis, com uma vasta gama de microcontroladores que vão desde a arquitetura 8051 como é o caso do RF Mote e vai até aos de arquitetura ARM como é o caso do .NOW. Em termos de radio alguns utilizam a tecnologia *bluetooth* outros *ZigBee* como é o caso do .NOW.

Tabela 2.1: Comparativo dos vários *motes* existentes

Nome	Microcontrolador	Clock (Mhz)	Flash (KiB)	RAM (KiB)	Armazenamento (KiB)	Radio	Ano
RF Mote [15]	AT9080515	0.15	8	0.5	32	TR1000	
Rene [16]	ATMega163	4	16	1	32	TR1000	
BTnode rev3 [17]	ATmega128	16 [18]	128	256	32	ZV4002 e CC1000	2004
iMote2 [19]	Intel PXA271	13 – 416	32000	256		CC2420	2005
Telos [20]	MSP430	8	60	2	512	CC2420	2004
ZebraNet [21]	MSP430	8	60	2	3.8	9xStream [22]	2003
panStamp NRG 2 pag github	CC11XX	8 - 24	32	4		CC11XX	2015
.NOW [23]	STM32F103	72	16 - 1024 [24]	6 - 96 [24]		AT86RF231	2012

É muito usual utilizar *System On Chip* (SoC) para RSSF quer por questões de espaço quer por eficiência energética. Um exemplo disso é o FireFly3 [25] desenvolvido pela Universidade *Carnegie Mellon* que utiliza como SoC o ATmega128RFA1. Este SoC tem um microcontrolador de 8 bit com um sistema de radio (que respeita a norma 802.15.4 do IEEE e *ZigBee*). Segundo o datasheet [26] é possível afirmar que o SoC consome apenas 16.6 mA enquanto recebe informação e cerca de 18.6 mA quando realiza a transmissão, o que é bastante satisfatório.

2.2.7 Meio de transmissão

Os nós de RSSF podem estar interligados de duas formas, por meio ótico ou por radio frequência [12].

Atualmente as RSSF utilizam radio frequência, a frequência de transmissão é normalmente 2.4Ghz, embora para determinadas aplicações isso possa ser relevante, sendo que pode existir a necessidade de baixar a frequência. O exemplo disso é o uAMPS [27] que é um *mote* que usa 2.4Ghz como frequência de transmissão. Já o *mote* apresentado em [28], anunciado como *Low-Power*, usa como frequência de transmissão apenas 916Mhz de modo a reduzir o consumo de energia.

Como foi dito, o meio de transmissão ótico, apesar de ser robusto a interferências eletromagnéticas, tem um grande requisito para que tudo funcione

em conformidade. Os *motes* têm que estar de certa forma alinhados para que estabeleçam a ligação uns com os outros [12].

As características do meio onde é inserida a rede é que determina qual o tipo de transmissão é preferível usar.

2.2.8 Consumo de energia

Um dos grandes requisitos de uma RSSF é o baixo consumo de energia. Em alguns cenários é impossível fazer a troca da unidade de energia. A vida útil de um nó sensorial depende diretamente da vida útil dessa unidade.

A falha de um nó sensorial numa rede pode obrigar a rearranjos na topologia da rede e haver necessidade de reencaminhamento de mensagens.

Como já foi visto (figura 2.1) a unidade de energia alimenta 3 outras unidades

- Unidade de Sensoriamento;
- Unidade de Processamento;
- Unidade de conectividade sem fios.

No caso da unidade de sensoriamento o consumo é o menos significativo comparando com as outras duas unidades.

Por parte da unidade de processamento, em termos quantitativos é inferior à Unidade de conectividade sem fios [29]. De modo a reduzir o consumo nesta unidade, sempre que possível, é reduzida a frequência de relógio e se possível a tensão de alimentação não abdicando da estabilidade do nó [30]. Com isto temos poupança de energia visto que apenas momentaneamente se deseja o desempenho máximo, no resto do tempo a tensão de alimentação é reduzida e a frequência de relógio também.

Relativamente à unidade de conectividade sem fios, esta é deveras a que tem um maior consumo de energia [29]. Esta tem como tarefa enviar e

receber dados, ambas as tarefas tem em média o mesmo consumo de energia. Em todos os elementos não se deve apenas contabilizar a potência quando ativo mas sim a sua potencia no arranque ou na inicialização. Mesmo que o arranque dure apenas escassos micro segundos, este não poderá ser ignorado.

2.2.9 Interferência

Com o aumento da quantidade de aplicações que hoje em dia utilizam comunicação sem fios o espetro disponível está a ficar mais "poluído", aumentando a interferência e reduzindo por exemplo a qualidade do serviço que as RSSF proporcionam.

Muitos dos dispositivos sem fios operam na *Industrial Scientific and Medical* (ISM) de 2.4 GHz, que é uma parte do espetro de rádio que pode ser utilizado para qualquer finalidade, sem uma licença na maioria dos países como é o caso do Bluetooth, do Wi-Fi entre outros. Apesar disso, outros dispositivos que não têm uma comunicação sem fios podem causar interferência.

Os resultados de vários estudos mostram que os equipamentos mais comuns que podem causar interferência na banda de 2.4 GHz são fornos micro-ondas, aquecedores industriais, sistemas de iluminação por radiofrequência e equipamentos de soldadura [31].

Na figura 2.2 é possível ver a potência induzida nas componentes de frequência da banda de 2.4 GHz (IEEE 802.11) e um forno micro-ondas.

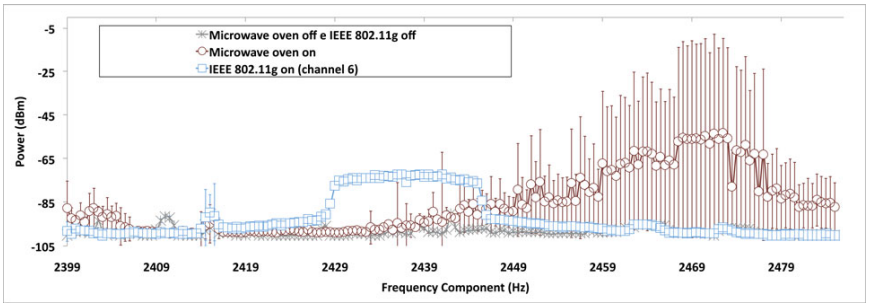


Figura 2.2: Potência induzida nas componentes de frequência [32]

2.3 Aplicações

As RSSF podem ser utilizadas para as mais diversas aplicações. Estas percorrem as mais diversas áreas como por exemplo agricultura, militar, ambiente, saúde, aplicações domésticas, entre outras.

2.3.1 Militares

As RSSF podem ser uma parte integrante na área militar. A implantação rápida, a auto-organização e a tolerância a falhas são características das RSSF o que proporciona uma solução viável na área militar [4]. As RSSF são baseadas numa implantação densa, nós sensoriais descartáveis e de baixo custo. A destruição de alguns nós numa RSSF por ações hostis não pode afetar uma operação militar, tanto quanto a destruição de um sensor tradicional, o que torna o conceito das RSSF uma boa abordagem para campos de batalha [4].

Nesta área é importante estar constantemente a par do estado das tropas aliadas, da condição e a disponibilidade do equipamento e das munições no campo de batalha e isso é possível com a ajuda das RSSF. Cada soldado, veículo, equipamento podem incorporar pequenos sensores de modo a reportar o seu estado [4].

Nos terrenos mais críticos, nas rotas de aproximação, nos caminhos estreitos pode ser implementado RSSF possibilitando assim a observação das atividades das forças inimigas [4].

2.3.2 Meio Ambiente

O custo relativamente baixo permite a instalação, de forma densa, de nós que podem monitorizar o meio ambiente. A rede pode fornecer informações como por exemplo, alertar os agricultores do início de geada, de chuva, vento forte entre outros.

Outra aplicação é na deteção de incêndios, tradicionalmente é utilizado sistemas de vigilância com câmaras, sensores de infravermelhos ou satélites [33]. Estes sistemas não conseguem proporcionar uma atuação em tempo real. Uma RSSF pode detetar e de certa forma prever fogo numa floresta, com maior rapidez do que a abordagem tradicional. Sistemas de vigilância de incêndios tendo a base de uma RSSF já foram implementadas. O exemplo disso é a Ege University que desenvolveu uma RSSF para deteção de incêndios, os nós sensoriais desta rede tem equipado um sensor de temperatura e um sistema de localização. O objetivo é medir as variações rápidas de temperatura e onde é que essa variação ocorreu determinando assim o foco do incêndio [34]. Os sensores recolhem informação sobre a humidade, temperatura, fumo e a velocidade do vento. Estes fatores são determinantes para avalizar o risco de incêndio [35].

Agricultura de Precisão

Outra grande área onde se aplica as RSSF é na agricultura de precisão (figura 2.3) . Com uma RSSF aumenta a eficiência, produtividade e rentabilidade, minimizando impactos sobre da vida selvagem e do meio ambiente [36]. A informação em tempo real proporciona uma base sólida para as estratégias dos agricultores. Beckwith, entre outros, implementaram uma RSSF numa vinha. Esta rede continha 65 extitmotes cujo objetivo era recolher as temperaturas durante seis meses. A informação foi utilizada para abordar dois parâmetros importantes na produção de vinho: soma de calor e danos causados pela geada [37].

2.3.3 Na área alimentar

A indústria de alimentar enfrenta mudanças críticas para dar resposta às necessidades dos consumidores, além de preocupações com a saúde e segurança, é exigido uma diversidade cada vez maior de produtos alimentares

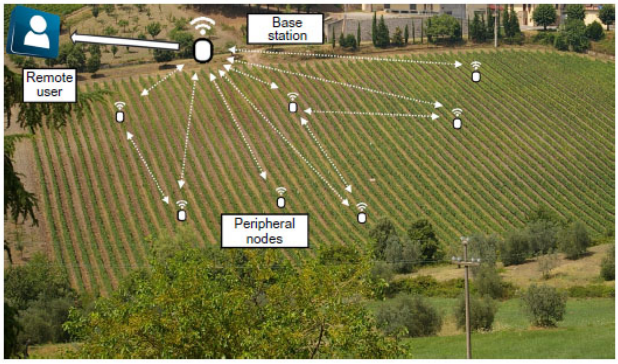


Figura 2.3: Arquitetura de uma rede de sensores sem fios implantada numa vinha [38]

com elevados padrões de qualidade.

A qualidade destes produtos pode mudar rapidamente, porque eles são submetidos a uma variedade de riscos durante a produção, transporte e armazenamento, o que compromete a qualidade [39].

Produtos alimentares perecíveis, como legumes, frutas, carne ou peixe exigem transportes frigoríficos. Portanto, a temperatura é o fator mais importante quando se pretende prolongar a vida útil deste tipo de alimentos. Estudar e analisar a temperatura dentro de salas de refrigeração, contentores e camiões é a principal preocupação da indústria. O uso de uma RSSF em veículos frigoríficos é uma solução para colmatar a falta informação da temperatura durante a viagem [39]. Os veículos contêm uma variedade de sensores para detetar, identificar, registar e comunicar o que acontece durante a viagem de modo controlar o estado de produtos perecíveis nos transportes.

2.3.4 Casos práticos

Seguidamente é apresentado alguns projetos que fizeram uso de uma RSSF.

Great Duck Island

Durante 2002, investigadores da Universidade da Califórnia em Berkeley e *The College of the Atlantic* desenvolveu uma RSSF com 32 nós numa ilha deserta ao longo da costa do estado de Maine, situado nos Estados Unidos. O objetivo era monitorizar o habitat de uma pequena ave marinha chamada Oceanodroma leucorhoa [40]. Estas aves são facilmente perturbadas pela presença humana, portanto, o uso de uma RSSF foi a forma mais adequada de compreender melhor o seu comportamento. A época de reprodução dura sete meses, desde abril a outubro. Os biólogos estavam interessados no padrão de nidificação, mudanças nas condições ambientais dentro e fora das tocas durante a época de reprodução, e ainda nas variações entre os locais de reprodução [40].

Os nós formam instalados no interior das tocas e na superfície. Os nós sensoriais medem a humidade, a pressão, a temperatura e nível de luz ambiente.

Os nós da rede são *motes* Mica mas ligeiramente modificados. O *mote* já modificado, conta com sensor de temperatura, um *Light Dependent Resistor* (LDR), sensor de pressão barométrica, sensor de humidade e sensores infravermelhos passivos (sensor de movimento)[40]. Para suportar as condições ambientais adversas estes *motes* foram cobertos com um selante de modo a proteger a placa da sua exposição à água. Os sensores, permaneceram expostos a preservar a sua sensibilidade [40]. Na figura 2.4 é possível ver a caixa em acrílico utilizada para a implantação da RSSF.

Os *motes* foram espalhados numa área de cerca de 61 mil metros quadrados e os resultados foram encaminhadas para uma base de dados [40]. Os *motes* enviavam regularmente a tensão da bateria com a leitura dos sensores. Esta informação permitiu aos pesquisadores analisar as falhas de nós remotas.

Esta experiência foi planeada para durar 7 meses, mas muitos do *motes*

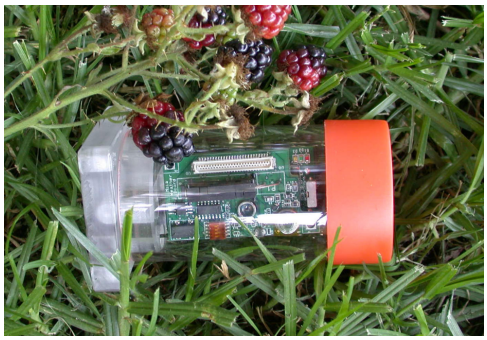


Figura 2.4: Caixa em acrilico utilizada para implantar a RSSF [40]

acabaram por deixar de responder muito mais cedo do que o previsto. Curiosamente só alguns deixaram de responder pelas baterias descarregadas, a maioria não conseguiu suportar o desgaste a partir do exterior [40]. Uma outra surpresa foi o desempenho da rede: os nós enviavam informação com pouca frequência, com o objetivo eliminar as colisões. No entanto, na implantação verifica-se que por diferentes causas os nós começam a enviar um número de pacotes elevado, causando assum um congestionamento da rede [41].

ZebraNet

O objetivo do projeto ZebraNet foi monitorizar manadas de zebras nas planícies do Quénia. Para isso foram anexados colares com *motes* nas manadas de zebras criando um registo de dados da sua posição utilizando o GPS durante um ano. Cerca de 35 mil zebras andaram livremente numa área de 40 mil quilómetros quadrados em Laikipia [42]. A velocidade e a direção dos animais individuais num grupo estão estreitamente correlacionadas, a análise de uma manada inteira pode ser conseguida analisando apenas um único ou poucos animais de um grupo, reduzindo assim o número de colares necessários.

A localização tradicional é baseada em colares nos animais com transmissor *Very High Frequency* (VHF). Para localizar os animais é necessário

sobrevoar os locais de modo a "escutar"as respostas dos transmissores [42]. Como as zebras andam livremente dá origem a um cenário radicalmente diferente do que o Great Duck Island falado anteriormente. Os *motes* são móveis, a estação base é móvel, os *motes* não estão sempre em contacto com uma estação base. Para este caso não é viável criar uma infraestrutura fixa, principalmente por causa do risco de vandalismo devido à sua grande área. Para resolver estes problemas, os autores observam que as manadas de zebras tendem a reunir-se regularmente em lagos espalhados por todo o parque. Usando esta observação, eles escolhem uma estratégia de divulgação de dados *peer-to-peer*.

Os autores desenvolveram várias versões do protótipo (colar), desde o primeiro protótipo até pequenas plataformas integradas alimentados por painéis solares [43]. Em janeiro de 2004, um lote da ultima versão foram implantados no Quénia. Esta ultima versão utiliza um recetor GPS para obter a localização em intervalos regulares e regista isso na própria memoria *flash* [43].

É também de salientar que, embora os autores apontam para muitas ineficiências e otimizações de energia na plataforma, que acabou por surtir resultados bastante satisfatórios [43].

2.4 *Internet of Things*

O IoT pode ser definido como uma rede de dispositivos eletrónicos pequenos, de baixo custo e de baixa potência, onde as comunicações ocorrem sem a intervenção humana. Cada dispositivo atua como um "nó inteligente"na rede, detetando informações e executando processamento de sinal de baixo nível para filtrar sinais de ruído e reduzir a largura de banda necessária para comunicações de nó para nó. Finalmente, os nós enviam esta informação para a "nuvem"de forma segura para proteger, armazenar e processar dados. Os analistas esperam que a IoT cresça para aproximadamente 36 bilhões

de dispositivos conectados até 2020 [44]. Atualmente, existem inúmeras aplicações para o IoT, das quais destacam-se as casas inteligentes, *wearables*, entre outros.

Smart Home é o termo mais pesquisado no Google relacionado com IoT. Numa casa com um sistema de *Smart Home*, é possível ligar o ar condicionado antes de chegar em casa ou desligar as luzes depois que se saiu de casa. Também é possível destrancar as portas para os amigos para acesso temporário, mesmo não estando em casa. As casas inteligentes tornam a vida mais simples e conveniente. Os produtos associados a *Smart Home* prometem economizar tempo, energia e dinheiro.

Outro ótimo assunto são os dispositivos *Wearables*, que contêm sensores e *software* que reúnem dados e informações sobre o utilizador. Estes dados são posteriormente processados para extrair um perfil do utilizador. Esses dispositivos são utilizados principalmente na área de *fitness*, saúde e entretenimento [45].

Na indústria existe o *Industrial Internet of Things* (IIoT). O grande objetivo por trás do IIoT é que, sejam criada máquinas mais inteligentes e mais precisas e consistentes do que os humanos na comunicação de dados. Isso pode ajudar as empresas a resolver ineficiências e problemas com mais facilidade [46].

Esta página foi intencionalmente deixada em branco.

Capítulo 3

Tecnologias de Redes de Sensores sem Fios

Muitas das novas inovações são possíveis devido à existência de *chips* de radio com baixo consumo de energia. Até há pouco tempo, a única forma de conseguir a transferência de dados entre um sensor e um servidor era com a utilização de fios, ou manualmente recolher dados a partir de um dispositivo de registo cronológico. As tecnologias sem fio já estão disponíveis há décadas. No entanto, é tendencial utilizar quantidades significativas de energia e precisam de equipamento dedicado para estabelecer as comunicações.

A maioria das aplicações são caracterizadas pela transferência periódica de pequenas quantidades de informações do sensor entre os nós de sensores e um dispositivo central. Alguns produtos que podem implementar um sistema de rádio de baixa potência são por exemplo: *smartphones*, dispositivos de *fitness*, domótica, aquecimento, ventilação e ar condicionado, controlos remotos, pagamentos, e muitos outros.

Estas aplicações são limitadas pelos seguintes requisitos críticos: baixo consumo de energia, baixo custo e tamanho físico.

O requisito do baixo consumo de energia é devido à necessidade dos dispositivos terem de operar por longos períodos de tempo a partir de pequenas

baterias. Além das vantagens óbvias de um baixo custo, a despesa global do produto é largamente afetada pela fonte de energia.

Atualmente existem tecnologias de radio de baixo consumo energético. Algumas foram desenvolvidas para aplicações específicas, enquanto outros são mais genéricos e flexíveis, como por exemplo *Bluetooth Low Energy* (BLE), ZigBee, ANT. Também é possível criar uma RSSF usando o Wi-Fi (*Institute of Electrical and Electronics Engineers* (IEEE) 802.11) [47], mas o consumo de energia é bastante elevado, e de curta autonomia uma vez alimentado com uma bateria, continua a ser uma desvantagem importante.

3.1 *Bluetooth Low Energy*

BLE apareceu como um projeto no Centro de Pesquisa Nokia com o nome Wibree. Em 2007, a tecnologia foi adotada pelo Bluetooth Special Interest Group (SIG) e renomeado para *Bluetooth Low Energy* [48].

Dispositivos BLE são normalmente desenvolvidos para funcionar durante muitos anos sem a necessidade de uma bateria nova.

A tecnologia BLE é direcionada principalmente para *smartphones*, onde é previsto uma topologia de rede em estrela, similar ao Bluetooth clássico, muitas vezes, criada entre o *smartphone* e um ecossistema de outros dispositivos.

3.2 ANT

ANT é uma tecnologia radio sem fios proprietária de baixa potência que opera no espectro de 2.4 GHz. Foi criado em 2004 pela empresa de Dynastream [49]. Normalmente, o *transceiver* ANT é tratado como uma "caixa preta"e não exige muito esforço para a implementação de uma rede. O principal objetivo é permitir que os sensores, destinados para o desporto, possam comunicar com uma unidade de visualização, como por exemplo um relógio

ou um ciclo computador[50].

O protocolo ANT oferece flexibilidade em termos de definições implementadas pelo utilizador, como por exemplo, implementações ao nível de segurança [51]. Devido à *stack* do ANT ser extremamente compacta, faz com que os recursos computacionais do microcontrolador possam ser reduzidos [51].

3.3 ZigBee

ZigBee é um protocolo que utiliza o padrão IEEE 802.15.4 como base e foi criado em 2002 por um grupo de 16 empresas. O ZigBee é projetado para criar redes em malha e é direcionado para aplicações como medidores inteligentes, automação residencial e unidades de controlo remoto. Infelizmente, complexidade e requisitos de energia do protocolo do ZigBee não tornam particularmente adequados para dispositivos sem manutenção que precisam operar por longos períodos com uma fonte de energia limitada. Os canais do ZigBee são semelhantes aos do BLE ambos de 2 MHz de largura. No entanto, são separados por 5 MHz, perdendo assim, um pouco espetro. ZigBee requer um planeamento cuidadoso durante a implantação, de modo a garantir que não há sinais de interferência na vizinhança [50].

3.4 Z-Wave

Z-Wave é uma tecnologia de baixo consumo energético de comunicações de radio que é projetado principalmente para produtos de automação residencial como controladores de lâmpadas e sensores, entre outros. É um protocolo de baixa latência para pequenos pacotes de dados com taxas de transferência até 100 kbit/s. Z-Wave opera na faixa de sub 1 GHz e é imune à interferência de Wi-Fi e outras tecnologias sem fio da faixa de 2.4 GHz, como é o caso do BLE ou ZigBee. Ele suporta redes em malha, sem a ne-

cessidade de um nó coordenador, permite o controlo de até 232 dispositivos numa rede. Z-Wave usa um protocolo mais simples do que outros protocolos, o que pode permitir o desenvolvimento mais rápido e mais simples, mas o único fabricante de chips é Sigma Designs o que torna a escolha um pouco limitada em comparação com outras tecnologias sem fio, como ZigBee ou BLE [52].

3.5 Comparativo

Atualmente as duas tecnologias padrão estão mais utilizadas para este tipo de redes sem fios são o ZigBee e BLE. Ambas utilizam a banda ISM de 2.4 GHz (tabela 3.1), que é uma parte do espetro de rádio que pode ser usado para qualquer finalidade, sem uma licença na maioria dos países. Geralmente, ao aumentar a frequência, permite que seja enviada uma maior quantidade de dados no mesmo período de tempo, mas os requisitos de energia são também mais elevados e a distância de transmissão é consideravelmente mais curta [53].

Cada uma destas tecnologias tem uma série de características que a outra não tem, caso seja necessário alguma delas, a escolha torna-se mais clara. Uma vez que é necessária uma comunicação de baixo consumo de energia chega-se à conclusão que o BLE é o mais adequado. Por outro lado, se for necessário implantar RSSF numa grande área, o ZigBee é uma escolha mais acertada. Em termos de energia, o ZigBee e o BLE são bastante diferentes também. Dependendo da aplicação ZigBee pode proporcionar um baixo consumo de energia face ao BLE ou vice-versa [54]. Mas em vários estudos em que o objetivo é comparar o consumo de energia entre o BLE e o ZigBee [54], o BLE ganha vantagem, ou seja, tem um consumo de energia inferior face ao ZigBee. O BLE utiliza uma comunicação síncrona, o que significa que tanto o mestre e o escravo acordam, por assim dizer, de forma síncrona. Isso ajuda a manter o baixo consumo de energia tanto no lado do escravo

Tabela 3.1: Comparativo entre BLE e ZigBee

	BLE	ZigBee
Noma IEEE	IEEE 802.15.1 [55]	IEEE 802.15.4 [56]
Frequência	2.400-2.4835 GHz [57]	868-868.8 MHz 902-928 MHz 2.400-2.4835 GHz [58]
Potência	0,01 a 10mW [57]	100 mW [59]
Modulação	*Gaussian Frequency Shift Keying* (GFSK) [57]	*Offset Quadrature Phase Shift Keying* (OQPSK) e *Binary Phase Shift Keying* (BPSK) [60]
Taxa de Transferência	até 0.3 Mbps,[61]	250 kbps, 40 kbps, and 20 kbps [62]
Topologia	Estrela	Estrela, Malha, Árvore
Alcance	77m [59]	291m [59]
Nós	Ilimitado [61]	64000 [63]
Segurança	128 bit AES encryption [61]	128 bit AES encryption

como no mestre. Já o ZigBee usa comunicação assíncrona, isto significa que os *routers* ficam sempre no ativo. O consumo de energia dos *routers* é relativamente elevado, mas cada nó sensorial pode acordar a qualquer momento e enviar os seus dados não tendo que esperar por um determinado período de tempo.

Contudo existe alguma sobreposição de áreas de aplicação com os seus prós e contras. BLE é capaz de comunicar com milhões de dispositivos *Bluetooth*, mas não suporta a topologia de malha. O ZigBee pode dar cobertura a grandes áreas, mas não é adequado para redes ponto a ponto e requer *routers* com elevada potência.

3.6 *Motes* existentes

Seguidamente apresenta-se vários *motes* existentes no mercado. Desde soluções mais profissionais e mais abrangentes com é o acaso do Waspmote até as mais amadoras como o JeeNode.

3.6.1 JeeNode

JeeNode (figura 3.1) tem como unidade de processamento o microcontrolador ATmega328. A unidade de rádio frequência é o chip RFM12B. É um *mote* com dimensões relativamente reduzidas (80.7 x 21.1 mm) [64]. Dispõe de conectores na sua lateral de modo a acoplar uma ou mais unidades sensoriais, tornando assim JeeNode uma solução modular. Este *mote* apresenta um conector com interface série que permite realizar a programação, alimentação e comunicação. A alimentação pode ser feita de duas formas, quer pelo conector genérico quer pelo próprio conector dedicado para a bateria. A tensão de alimentação pode ser de 3.5V a 13V.

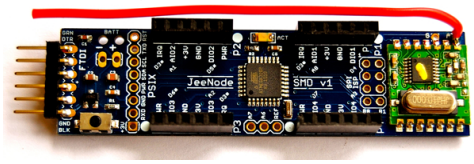


Figura 3.1: Vista geral do JeeNode SMD [64]

3.6.2 WiSense

O *hardware* da plataforma Wisense é modular, o que permite personalizar, redesenhar e modificar o *mote* [65]. Na figura 3.2 é possível visualizar de que modo o *mote* WiSense está organizado, a placa destinada aos sensores pode incluir até oito sensores, no entanto, a placa de rádio já dispõe de dois sensores (temperatura e luminosidade). Esta mesma placa dispõe de um chip de radio, CC2520 que permite transmitir as informações via radio de acordo com o protocolo ZigBee. A placa que contém a unidade de processamento dispõe de um microcontrolador (MSP430) da Texas Instruments. Na figura 3.3 é apresentado o *mote* WiSense, de salientar que não está incluído a placa de sensores pois esta é opcional.

Ao microcontrolador é ligado os sensores por diversas interfaces, *Serial Peripheral Interface* (SPI), *Inter-Integrated Circuit* (I2C), pinos digitais

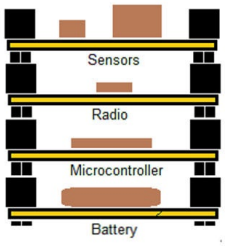


Figura 3.2: Diagrama do *mote* WiSense [65]

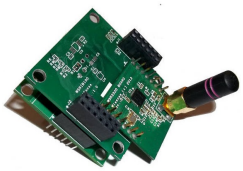


Figura 3.3: Vista geral do *mote* WiSense [65]

ou analógicos, ou mesmo por *Universal Asynchronous Receiver/Transmiter* (UART). Na figura 3.4 é possível ver a arquitetura do sistema. Relativamente ao sistema de alimentação caso se pretenda utilizar uma bateria do tipo moeda ou botão, é possível, existe um *socket* para esse tipo de baterias na parte inferior da placa dedicada ao sistema de radio. Caso se opte por utilizar uma bateria de maior capacidade terá que ser aplicada uma placa extra ao *mote* como é possível visualizar na figura 3.2.

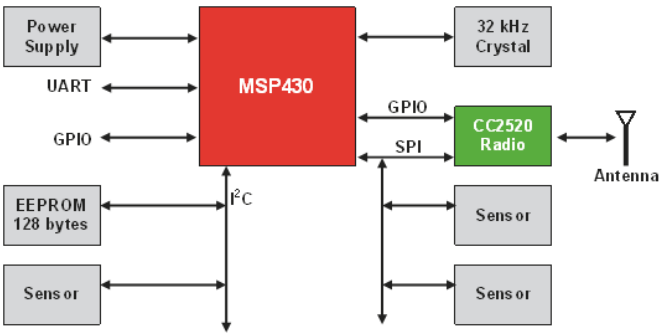


Figura 3.4: Diagrama da arquitetura geral do sistema [64]

Por ser uma plataforma *opensouce* existem vários projetos desenvolvidos com esta plataforma como por exemplo a monitorização de uma estufa [66], sistema de rega sem fios para áreas verde [67], monitorizar botijas de gás [68], entre outros.

3.6.3 Zolertia RE-Mote

O RE-Mote (figura 3.5) é uma plataforma de desenvolvimento elaborada por universidades em conjunto com parceiros industriais. O RE-Mote visa preencher a lacuna das plataformas IoT existentes, a falta um projeto a nível industrial e ultra baixo consumo de energia [69].

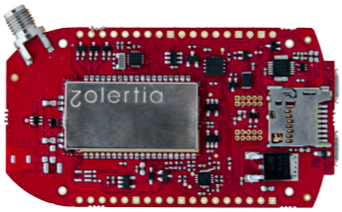


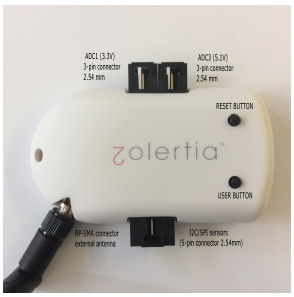
Figura 3.5: Visão geral do Zolertia RE-Mote [69]

A plataforma é baseada no sistema Texas Instruments CC2538, ou seja uma SoC que dispõem de um ARM Cortex-M3 juntamente com um equipado com a interface de radio de 2.4 GHz, pode funcionar até 32 MHz, com 512 KB de *flash* programável e 32 KB de RAM. Além disso esta plataforma dispõe de um chip, o CC1200, que nada mais é que um sistema de radio a funcionar nas bandas 868/915 MHz o que permite a operação de banda dupla, quer a 2.4 GHz ou a 868/915 MHz [69].

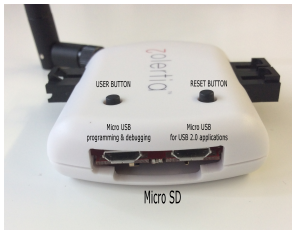
A disponibilidade de dois rádios permite usar esta plataforma em aplicações residenciais / interior ou de longo alcance. O alcance máximo é entre 100 metros e 20 km, com os parâmetros de rádio altamente configuráveis como a modulação, taxa de dados, potência de transmissão, etc. [69].

O ultra baixo consumo de energia, desde 1uA até 150nA, significa que é possível alimentar uma aplicação, teoricamente, durante mais de 10 anos com um par de 2 pilhas AA com capacidade de 2300mAh [69].

Este *mote* dispõe de armazenamento externo, cartão micro SD, de modo a armazenar dados [69].



(a) Vista lateral



(b) Vista Frontal

Figura 3.6: Portos e conectores do RE-Mote [69]

Relativamente à bateria, utiliza baterias recarregáveis *Lithium-ion Polymer Battery* (LiPo) e é possível recarregar a mesma sem qualquer componente externo. Diretamente, com painéis solares ou outros métodos de colheita de energia ou até enquanto o RE-Mote está ligado através da da interface *Universal Serial Bus* (USB) [69].

O RE-Mote dispõe de interfaces e conectores (figura 3.6) para que seja possível conectar diretamente sensores (analógico e digital), é ainda possível acoplar uma antena externa de modo aumentar a propagação do sinal.

3.6.4 Waspmote

Waspmote (figura 3.7) é uma plataforma de sensores sem fios *opensouce* especialmente desenvolvido para uma implementação de baixo consumo de energia para permitir que os nós sensoriais sejam completamente autónomos alimentados apenas por uma bateria, oferecendo assim um período de vida variável entre 1 e 5 anos, dependendo do ciclo de trabalho e o rádio utilizado [70].

Waspmote baseia-se numa arquitetura modular. A ideia é integrar apenas os módulos necessários para cada aplicação de modo a otimizar os custos. Por este motivo todos os módulos (rádio, placas de sensores, etc.) são conectáveis fisicamente [70].

Em termos de *hardware* este *mote* tem integrado um microcontrolador

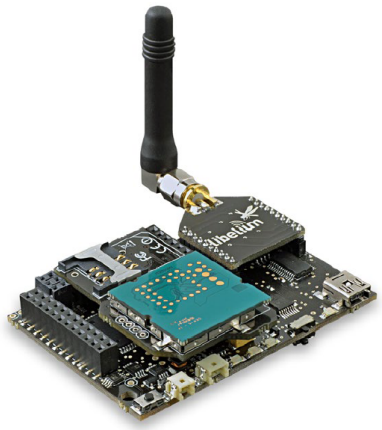


Figura 3.7: Waspmote [71]

ATmega1281 a funcionar a 14.7456 MHz. A placa dispõe de 7 entradas analógicas, 8 entradas/saídas digitais, 1 *Pulse Width Modulation* (PWM), 2 UART, 1 I2C, 1 USB, 1 SPI [71].

Waspmote conta com um acelerómetro de três eixos na própria placa e mais de 80 placas de sensores (*shields*) já desenvolvidas que é possível ligar por cima da placa principal. A ideia é facilitar a integração e uso de sensores complexos que necessitam de sistemas eletrónicos especiais para funcionarem. Na figura 3.8 é possível visualizar 4 das mais de 80 *shields* [71]. No caso da *Sheild* Eventos (figura 3.8b), inclui sensores de pressão, vibração, temperatura, presença e presença de líquidos. Já na *Sheild* Água (figura 3.8c) é possível determinar a quantidade de cálcio, fluoreto, nitrato, brometo, cloreto, iodeto, temperatura e o pH de uma determinada solução.

Há 17 interfaces sem fios diferentes para Waspmote. Algumas delas são [71]:

Longo alcance *General Packet Radio Service* (GPRS), 868MHz, 900MHz;

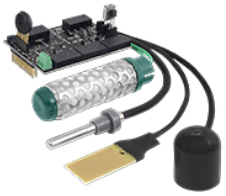
Médio alcance ZigBee e do WiFi;

Curto alcance *Radio-Frequency IDentification* (RFID), *Near Field Communication* (NFC) e o Bluetooth 4.0;

(a) *Sheild* Gases

(b) *Sheild* Eventos

(c) *Sheild* Água



(d) *Sheild* Agricultura

Figura 3.8: Placas de sensores já desenvolvidas para o *mote* Waspmote [70]

Estas interfaces podem ser utilizadas isoladamente ou uma combinação de dois, pois a placa tem 2 conectores para conectar *shields* de rádio [71].

Esta página foi intencionalmente deixada em branco.

Capítulo 4

Bluetooth Low Energy

Bluetooth atualmente é uma tecnologia indispensável num telemóvel, permitindo assim os dispositivos moveis comunicarem entre si em curtas distâncias. É deste modo que o auricular sem fios comunica com o telemóvel sem a complexibilidade e a energia que o WiFi requer. Em particular o BLE é essencial para pequenos dispositivos móveis porque este requer pouquíssima energia para funcionar.

4.1 O que é o *Bluetooth Low Energy*?

Comparando com outras tecnologias sem fios, o rápido crescimento do BLE é relativamente fácil de explicar. Esse crescimento está diretamente relacionado com a evolução dos *smartphones*, *tablets* e comunicações móveis. A adoção do BLE desde o inicio foi feita pelos dois grandes líderes de mercado na área dos *smartphones* a Apple e a Samsung. O primeiro *smartphone* que faz uso do BLE, é o iPhone 4S, lançado em outubro de 2011 [72].

Embora o mercado dos *smartphones* e *tablets* estar perfeitamente maduro, surge a necessidade de criar algum tipo de conectividade com o mundo exterior.

O propósito do BLE é a criação de uma nova classe de pequenos dispo-

sitivos, os quais que apenas necessitam de enviar ou receber uma pequena quantidade de dados com pouca frequência. Para manter o consumo de energia relativamente baixo, o BLE tem uma potência de transmissão (de 0.01 a 10mW) inferior ao *Bluetooth* padrão (100mW para dispositivos de classe 1 e 1mW para classe 3)[57]. Os dados são enviados da mesma forma, mas com uma velocidade inferior, o máximo de 100kbps. Dispositivos BLE podem alternar entre o modo *standby* e o modo ativo mais rápido do que o *Bluetooth* padrão, desta forma é feita economia de energia, permitindo assim que pequenos conjuntos de dados sejam transmitidas.

Muitos mais benefícios estão por trás do BLE, abriram-se portas para o desenvolvimento de *chips* direcionados única e exclusivamente para isso. O que criou um mercado enorme, com produtos, para o consumidor final, criativos e inovadores, como é o caso das pulseiras de *fitness*. Para ter uma ideia este é uso mais comum para BLE, 96% dos dispositivos *fitness* tem BLE [73].

Atualmente é possível comprar um SoC já com BLE incluído. Normalmente o preço é inferior face às tecnologias semelhantes sem fios, como é o caso do WiFi, GSM, Zigbee, etc.

4.1.1 Versões

BLE é também conhecido como *Bluetooth Smart*, corresponde assim à quarta geração do *Bluetooth*. Desde 2010 (ano que o *Bluetooth* 4.0 foi lançado) até agora, o protocolo sofreu várias atualizações, a última, a 4.2, lançada em Julho de 2015 [74] trouxe 3 diferentes classes:

- *Bluetooth* Clássico
- *Bluetooth Smart Ready*
- *Bluetooth Smart*

Tabela 4.1: Compatibilidade entre as versões *Bluetooth*

Dispositivo	Suporte para Bluetooth Clássico	Suporte para Bluetooth Low Energy
Até 4.0	Sim	Não
4.x Single Mode (Bluetooth Smart)	Não	Sim
4.x Dual Mode (Bluetooth Smart Ready)	Sim	Sim

De forma sucinta, a classe *Bluetooth Smart* indica que o dispositivo, como por exemplo um pedómetro ou um monitor de frequência cardíaca, apenas pode enviar ou receber sinais de *Bluetooth Low Energy* enquanto o *Smart Ready* pode tratar sinais *Bluetooth* Clássico e *Bluetooth Low Energy*. O *Bluetooth* Clássico pode enviar ou receber sinais BLE.

Bluetooth Smart é também conhecido por *Bluetooth Single Mode* pois apenas aceita um tipo de sinal, o *Low Energy*. Já o *Bluetooth Smart Ready* é também conhecido por *Bluetooth Dual Mode*, como o nome indica este aceita dois tipos de sinais o Clássico e o *Low Energy*. Na tabela 4.1 é possível ver as compatibilidades entre as versões *Bluetooth*.

4.2 Protocolo/Arquitetura do BLE

Irá ser analisado sucintamente as camadas do BLE. Assim é possível obter uma base sólida sobre o protocolo e entender mais facilmente como e porquê que o BLE funciona deste modo.

4.2.1 *Stack* do BLE

Tal como o *Bluetooth* Clássico [75], a *stack* do BLE é composta por 3 partes [61]:

- Controlador;
- *Host*;
- Aplicação.

O controlador é tipicamente o dispositivo físico que pode transmitir e receber sinais de radio. O *host* é normalmente uma camada de *software* que gere como dois ou mais dispositivos comunicam entre si. A camada de aplicação é a camada mais superficial e é a que contem a lógica e a interface com o utilizador. A arquitetura desta ultima camada depende muito da aplicação para o qual foi desenvolvido.

Na figura 4.1 é apresentado um diagrama onde apresenta a arquitetura do BLE e seguidamente é detalhada cada uma das camadas do BLE.

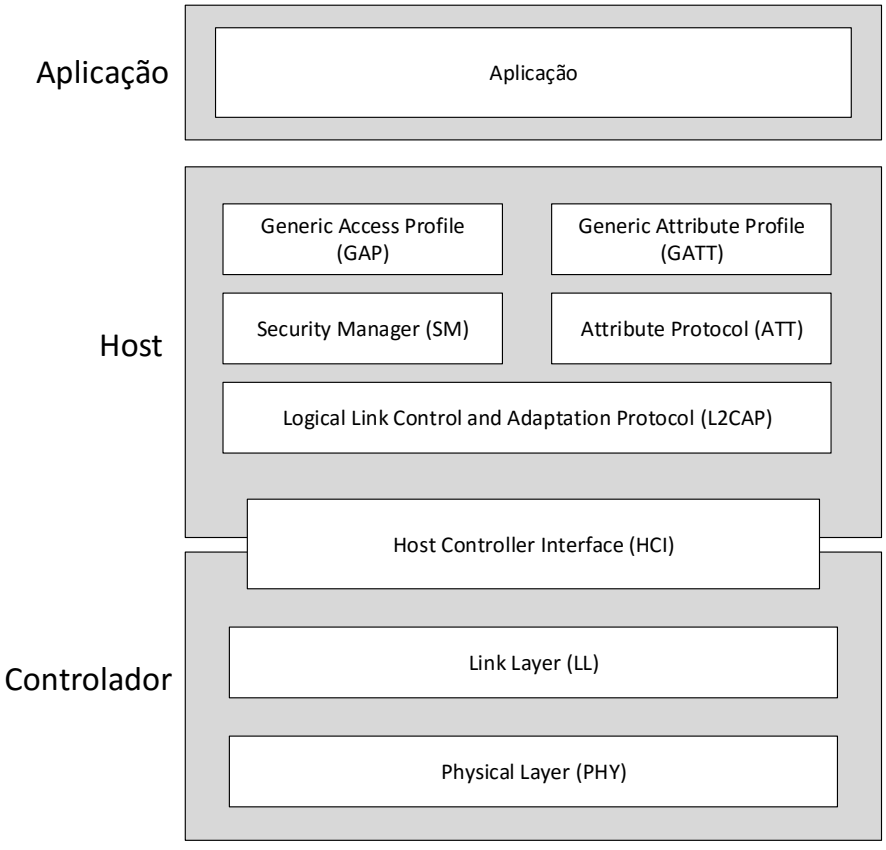


Figura 4.1: *Stack* do BLE (adaptado de [61])

4.2.2 *Physical Layer*

A *Physical Layer* é a camada responsável pela transmissão e receção de sinais via radio de 2.4Ghz. Para isso é necessário existir modulação e desmodulação, transformação de sinais analógicos em sinais digitais e é nesta camada onde isso acontece.

As ondas de radio podem propagar informação de vários modos, desde a variação da amplitude, frequência, entre outros. No BLE, é utilizado o esquema de modulação chamado GFSK [76]. Isto quer dizer que os níveis lógicos alto e baixo são codificados dentro da onda de radio pelo deslizamento da frequência. O nível lógico alto, que é representado por um aumento da frequência. O nível lógico baixo é representado por uma diminuição da frequência.

Como se sabe a transmissão ou receção é feita a 2.4Ghz. Esta banda, no caso do BLE é divida em 40 canais, no caso do *Bluetooth* Clássico é dividido em 79. Ambas desde 2.4000GHz até 2.4835GHz [61].

Na figura 4.2 existem três canais que são chamados de *advertising*. Estes canais são utilizados para realizar a procura de outros dispositivos ou para promover a sua presença enquanto outros dispositivos realizam a conexão. O *Bluetooth* Clássico usa 32 canais para a mesma tarefa.

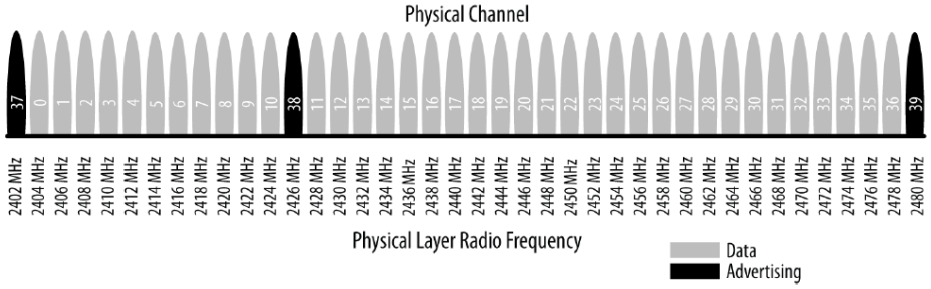


Figura 4.2: Canais do BLE (A escuro estão apresentados os canais *advertising*) [77]

Esta drástica redução de canais de *advertising* é mais um truque que o BLE usa para minimizar o tempo "ativo", assim reduzindo o consumo de

energia. O BLE está no "ativo"durante apenas 0.6ms a 1.2ms para procurar outros dispositivos utilizando os três canais *advertising*. O *Bluetooth* Clássico, requer cerca de 22.5ms para percorrer os 32 canais. A poupança de energia é significativa. O BLE consome de 10 a 20 vezes menos energia que o *Bluetooth* Clássico para localizar outros dispositivos [61].

4.2.3 *Link Layer*

Nesta camada é realizada a interface com a *Physical Layer*, é responsável pela procura de dispositivos, criação e manutenção das conexões[77].

Antes de começar a descrever esta camada de forma sucinta existe algumas designações que devem estar presentes.

Olhando para dois dispositivos estes podem ser, mestre e escravo quando estão conectados. Mas também podem ser denominados por anunciante (*advertiser*) e por procurador (*scanner*) quando não estão conectados.

Podemos assim descrever cada um deles [77]:

Mestre - Dispositivo que inicia a conexão.

Escravo - Dispositivo que aceita o pedido de conexão.

Anunciante - Dispositivo que envia pacotes de anúncio.

Procurador - Dispositivo que que procura pacotes de anúncio.

Como já foi visto existem canais de *advertising* e canais de dados. O envio de pacotes de anúncio é feito apenas nos canais de *advertising*. O canal dos dados apenas é utilizado como o nome indica para o envio ou receção de dados depois de a conexão estar estabelecida.

A transmissão de pacotes através dos canais de *advertising* ocorre em intervalos de tempo específicos, denominados de eventos de *advertising*. Dentro de um evento de *advertising*, o anunciante usa o canal de *advertising* para transmitir pacotes. A comunicação de dados bidirecional entre dois dispositivos requer uma conexão. A criação dessa conexão entre dois dispositivos é

um processo assimétrico no qual um anunciante comunica através dos canais de *advertising* que é um dispositivo conetável, enquanto o outro dispositivo (referido como procurador) está à escuta nos canais de *advertising*. Quando um procurador encontra um anunciante, pode transmitir uma mensagem a solicitar uma conexão para o anunciante, o que cria uma conexão ponto-a-ponto entre os dois dispositivos. Ambos os dispositivos podem se comunicar usando os canais de dados [77].

O BLE define duas funções do dispositivo na camada *Link Layer* para uma conexão criada: o mestre e o escravo. Estes são os dispositivos que funcionam como procurador e anunciante durante a criação da conexão, respetivamente. Um mestre pode administrar múltiplas conexões de forma simultânea com diferentes escravos, enquanto cada escravo só pode estar conectado a um mestre. Assim, a rede composta por um mestre e vários escravos, o que é chamado de piconet, que segue uma topologia em estrela [77].

De modo a haver poupança de energia, os escravos estão no modo de suspensão por padrão e acordam periodicamente para ouvir possíveis pacotes provindos do mestre. O mestre determina os casos em que os escravos são obrigados a estar ativos, e portanto, coordena o acesso ao meio usando o esquema de *Time Division Multiple Access* (TDMA) [77].

A partir do momento que conexão entre um mestre e um escravo é estabelecida, o canal físico (*physical Channel*) é dividido em unidades de tempo não sobrepostos chamados eventos de conexão (*Connection Event*). Dentro de um evento de conexão, todos os pacotes são transmitidos utilizando o mesmo canal de dados. Cada evento de conexão inicia-se com a transmissão de um pacote pelo mestre. Se o escravo recebe um pacote, o escravo deve enviar um pacote para o mestre em resposta, como se pode observar na figura 4.3.

Enquanto o mestre e o escravo continuam a trocar pacotes, o evento de

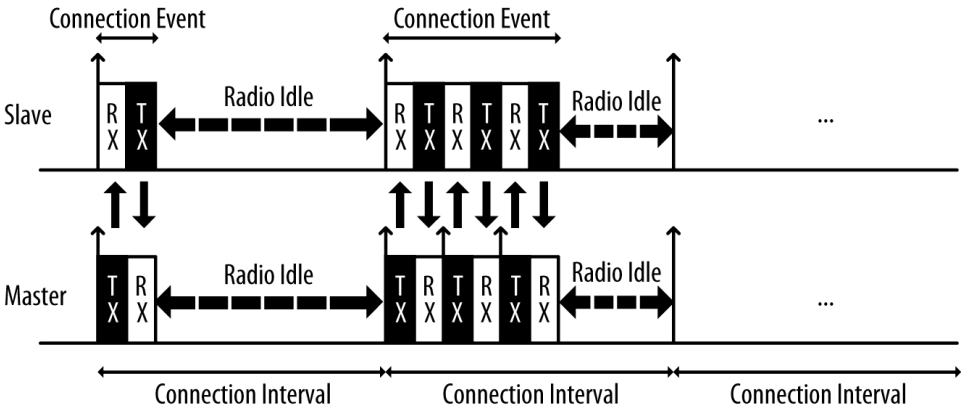


Figura 4.3: Eventos de conexão [77]

conexão é considerado ativo. Pacotes provindos de canais de dados incluem um bit chamado *More Data* (MD) que sinaliza se o remetente tem mais informações a transmitir. Se nenhum dos dispositivos tem mais dados para transmitir, o evento de conexão acabará e o escravo deixa de estar à escuta até ao início do próximo evento de conexão. Outras circunstâncias que forçam o fim de um evento de conexão incluem a receção de dois pacotes consecutivos, quer pelo mestre ou pelo escravo, com erros e a corrupção do campo *access address* de um pacote enviado por qualquer dispositivo. De modo a realizar a deteção de erros, todas as tramas de dados incluem um campo de 24 bits de *Cyclic Redundancy Check* (CRC) [78].

Para um novo evento de conexão, o mestre e o escravo usam o esquema de saltos em frequência de modo a garantir que os dois troquem informação num dado canal num certo instante de tempo, que é calculado através do algoritmo *frequency hopping*. O tempo entre o início de dois eventos de conexão consecutivas é especificado pelo parâmetro connInterval, que é um múltiplo de 1.25ms no intervalo entre 7.5ms e 4s [78]. Outro parâmetro importante é connSlaveLatency, que define o número de eventos de conexão consecutivos durante os quais o escravo não é obrigado a ouvir o mestre e portanto, pode manter o rádio desligado, poupando assim energia. Este parâmetro é um número inteiro entre 0 e 499 e não deve causar um tempo li-

mite de supervisão. O tempo limite de supervisão acontece quando o tempo decorrido desde o último pacote recebido excede o parâmetro connSupervisionTimeout, que é no intervalo entre 100ms e 32s [78]. O objetivo deste mecanismo é para detetar a perda de alguma ligação devido a interferência ou pelo movimento de um dispositivo para fora de alcance.

A *Link Layer* usa um mecanismo de controlo de fluxo *stop-and-wait* baseado em *acknowledge*, o que, ao mesmo tempo, fornece capacidades de recuperação de erro. Cada cabeçalho da trama de dados contém dois campos de um bit chamado o *Sequence Number* (SN) e o *Next Expected Sequence Number* (NESN). O bit SN identifica o pacote, enquanto a NESN indica quais os próximos pacotes. Se um dispositivo recebe uma trama de dados com êxito, o NESN da próxima trama será incrementado, o que no fundo serve como um *acknowledge*. Caso contrário, se um dispositivo receber um pacote com um CRC inválido, o NESN da trama recebida não pode ser lido. Isto força o dispositivo de receção para reenviar a sua ultima trama transmitida [78].

4.2.4 *Host Controller Interface*

O *Host Controller Interface* permite que um *host* comunique com o controlador através de uma interface normalizada.

A implementação desta camada pode ser por intermédio de um API situada no próprio CPU ou por interfaces de *hardware* como UART, SPI ou USB.

A tecnologia dos semicondutores atualmente é barata o suficiente para permitir que as três camadas sejam implementadas num SoC. Em muitos casos de sistemas embebidos, esta integração é preferível, para reduzir o custo e o tamanho do dispositivo final.

4.2.5 *Logical Link Control and Adaptation Protocol*

O *Logical Link Control and Adaptation Protocol* usado no BLE é um protocolo otimizado e simplificado baseado no *Bluetooth* clássico L2CAP. No BLE, o principal objetivo da camada *Logical Link Control and Adaptation Protocol* é multiplexar os dados de três protocolos das camadas superiores, *Attribute Protocol* e *Security Manager*. Os dados destes serviços são manipulados por *Logical Link Control and Adaptation Protocol* sem o uso de retransmissão e fluxo de mecanismos de controlo, que estão disponíveis noutras versões do *Bluetooth* [78]. Os protocolos das camadas superiores fornecem estruturas de dados que se encaixam no tamanho máximo do payload do L2CAP, que é 23 bytes.

4.2.6 *Attribute Protocol*

A *Attribute Protocol* define a comunicação entre dois dispositivos alternando as funções de servidor e de cliente, respetivamente. O servidor contém um conjunto de atributos. Os atributos são estruturas de dados que armazenam as informações da camada *Generic Attribute Profile*, o protocolo que opera por cima do *Attribute Protocol*. O papel do cliente ou servidor é determinado pelo *Generic Attribute Profile*, e é independente do ser escravo ou mestre[78].

O cliente pode aceder os atributos do servidor, fazendo um pedido, que desencadeiam uma serie de mensagens de resposta por parte do servidor. Para maior eficiência, um servidor também pode enviar para um cliente dois tipos de mensagens não solicitadas que contêm atributos, e são eles [78]:

- Notificações, neste caso o cliente não confirma a sua receção;
- Indicações, que exigem o cliente para envie a confirmação de receção da mensagem.

Um cliente também pode enviar comandos para o servidor, de modo a escrever valores nos atributos.

Como foi dito cada servidor contém dados organizados na forma de atributos. Cada um contem 16bits para a *handle*, um identificador único universal *Universally Unique Identifier* (UUID), um conjunto de permissões e finalmente, é claro, um valor do atributo. O *handle* é simplesmente um identificador usado para ter acesso ao valor do atributo. O UUID especifica o tipo e a natureza dos dados contidos no valor [77].

4.2.7 *Generic Attribute Profile*

A camada *Generic Attribute Profile* define uma *framework* que utiliza a camada *Attribute Protocol* para a descoberta de serviços, e a troca de características de um dispositivo para outro. Uma característica é um conjunto de dados que inclui um valor e determinadas propriedades. Os dados relativos aos serviços e as características são armazenados em atributos [78].

Esta informação está organizada hierarquicamente, em secções chamadas de serviços. E cada serviço pode ter várias características.

Por exemplo, um servidor que possui um serviço de "sensor de temperatura"e pode contribuir com uma característica "temperatura"que utiliza um atributo para descrever o sensor, outro atributo para armazenar os valores de medição de temperatura e um atributo adicional para especificar as unidades de medida.

4.2.8 *Security Manager*

O BLE oferece diversos serviços de segurança para proteger a troca de informações entre dois dispositivos conectados. A maioria dos serviços de segurança suportados pode ser expressa em termos de dois modos de segurança mutuamente exclusivas chamado *LE Security Mode 1* e o *LE Security Mode 2* [78]. Estes dois modos oferecem funcionalidade de segurança na

camada *Link Layer* e na camada de *Attribute Protocol*, respetivamente.

O método de emparelhamento utilizado para emparelhar os dispositivos depende da capacidade de entrada/saída dos dois dispositivos.

A camada *Security Manager* tem três métodos de emparelhamento [78]:

Just Works O método de *Just Works* não há troca de chaves a nível de interface do utilizador. Isto é utilizado quando pelo menos um dos dispositivos não tem um mecanismo para qualquer exibição de um código de 6 dígitos ou para a inserção da mesma. Um exemplo disso é os auriculares *Bluetooth* que não tem capacidade de exibição ou a capacidade de inserir uma chave de acesso.

Passkey Entry Este método é projetado para quando o dispositivo tem capacidade de entrada e o outro dispositivo tem capacidade de exibição. Neste método, o código de acesso numérico de 6 dígitos é apresentado num dos dispositivos e o utilizador terá que inserir esse mesmo código de acesso no segundo dispositivo.

Out of Band Este método é desenvolvido para cenários em que um mecanismo de *Out-of-Band* podem ser utilizados para transferir informações de segurança. Por exemplo o NFC, quando é utilizado basta tocar os dois dispositivos para trocar as informações de segurança a fim de emparelhar os dois dispositivos.

O processo de emparelhamento é utilizado para estabelecer uma conexão entre o mestre e o escravo. O emparelhamento é um processo com três fases. As duas primeiras fases são obrigatórias, enquanto a terceira fase é opcional. As três fases são:

1. Emparelhamento e troca de recursos;
2. Criação de *Short-Term Key* (STK);

3. *Key Distribution* (opcional).

A fase 1 e a fase 2 têm como objetivo criar uma ligação (de forma encriptada ou não) enquanto a terceira fase usa uma ligação encriptada usando a STK anteriormente gerada na fase dois.

Na primeira fase os dois dispositivos anunciam as suas capacidades de entrada/saída e a partir delas eles escolhem um método adequado para a segunda fase. O procedimento de emparelhamento é sempre iniciado pelo dispositivo mestre.

A segunda fase tem o objetivo de criar a chave STK, que vai ser utilizado na terceira fase, é gerada nos dois lados de forma independente [77]. Na segunda fase, os dispositivos que estão a emparelhar, têm concordar com uma chave temporária. De uma das formas descritas acima, *Just Works* ou *Passkey Entry* ou *Out of Band*.

A terceira e ultima fase é opcional e apenas realiza uma conexão que é encriptado devido à chave STK criado na fase anterior. Nesta fase, o mestre e escravo podem partilhar as chaves de segurança:

***Long Term Key* (LTK)** LTK é uma chave de 128bits e é utilizado para gerar uma conexão encriptada. O LTK é fornecido do *host* para o controlador quer o dispositivo seja mestre ou escravo. Os controladores do quer sejam de dispositivos mestres ou escravos usam uma combinação de LTK, EDIV e Rand para encriptar a ligação.

***Encrypted Diversifier* (EDIV) e *Random Number* (Rand)** O EDIV é um valor de 16bits para identificar o LTK. O EDIV é gerado sempre que o LTK é distribuído. Rand é um valor de 64bits usado para identificar a LTK. Um novo Rand é gerado sempre que um LTK é distribuído. Uma combinação de LTK, EDIV e Rand permite que seja criado uma ligação encriptada.

***Ientity Resolution Key* (IRK)** IRK é uma chave de 128bits utilizada para gerar endereços aleatórios. Endereço aleatório é um recurso de privacidade que foi introduzido no BLE. Isso permite que um dispositivo para use um endereço aleatório diferente, de modo que é difícil de controlar esse dispositivo.

***Connection Signature Resolving Key* (CSRK)** BLE fornece o recurso para assinar os dados. Para assinar os dados, o dispositivo de envio anexa uma assinatura de 12 octetos, posteriormente o dispositivo que recebe a mensagem verifica a assinatura de modo certificar se os dados têm origem de uma fonte confiável. CSRK é uma chave de 128bits utilizada para assinar dados.

4.2.9 *Generic Access Profile*

Na camada mais alta do BLE, o *Generic Access Profile* define quatro funções com requisitos específicos para o controlador subjacente: *Broadcaster*, observador, periférico e central. Um dispositivo no papel de *Broadcaster* apenas transmite dados (através dos canais de *advertising*) e não suporta conexões com outros dispositivos. O papel do observador é complementar o *Broadcaster*, ou seja, tem a finalidade de receber os dados transmitidos pela *Broadcaster*. O papel do central é iniciar e gerir varias conexões, enquanto o papel periférico é projetado para dispositivos que utilizam uma única conexão com um dispositivo no papel de central. Um dispositivo pode suportar vários papéis, mas apenas um papel pode ser tomado num determinado momento.

4.3 Pacotes

Os dados transmitidos a partir de um dispositivo BLE tem formato de acordo com a Especificação Bluetooth Core. O pacote tem 4 campos: *Preamble* (1

byte), *Access Address* (4 bytes), PDU (2 a 39 bytes) e o CRC (3 bytes). É então apresentado a Figura 4.4 que exemplifica esses 4 campos.

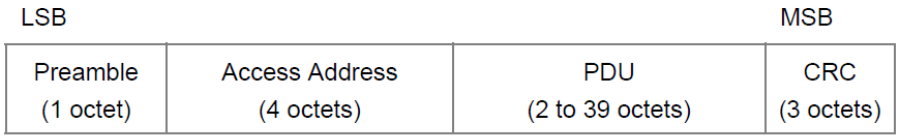


Figura 4.4: Formato geral de um pacote BLE [57]

O tamanho mínimo de um pacote de um dispositivo BLE é de 80 bits e o maior possível a ser transmitido é de 376 bits [57].

O *Preamble* é um campo utilizado para executar a sincronização de frequência, estimar o sincronismo e para o controlo automático do ganho.

O campo *Access Address* nos pacotes do canal de *advertising* é fixo com um valor de 0x8E89BED6 [57]. Nos canais de dados deve ser diferente para cada conexão. O *Access Address* é um valor de 32 bits aleatório, gerado pelo dispositivo no estado de iniciação por parte de quem realiza o pedido de ligação.

Quando um pacote é transmitido em um canal de *advertising*, o PDU é diferente. Estes dois tipos são descritos na secção 4.3.1 e 4.3.2, respetivamente.

A parte final do pacote transmitido está o campo CRC. CRC é um código de deteção de erros utilizado para validar o pacote. Ele garante a integridade dos dados para todos os pacotes transmitidos.

4.3.1 PDU no canal de *advertising*

O PDU no canal de *advertising* tem um cabeçalho (*header*) de 16 bits e um *payload* de tamanho variável (figura 4.5).

O campo *header* contém seis subcampos (figura 4.6).

O campo PDU Type indica qual é o tipo de PDU. Os tipo possíveis de PDU estão apresentados na tabela 4.2

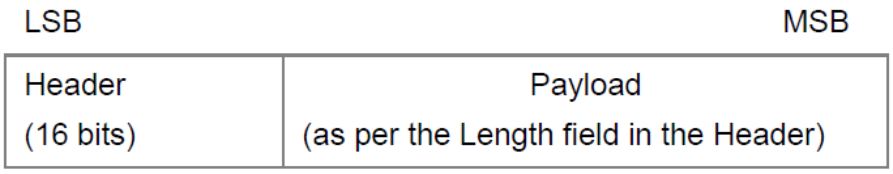


Figura 4.5: Formato geral do PDU no canal de *advertising* [57]

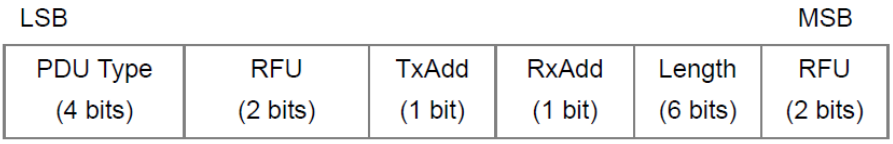


Figura 4.6: Formato do *header* do PDU no canal de *advertising* [57]

Os campos TxAdd e RxAdd do PDU canal de *advertising* contêm informações específicas para o tipo PDU definido. Se estes campos não forem definidos, então eles serão considerados como *Reserved Future Use* (RFU).

O campo *lenght* indica o comprimento do *payload* em octetos. Este pode ir de 6 a 37 octetos.

4.3.2 PDU no canal de dados

O PDU no canal de dados tem um cabeçalho (*header*) de 16 bits, um *payload* de tamanho variável e um campo *Message Integrity Check* (MIC) (figura 4.7). Deve-se salientar que este ultimo campo não está incluído quando a conexão não é encriptada ou com uma conexão encriptada mas com o *payload* igual a zero. O formato do campo *payload* depende do campo LLID presente no campo *header*.

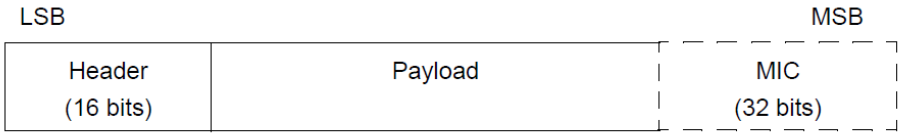


Figura 4.7: Formato geral do PDU no canal de *advertising* [57]

Tabela 4.2: Tipos de PDU (adaptado de [57])

Tipo do PDU	Nome do pacote
0000	ADV_IND
0001	ADV_DIRECT_IND
0010	ADV_NONCONN_IND
0011	SCAN_REQ
0100	SCAN_RSP
0101	CONNECT_REQ
0110	ADV_SCAN_IND
0111 - 1111	Reservado

O campo *header* contém cinco subcampos (figura 4.8).

Header						
LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	RFU (3 bits)	Length (5 bits)	RFU (3 bits)

Figura 4.8: Formato do *header* do PDU no canal de *advertising* [57]

Quando campo LLID é 01b ou 10b, o campo *Payload* irá conter dados do tipo "LL Data PDU". Se o campo LLID é 11b, então, o campo *payload* contém Dados do tipo "LL Control PDU".

O campo NESM dá indicação do próximo número de sequência esperado, ou seja, é usado pelo recetor para reconhecer o último PDU enviado ou para requisitar um novo envio do mesmo. O campo SN contém o número de sequência usado para identificar os pacotes enviados pela camada de ligação [57].

O bit de MD é utilizado para indicar que o dispositivo tiver mais dados para enviar. Se nenhum dispositivo definiu o bit MD nos seus pacotes, o pacote a partir do escravo fecha o evento de conexão. Se um ou ambos os dispositivos de ter definido o bit MD, o mestre pode continuar o evento de conexão através do envio de um outro pacote, e o escravo deve escutar depois de enviar o seu pacote. Se um pacote não for recebido pelo mestre

vindo do escravo, o mestre irá fechar o evento de conexão. Se um pacote não for recebido pelo escravo vindo mestre, o escravo vai fechar o evento de conexão.

O campo de *lenght* de 5 bits indica o tamanho do *payload* e MIC se existir. A gama de valores é 0 a 31 bytes. O campo *payload* deverá ser inferior ou igual a 27 bytes. O MIC tem um tamanho fixo de 4 bytes.

Capítulo 5

Redes de sensores com fios

Neste capítulo irá ser apresentado os vários protocolos de comunicação com fios diferentes que são utilizados neste projeto. Apresentando algumas das características mais relevantes e explicando o seu funcionamento.

5.1 Protocolo I2C

A empresa Philips (agora NXP Semiconductors) [79], desenvolveu um protocolo de comunicação bidirecional de apenas 2 fios, o I2C. Originalmente o I2C foi projetado para conectar um número pequeno de dispositivos com uma velocidade máxima de 100 kbps. Em 1992, a velocidade padrão do barramento foi aumentada para 400 kbps, para acompanhar os requisitos de desempenho dos novos microcontroladores que iam surgindo. A especificação *I2C* em 1998, aumentou a velocidade máxima para 3.4 Mbit/s.

5.1.1 Tipologia

O barramento I2C necessita de dois sinais, o SCL e o SDA. O SCL é o sinal de relógio, e o SDA é o sinal de dados. Um barramento I2C pode suportar até 112 dispositivos com endereços de 7 bits e 1008 com endereços de 10 bits [80].

A figura 5.1 apresenta a tipologia da rede I2C. Os sinais SCL e SDA são em *open-drain*, portanto, estão conectados a uma tensão de alimentação positiva através das resistências de *pull-up*. Isso significa que os dispositivos *I2C* só podem impor o nível lógico zero no barramento. Quando nenhum dispositivo está a escrever no barramento, este ficará no nível alto através da resistência de *pull-up*.

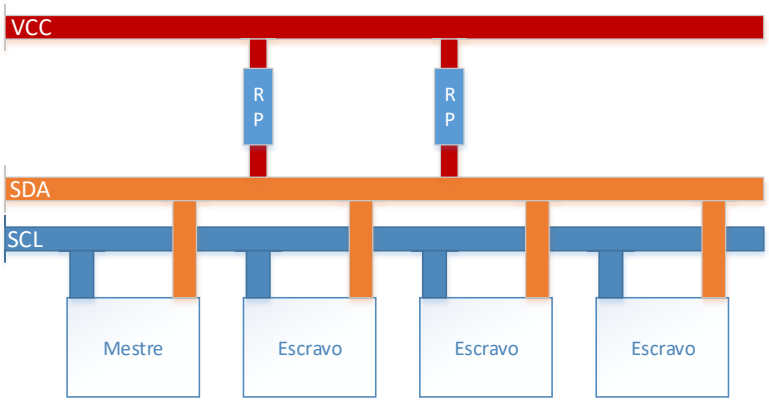


Figura 5.1: Tipologia da rede I2C

Antes de qualquer transação no barramento existe uma ordem de *Start* e uma ordem de *Stop*, presente na figura 5.2. Estas ordens são impostas pelo dispositivo mestre da rede, no caso da ordem de *Start* serve para informar a todos os dispositivos escravos que algo está prestes a ser transmitido no barramento. Consequentemente, todos os dispositivos escravos conectados escutarão a linha de dados (SDA). A ordem de *Start* é emitida forcando a linha SCL a nível lógico alto e a linha SDA a nível lógico baixo. Quando a transferência de dados for concluída, o dispositivo mestre envia uma ordem de *Stop* para informar aos outros dispositivos que a trama de dados acabou. A sinalização utilizada como ordem de *Stop* é definida por uma transição do nível lógico baixo para o nível lógico alto, na linha SDA, após uma transição do nível lógico baixo para o nível lógico alto na linha SCL, com SCL restante

a nível lógico alto.

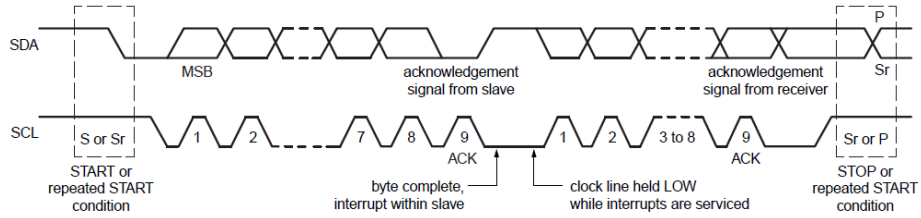


Figura 5.2: Trama de dados do protocolo I2C [79]

5.1.2 Velocidade

Originalmente, o I2C estava limitado a 100 kbit/s. Ao longo do tempo, houve várias alterações, de maneira que agora existem cinco categorias de velocidade operacional, e são elas [79]:

- *Standard Mode*;
- *Fast Mode*;
- *Fast Mode Plus*;
- *High Speed Mode*;
- *Ultra Fast Mode* (unidirecional);

Qualquer dispositivo pode ser funcionar a uma velocidade de barramento inferior. Dispositivos de modo ultra rápido não são compatíveis com versões anteriores, pois o barramento é unidirecional [79]. Na tabela 5.1 é possível ver a as velocidades de comunicação de cada modo.

5.2 Protocolo OneWire

OneWire foi um sistema originalmente desenvolvido pela empresa Dallas Semiconductor, atualmente Maxim [81]. O objetivo foi fornecer uma forma

Tabela 5.1: Tabela comparativa dos modos de funcionamento

Modo	Data Rate
Standard Mode	100 kbit/s
Fast Mode	400 kbit/s
Fast Mode Plus	1 Mbit/s
High Speed Mode	3.4Mbit/s.
Ultra Fast Mode (unidirecional)	5 Mbit/s

simples e eficaz de comunicar entre microcontroladores e periféricos, como sensores, utilizando o mínimo número de fios possíveis. Daí, como o nome sugere, *OneWire* (um fio). Os dispositivos que utilizam este protocolo requerem apenas 1 pino do microcontrolador e uma ligação de massa em comum, para que seja possível uma comunicação bidirecional.[82]

5.2.1 Tipologia

Relativamente à tipologia da rede, como qualquer outro tipo de rede, existem várias tipologias diferentes. Os dois tipos mais utilizados para o protocolo *OneWire* são o barramento e estrela.

A tipologia mais simples de rede é o barramento. Fisicamente consiste num cabo com sensores nele ligado. A Figura 5.3 mostra um exemplo. Uma vantagem de uma rede em barramento é a simplicidade elétrica e o desempenho da transmissão de dados. As redes em barramento geralmente são imunes a ruído elétrico, portanto, as comunicações são mais robustas que as redes em estrela. As redes em barramento são de fácil implementação, no entanto, nem sempre é conveniente encadear um cabo em torno dos locais de todos os sensores. Além disso, nas redes em barramento qualquer interrupção no cabo do barramento causará uma falha, essa falha irá incapacitar os sensores posteriores à quebra. Qualquer curto circuito no barramento afetará todos os sensores do mesmo. [82]

A tipologia em estrela é um pouco mais complexa do que em barramento. Uma rede em estrela típica é mostrada na figura 5.4. Uma vantagem de uma

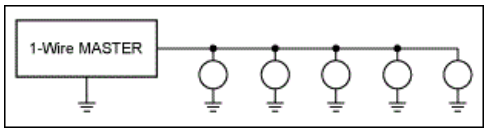


Figura 5.3: Tipologia *OneWire* em barramento [83]

rede em estrela é que se torna mais fácil conectar fisicamente os periféricos. Uma interrupção da conexão física do sensor não afetará outros sensores. A maior desvantagem de uma rede em estrela é o ruído elétrico. Para colmatar esse facto é necessário adicionar uma resistência em série a todos os ramos da rede, para minimizar as reflexões do sinal. Essas resistências reduzem as reflexões e consequentemente o ruído elétrico tornando assim as comunicações mais robustas. [82]

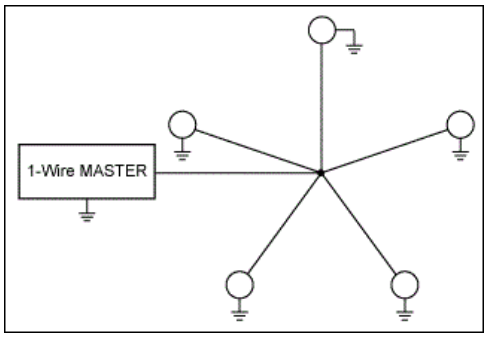


Figura 5.4: Tipologia *OneWire* em estrela [83]

5.2.2 Implementação através de *software*

Existem quatro operações básicas, são *Reset*, *Write* 1 bit, *Write* 0 bit e *Read* bit. O tempo necessário para executar um bit de comunicação é chamado de *time slot*. Na tabela 5.2 é apresentado uma breve descrição de cada operação e uma lista das etapas necessárias para gerá-la.

Tabela 5.2: Operações possíveis no protocolo *OneWire* (adaptado de [84]

Operação	Descrição	Implementação
Write 1 bit	Enviar bit 1 para o periférico *OneWire*	-Impor nível baixo durante 6uS -Libertar o barramento durante 64uS
Write 0 bit	Enviar bit 0 para o periférico *OneWire*	-Impor nível baixo durante 60uS -Libertar o barramento durante 10uS
Read bit	Ler um bit do periférico *OneWire*	-Impor nível baixo durante 6uS -Libertar o barramento durante 9uS -Ler o bit enviado pelo *slave* -Esperar 55uS
Reset	Reiniciar o barramento	-Esperar entre 0 a 2.5uS -Impor nível baixo durante 480uS -Libertar o barramento durante 70uS -Ler bit, se 0 = dispositivo presente, se 1=nenhum dispositivo presente -Esperar 410uS

5.3 Protocolo CAN

O protocolo CAN, é um barramento *standard* desenvolvido para permitir a microcontroladores e outros dispositivos comunicar entre eles em aplicações sem um módulo "mestre", isto é, neste tipo de protqocolo não existe a teoria de *master-slave* criada na maioria de comunicações, e devido a esta e outras características, o CAN é um tipo de comunicação segura e robusta.

Em sistemas que fazem uso deste protocolo as informações são transmitidas em tempo real, e daí surge a necessidade de um controlo mais rígido em relação a erros nas mensagens e na sua receção, isto é, a eficácia do protocolo deve ser tal que a troca de mensagens entre os dispositivos pertencentes à rede nunca deve falhar. As mensagens que percorrem os nós de uma rede deste tipo são geradas por *broadcast*, e fazem uso do conceito de comunicação *broadcasting multi-master*.

5.3.1 Características

A utilização de um protocolo de comunicação como o CAN traz ao utilizador uma série de vantagens em relação a outros tipos de comunicação, e de todas as que existem, pode-se destacar as seguintes:

- Capacidade multi-mestre - O CAN é um protocolo de comunicação série que permite controlo distribuído em tempo real e o barramento disponível para as comunicações possui a capacidade de atender vários nós com pedidos de acesso ao meio de transmissão, daí a sua capacidade de multi-mestre.

- Processo de arbitragem não destrutiva - O endereçamento dos destinatários não é utilizado no CAN como no sentido convencional, pois este protocolo utiliza identificadores nas mensagens transmitidas. Isto resulta que uma mensagem que esteja a circular na rede vai ser recebida por um ou mais dispositivos que decidem, com base no identificador recebido, se devem ou não processar esta mesma mensagem. O identificador da mensagem é também responsável por determinar a prioridade da mensagem que compete com todas as outras pelo acesso ao barramento.

- Capacidade *broadcast* - O conceito de *broadcast* no CAN pressupõe que a transmissão de uma mensagem possa ser efetuada a um conjunto de recetores em simultâneo.

- Elevadas taxas de transferências - A taxa máxima de transmissão especificada é de 1 Mbit/s, e está associada a sistemas com barramentos de comprimento até 40 m, e ao aumentar a distância dos barramentos, esta mesma taxa diminui.

- Redução do número de fios - O barramento da rede CAN apenas é constituída por dois fios entrelaçados que correspondem ao CAN_H

e CAN_L, do inglês CAN_HIGH e CAN_LOW, e resistências nos extremos do barramento o que torna este protocolo muito simples de implementar numa rede, em relação ao *hardware*.

- Máximo de 8 bytes de informação útil - O facto da informação transmitida numa rede CAN ser de tamanho curto faz com que este sistema seja ideal para a ligação com subsistemas inteligentes, tais como sensores e atuadores. Uma mensagem CAN pode conter no máximo 8 bytes de informação útil, embora seja possível, através de segmentação, transmitir blocos de dados superiores.

- Deteção e Sinalização de Erros - O controlador CAN presente em cada nó utiliza mecanismos capazes de registar os erros ocorridos, avaliando-os estatisticamente, de forma a desencadear ações com eles relacionados.

- Flexibilidade e Confiabilidade - A flexibilidade neste protocolo está presente no facto de ser possível adicionar novos nós a uma rede CAN sem requerer alterações tanto da parte do *software* como de *hardware* dos restantes nós, no caso de o novo nó não ser emissor ou não necessitar de transmissão de dados adicionais.

- Baixo custo - Devido ao facto de a maioria destes sistemas que possuem uma rede CAN sejam constituídos por pouco *hardware* (microcontrolador, *transceiver*, sensores e atuadores) faz com que a implementação de sistemas deste tipo seja de mais baixo custo que sistemas de outros tipos.

5.3.2 Funcionamento

Neste subcapítulo o objetivo é explicar o funcionamento do protocolo CAN, Para isso irá se falar sobre a arquitetura do sistema bem como as suas camadas envolventes.

Face ao modelo *International Organization for Standardization* (ISO) - *Open Systems Interconnection* (OSI), a especificação da rede CAN descreve apenas a camada física e a camada de ligação de dados apresentado na figura 5.5, ambas implementadas em *hardware*. Para facilitar a programação, e devido aos diferentes cenários de utilização, foram desenvolvidas diversas camadas de *software* que se posicionam na camada de aplicação.

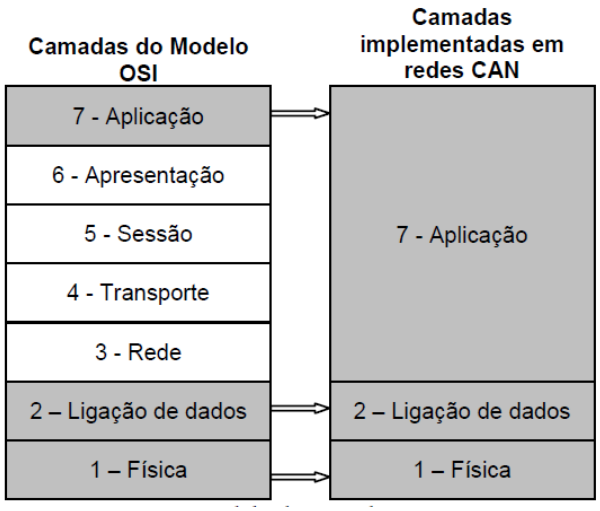


Figura 5.5: Modelo de camadas OSI e CAN.

Camada física

A camada física de um sistema de comunicação cobre os aspetos da transmissão física dos dados (bits) entre os nós da rede. Nesta camada destacam-se três subcamadas [85]:

- *Physical signaling* (PLS), implementada no controlador CAN

  Codificação/descodificação de bits;

  Timing dos bits;

  Sincronização entre nós;

- *Physical Medium Attachment* (PMA)

  Características do *transceiver*;

- *Medium Dependent Interface* (MDI)

  Características do meio de transmissão (cabo, fichas, etc);

Camada de ligação de dados

A camada de ligação de dados, DLL (*Data Link Layer*) impõe o formato das mensagens que viajam no barramento e fornece mecanismos de deteção e prevenção de falhas. Existe uma subdivisão desta camada de forma a distinguir níveis de serviço fornecidos [86]:

- *Logical Link Control* (LLC)

  Filtro de aceitação de mensagens;

  Notificação de *overload*;

  Gestão de recuperação de mensagens ou de situações de erro;

- *Medium Access Control*(MAC)

  Encapsulamento/desencapsulamento dos dados nas mensagens;

  Codificação das mensagens (*stuffing*/*destuffing*);

  Gestão de acesso ao meio;

  Deteção e sinalização de erros;

  Confirmação da integridade das mensagens (*acknowledge*);

Filtro de aceitação de mensagens: Na subcamada LLC da DLL existe um mecanismo que filtra as mensagens que recebe, ou seja, existe uma ou várias máscaras de bits de comprimento variável que são sobrepostas (AND) com os n primeiros bits do identificador da mensagem, de forma a decidir se a mensagem é entregue à camada acima ou descartada. Este

mecanismo é programável pelas camadas superiores ou, em ultima análise, pelo utilizador [86].

Stuffing/destuffing: Sempre que o transmissor detetar a presença de 5 bits consecutivos da mesma polaridade numa trama de bits (após a serialização), introduz um bit de polaridade inversa que vai ser automaticamente removido pelos recetores. Esta regra de *stuffing* pode ser utilizada para uma verificação de integridade.

Na construção de uma mensagem, o nó transmissor preenche o cabeçalho e a terminação, executa o processo de *stuffing* em determinadas zonas da mensagem e calcula o CRC (*Cyclic redundancy check*). Nesta altura, a mensagem está pronta para ser enviada.

Nos nós recetores o processo é inverso. Após a receção da mensagem sem erros detetados no barramento, esta é *destuffed*, os campos de correção de erros são verificados, passa pelo filtro de aceitação e, caso seja essa a decisão, os bits podem ser novamente agrupados em bytes e enviados à camada acima [86].

Difusão: O conceito de difusão da rede CAN significa que todos os nós veem as mensagens transmitidas por qualquer um deles, como pode ser visualizado na figura 5.6. Após a receção de cada mensagem, cabe aos nós a decisão de descarte ou entrega da mensagem. [86]

Acesso múltiplo: Um dos requisitos de uma rede CAN é a possibilidade de acesso múltiplo ao barramento, ou seja, vários nós podem ler do barramento ao mesmo tempo mas, para evitar colisões (dois ou mais nós a enviar ao mesmo tempo uma mensagem diferente, o que resultaria numa mensagem inválida), é preciso um mecanismo de prioridades. O mecanismo utilizado é o *Carrier Sense Multiple Access/Deterministic Collision Resolution* (CSMA/DCR) [86].

Cada mensagem leva consigo um identificador, uma identidade que é atribuída pelo nó. Quando o barramento está em estado *idle*, vários nós

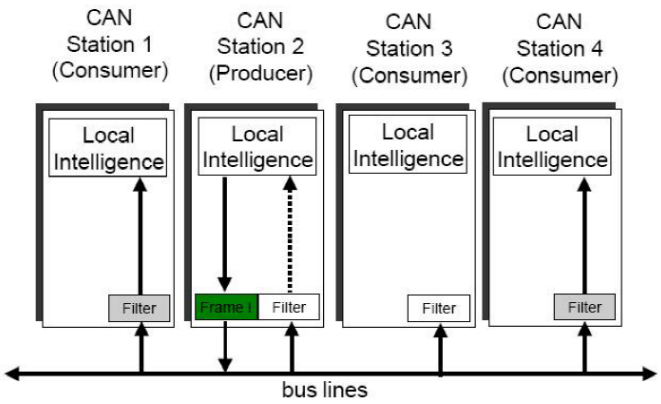


Figura 5.6: Difusão de mensagens CAN [86]

podem iniciar as suas transmissões de dados ao mesmo tempo. Cada nó lê do barramento a cada bit que envia e compara os valores. Caso o bit que o nó tenha tentado escrever seja recessivo, será lido do barramento um bit recessivo se e só se todos os nós que estão a transmitir na altura estiverem a enviar um bit recessivo. Por definição, o *transceiver* garante que um bit dominante permanece no barramento em detrimento de um bit recessivo. Para melhor exemplificação deste sistema é apresentado um exemplo na figura 5.7.

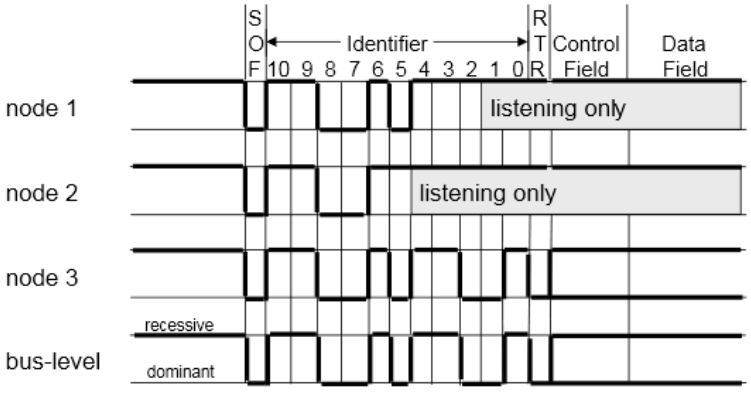


Figura 5.7: Sistema de anti-colisões CSMA/DCR

Durante a transmissão do identificador da mensagem, o algoritmo CS-

MA/DCR decide de forma não destrutiva e sem atrasos ou retransmissões quem transmite ou não a mensagem. No funcionamento do algoritmo CSMA/DCR em que n nós tentam escrever no barramento, e assumindo que todos tentam enviar mensagens diferentes, n-1 nós vão desistir de transmitir a mensagem para que o restante nó possa transmitir a dele. Depois da transmissão da mensagem do nó que "ganhou" o barramento, os outros nós tentam novamente transmitir as suas mensagens. Com a utilização do algoritmo CSMA/DCR, é possível estabelecer ordens de prioridade nas mensagens, porque estas têm identificadores programáveis [86].

A figura 5.7 mostra como o algoritmo CSMA/DCR funciona. Como se pode verificar temos 3 nós (1,2,3) e o barramento (n).Cada um dos nós está a tentar enviar uma mensagem simultaneamente. No bit 5 do identificador, o nó 2 percebe que enviou um bit recessivo mas leu do barramento um bit dominante e desiste de enviar a sua mensagem, este fica em modo de escuta apenas. No bit 2 do identificador, o nó 1 também entende que enviou um bit recessivo mas leu um bit dominante. Também este nó desiste de transmitir a mensagem. Desta forma, o nó 3 prevalece e transmite a sua mensagem.

Trama de dados: Até agora referenciadas genericamente como mensagens, as tramas de dados são as unidades de comunicação entre as camadas de ligação de dados dos vários nós. Como pode ser visualizado na figura 5.8, é apresentado a trama de dados, no caso *standard* de 11 bits de identificador.

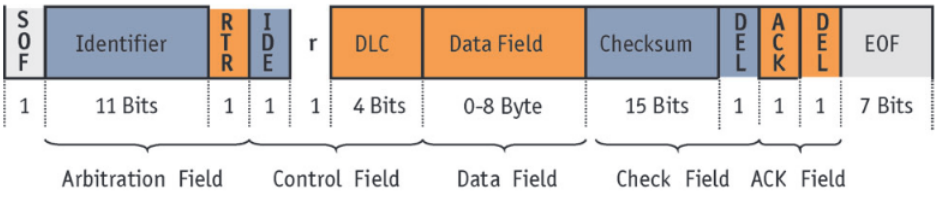


Figura 5.8: Trama de dados [87]

Sempre que um nó pretende enviar dados para o barramento, é feito através do encapsulamento da informação numa ou mais tramas de dados,

em que cada uma pode transportar 1 a 8 bytes de informação. O trama de dados começa com um bit *Start of Frame* (SoF) para a sincronização em todos os nós. Depois do SoF está o campo de identificação (*arbitration field*) que identifica a prioridade e o conteúdo da mensagem. O campo de controlo (*control field*) especifica o comprimento em bytes da mensagem. O campo de dados (*data field*) contém a informação propriamente dita e o campo de CRC (cyclic redundancy check) guarda informação dedicada para deteção de erros em determinadas zonas da trama. O campo de confirmação (Acknowledge field - ACK) é utilizado para o transmissor entender se pelo menos um nó recebeu a mensagem sem erros de barramento. A mensagem é terminada pelo *End of Frame* (EoF). Atualmente, existem dois tipos de tramas: tramas de dados *standard* e as *extended*, como referido anteriormente. Os dois tipos diferem no significado de algumas *flags* e no comprimento do campo de identificação [88].

Controlo de Erros: Na ocorrência de um erro, é enviada uma trama de erro (*error frame*). A trama é composta por uma *flag* de 6 bits dominante e é seguida de sobreposições dos outros nós, pois é possível que nem todos detetem o erro exatamente no mesmo bit. No fim existe o delimitador do erro que é composto por 8 bits recessivos e permite aos nós recomeçar a transmissão após a ocorrência de um erro.

Os erros que podem surgir e causar a geração de uma trama de erro são [89]

- Erros de *stuffing*: Quando o recetor deteta mais do que 5 consecutivos no mesmo estado, sejam recessivos ou dominantes. Esta regra é apenas aplicada entre o SoF e o CRC *field* inclusive.

- Erros de bit: Sempre que um nó escreve no barramento ele lê, por assim dizer o que escreveu. Quando um nó escreve no barramento mas a leitura é diferente da escrita então é detetado um erro de bit.

- Erro de formato: Este erro ocorre quando o formato da trama não é o esperado, por exemplo, como já foi dito anteriormente no EoF, este campo é composto por sete bits recessivos, se por ventura o EoF conter um bit dominante significa que o formato da trama não é o esperado.

Camada de aplicação

A camada de aplicação não está explicitamente definida para o CAN. Está em aberto para que os utilizadores definam os seus algoritmos de acordo com as características do sistema quando usando CAN.

Esta página foi intencionalmente deixada em branco.

Capítulo 6

Arquitetura e Projeto do Sistema

Neste capitulo, são abordados vários aspetos referentes à arquitetura do sistema a implementar. Primeiramente será feito uma pequena análise dos requisitos, seguidamente é exposto o diagrama de blocos do sistema a ser implementado. Posteriormente serão descritos os elementos da rede. Por fim, é apresentado qual a escolha das tecnologias para este projeto. Também importante neste capitulo é o teste realizado com o protocolo CAN para descobrir as mensagens referentes à velocidade instantânea e as rotações do motor do veículo.

6.1 Análise de Requisitos

A arquitetura do sistema é composta essencialmente por uma rede de sensores sem fios interface CAN cujo objetivo é monitorizar os dados relevantes de uma viagem de um veículo (posição geográfica, velocidade do veículo, temperatura da carga, entre outras), recorrendo a uma tecnologia de baixo consumo de energia. Para isso é necessário analisar todos os requisitos do sistema de modo que os objetivos sejam cumpridos.

6.1.1 Requisitos

- A tecnologia sem fios utilizada terá que ser de baixo consumo energético para que a autonomia seja o mais elevado quanto possível.
- O sistema deverá ser flexível para que a inserção de um novo *Slave* no sistema seja feita de forma autónoma.
- O sistema terá que possuir comunicação com o exterior, por exemplo um servidor.
- Os sensores incluídos nos Nós *Slaves* deverão ser precisos de forma a não comprometer todo o sistema.
- O sistema deverá possuir um sensor GPS para determinar o posicionamento geográfico do veículo.
- A comunicação entre o veículo e o sistema será feita através do protocolo CAN.

Além disto, o sistema tem como requisito que este atenda as necessidades do IoT, como o encaminhamento das mensagens para um servidor. Tornando assim a possibilidade deste sistema ser monitorizado remotamente.

Estes são os principais requisitos do sistema, obviamente que outros também serão considerados como por exemplo, facilidade de uso, fiabilidade e durabilidade.

6.2 Arquitetura Geral do Sistema

A arquitetura do sistema vê-se como um sistema de vários sensores posicionados dentro de um veículo de modo a monitorizar a sua carga. Na figura 6.1 mostra um exemplo como fisicamente será implementada. Cada nó contém um sensor acoplado de modo a realizar a aquisição de dados de uma determinada grandeza. Após essa informação ser captada, é enviada via BLE

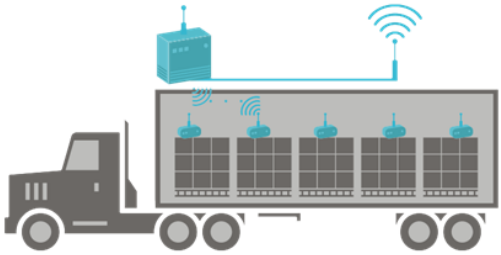


Figura 6.1: Ideia base da arquitetura do Sistema [90]

para o Nó *Master*. O Nó *Master* tem a função de receber a informação dos Nós *Slaves* e fazer o *bypass* da informação para a Unidade Central. A Unidade Central tem um GPS para determinar a sua localização. Aliado a isso existe uma ligação à rede CAN do veículo onde é retirada informação referente à velocidade e às rotações do motor. Tendo a Unidade Central toda a informação do sistema, a função do mesmo é reencaminhar para a *cloud*. O diagrama pode ser visualizado na figura 6.2.

O número limite de Nós *Slaves* não está por si só definido no padrão do BLE. É difícil de quantificar quantos *Slaves* é possível ligar pois isso depende do intervalo de conexão. O número limite de Nós *Slaves* é difícil de definir. Cada *Slave* ocupa um pouco da largura de banda dos canais de *advertising* e de dados, ora se os canais de *advertising* e de dados ficarem totalmente preenchidos é impossível albergar mais *slaves*. Para validação do projeto irá se incluir apenas 5 Nós *Slaves*. De modo a simplificar e definir como irão ser chamados estes nós é apresentada a tabela 6.1. A tabela serve também para definir o nome de cada nó para uma melhor explicação daqui em diante.

6.2.1 Nós *Slaves*

Atualmente o sistema dispõe de 5 Nós *Slaves*. Eles têm a capacidade de mensurar uma grandeza física especifica. Essa grandeza muda de Nó *Slave*

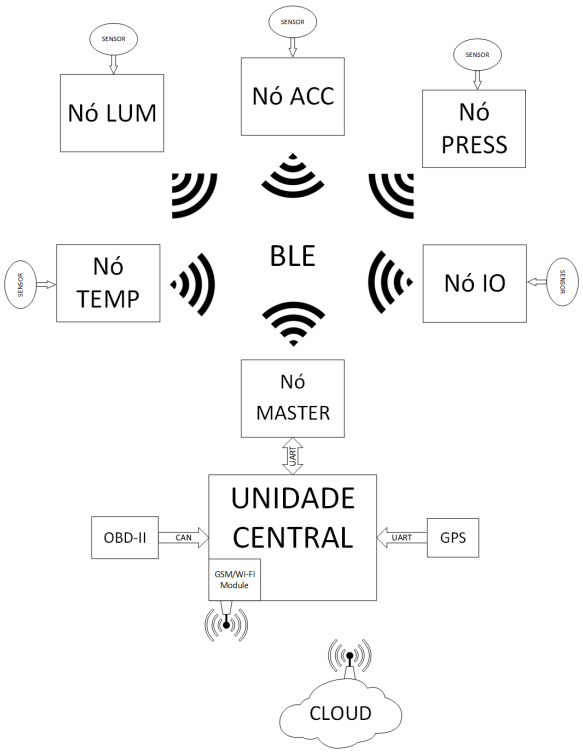


Figura 6.2: Arquitetura do Sistema

para Nó *Slave*, seguidamente irá ser apresentado cada um deles.

O Nó TEMP que tem como capacidade medir a temperatura, para isso utiliza o sensor DS18B20 da Maxim Integrated. Este sensor tem como interface, o protocolo OneWire. OneWire é um protocolo de comunicação série entre um dispositivo mestre e um ou mais escravos, todos partilham o mesmo barramento, barramento este de apenas um fio, daí a origem do nome OneWire [91].

A aquisição dos dados é realizada de forma periódica. A temperatura é uma grandeza física de variação lenta, isto significa que a frequência de aquisição de dados pode ser baixa. O facto da frequência de aquisição ser baixa permite poupar energia o que vai ao encontro dos objetivos do projeto.

O Nó LUM está responsável por realizar a aquisição de dados referentes à luminosidade e utiliza um LDR para o efeito. Este é ligado através de

Tabela 6.1: Nós *Slave* na rede BLE

Nº	Nome	Grandeza	Sensor	Interface
1	Nó TEMP	Temperatura	DS18B20	One-wire
2	Nó LUM	Luminosidade	LDR	ADC
3	Nó ACC	Aceleração	MPU-6050	I2C
4	Nó PRESS	Pressão atmosférica	BMP280	I2C
5	Nó IO	N/A	Tact Switch	Digital

um divisor de tensão a uma entrada analógica do módulo nrf51822. Este nó requer uma pré-calibração da luminosidade máxima e mínima que irá estar sujeito, essa calibração é feita através de dois botões presentes na placa pelo utilizador. A aquisição de dados é feita de forma periódica, à semelhança do Nó TEMP. Os dados serão enviados para o *Master* também de forma periódica.

O objetivo do Nó ACC é monitorizar as acelerações geradas pelo veículo durante a sua viagem. O sensor utilizado é o MPU-6050 da InvenSense, o protocolo de comunicação é o I2C. Este sensor apresenta 3 eixos, e serão todos contabilizados para que seja detetado alguma movimentação suspeita na carga durante o transporte.

O Nó PRESS foi pensado para casos onde há necessidade de contabilizar a pressão atmosférica. Para isso, o Nó PRESS, conta com um sensor da BOSCH, o BMP280. A comunicação é também feita através do I2C. A pressão atmosférica também é uma grandeza que varia pouco ao longo do tempo, significa que a frequência de aquisição de novos dados é baixa.

O Nó IO será o ultimo e terá um sensor com saída digital. Este sensor detetará o estado, por exemplo, de uma determinada porta. Por vezes é necessário haver um controlo de quantas vezes a porta do compartimento da carga é aberta para que se possa determinar uma possível violação da carga durante o transporte. Este sensor será simulado através de um *tact switch* que será ligado ao microcontrolador numa entrada digital.

6.2.2 *Master*

O Nó *Master* irá receber toda a informação dos Nós *Slaves* via BLE. Este nó está encarregue de centralizar a informação vinda dos Nós *Slaves* e posteriormente enviar essa informação via UART para a Unidade Central. A informação que é enviada para a Unidade Central contém dois dados relevantes, qual a origem da informação e a informação propriamente dita.

Sempre que um Nó *Slave* envie uma nova leitura para o Nó *Master*, este vai encaminhar essa informação para a Unidade Central mantendo assim os dados na Unidade Central sempre atualizados.

6.2.3 Unidade Central

Este elemento é o cérebro da toda a rede. A ele está conectado um sensor de GPS que tem como função determinar a localização de forma periódica. Tem também disponível uma porta série para ligar um modulo que faça a interface entre este sistema e a *cloud*. Contém ainda uma ligação ao veículo através do conector *On-Board Diagnostic* (OBD-II) utilizando o protocolo CAN.

Esta unidade, irá receber via UART os dados do Nó *Master*. Dados estes que contêm a informação atualizada dos nós *Slaves*. Juntamente a estes dados, esta unidade irá adquirir a sua localização geográfica através do sensor GPS também através do protocolo UART. Para além disso, este elemento irá recolher a informação relativa à velocidade instantânea e às rotações por minuto do motor. Essa recolha será feita com o protocolo CAN.

Recolha de dados do veículo via CAN

É necessário ter em conta que os fabricantes de automóveis não disponibilizam os ID's das mensagens que circulam na rede CAN dos seus veículos. Esses ID's mudam de fabricante para fabricante. Para descobrir quais são

os ID's relativos à velocidade e às rotações foi feito engenharia inversa, ou seja, ligou-se um analisador lógico à rede CAN presente no conector OBD-II do veículo e seguidamente fez-se uma análise de todas as mensagens que circulam na rede. Os pinos do conector OBD-II foram os 6 e 14 e o bit rate é de 500kbps. O carro utilizado para este ensaio foi um Renault Kangoo do ano de 2009.

O primeiro ensaio foi aumentar a rotação até ao máximo e deixar que o motor volte ao ralenti, parar a captura e exportar os dados para uma folha de calculo. Foram feitos vários gráficos com o tipo de mensagens diferentes e já sabendo qual a forma que o gráfico iria tomar e que quais as grandezas de valores, foi relativamente simples de entender que os 2 bits menos significativos da mensagem com o ID 0x155 é correspondente à rotação do motor. Na figura 6.3 é a apresentado o gráfico do ensaio.

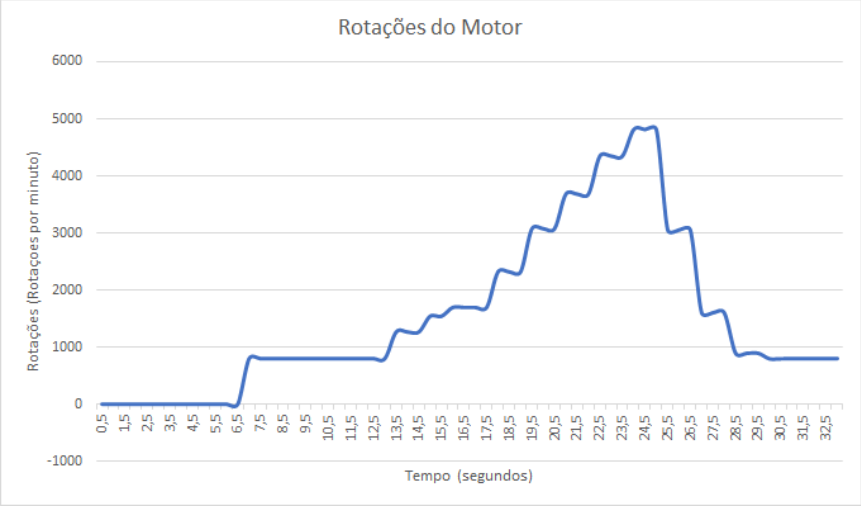


Figura 6.3: Gráfico da rotação

No outro ensaio, para determinar o ID da mensagem relativo à velocidade, o método foi idêntico ao anterior. Começar a captura com o veículo parado, arrancar e acelerar até aos 50 km/h depois travar de forma suave até imobilizar o veículo. Após realizar os devidos gráficos com a ajuda da

folha de calculo chegou-se à conclusão que a informação de velocidade está presente nos bits 3 e 4 na mensagem com o ID 0x155, o mesmo ID da mensagem para as rotações do motor. Como já se conhecia as rotações então nesta análise foram inseridas 2 linhas no mesmo gráfico, a das rotações e da velocidade. Na figura 6.4 é possível ver o gráfico da mensagem com a velocidade e a rotação e é percetível a troca de mudança com o aumento gradual da velocidade.

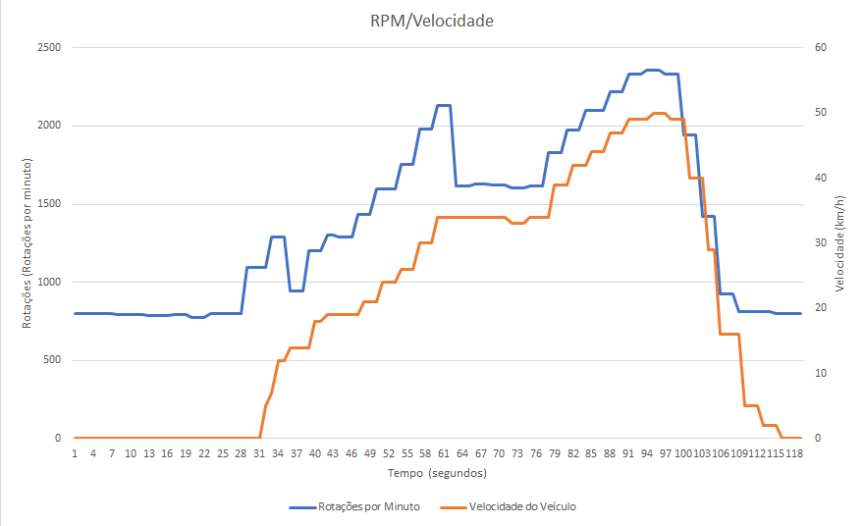


Figura 6.4: Gráfico da rotação com velocidade do veículo

Toda esta informação irá ser enviada para a *cloud*, recorrendo ao protocolo UART, de forma a poder ser consultada em qualquer parte do mundo. O objetivo é monitorizar o trajeto de um veículo numa entrega ao cliente.

6.2.4 *Cloud*

A informação de todos os elementos da rede é enviada para a *cloud*. Isso permite a monitorização remota de todo o sistema. Não é objetivo deste trabalho desenvolver a componente do servidor.

6.3 Escolha das Tecnologias

Seguidamente irá ser apresentada a escolha a nível de tecnologias para todo o sistema. Primeiramente é escolhido qual o melhor módulo BLE para este sistema e qual o microcontrolador para a Unidade Central e posteriormente os sensores.

6.3.1 Módulo BLE

A melhor abordagem para este sistema será a escolha de um SoC com BLE pois assim é possível obter um menor consumo de energia face à solução de microcontrolador + radio BLE.

Os grandes fabricantes deste tipos de *chips* apresentam soluções bastante competitivas. No caso da Texas Instruments oferece por exemplo o SoC CC2640, anunciado como *ultra low power*. Este SoC dispõe de um processador ARM Cortex-M3 a funcionar a 48 Mhz [92]. A Nordic Semiconductor também dispõe de diversas soluções. Uma solução interessante é o nRF51822 que possui um um processador ARM Cortex-M0 com vários periféricos disponíveis. Outra grande empresa na área dos semicondutores, a Dialog Semiconductor, apresenta a sua solução com o SmartBond DA14580. Contém um ARM Cortex-M0 à semelhança do nRF51822, mas este é apresentado como líder mundial de baixo consumo de energia [93]. Para ter uma vista geral dos vários SoC existentes no mercado é apresentado a tabela 6.2.

Na tabela apenas faz referência ao consumo energético enquanto transmite ou recebe informação, mas sabe-se grande parte do tempo o SoC não está a transmitir informação mas sim em modo *Low Power*, para que possa gastar o mínimo de energia possível. O SoC irá acordar de forma periódica ou quando ocorrer uma função de interrupção assim o obrigue.

Além das especificações técnicas é necessário ter em conta outros aspetos importantes, a disponibilidade e facilidade de aquisição do SoC, curva de aprendizagem, sem esquecer o custo, pois um dos grandes objetivos é tornar

Tabela 6.2: Comparativo entre vários SoC existentes no mercado

Marca	Nome	BLE	Processador	consumo RX/TX
Nordic Semiconductor	nRF51822	4.1	Cortex-M0	6.3mA / 8mA
Dialog Semiconductor	DA14580	4.1	Cortex-M0	4.9mA / 4.9mA
Texas Instruments	CC2540	4.0	8051	15.8mA / 18.6mA
Atmel	ATBTLC1000	4.1	Cortex-M0	3mA/3mA
Texas Instruments	CC2640	4.1	Cortex-M3	5.9 mA / 6.1 mA

esta solução económica.

Analisando estes fatores todos, chega-se à conclusão que a escolha mais acertada é o nRF51822 da Nordic Semiconductor. Isto porque este SoC apresenta um ARM Cortex de 32Bit com 256kB de memória *flash* + 16kB de memoria RAM. Na figura 6.5 é apresentado o diagrama de bloco do SoC. Este SoC apresenta um esquema de mapeamento de *General Purpose Input/Output* (GPIO) flexível de 31 pinos e permite que as entradas/saída, como PWM, interfaces UART, I2C, sejam mapeadas para qualquer pino do dispositivo, isto permite uma grande flexibilidade de *design*. Outro fator não menos importante é a facilidade de aquisição que é maior que os restantes concorrentes. Relativamente ao consumo de energia, de facto este SoC não é deveras o mais económico em termos de consumo enquanto transmite ou recebe informação. Mas por outro lado o SoC escolhido é o mais barato face às outras possíveis soluções.

6.3.2 Microcontrolador da Unidade Central

O microcontrolador que irá equipar a Unidade Central terá que ter interfaces para se ligar aos elementos da rede. Terá que ter um periférico CAN para comunicar com o veículo. Terá de possuir também 3 periféricos UART, pois

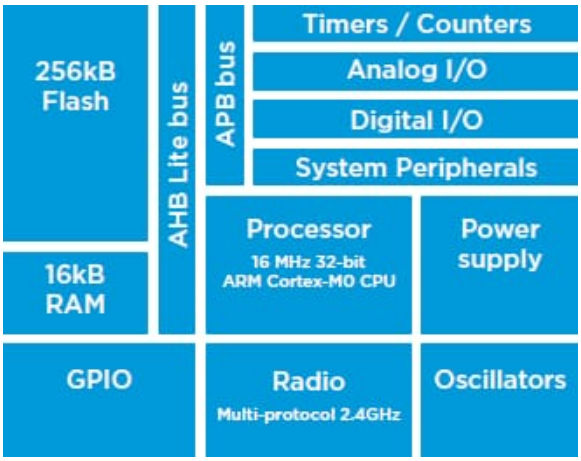


Figura 6.5: Diagrama de blocos do SoC NRF51822 [94]

o Nó *Master* e o GPS comunicam utilizando este protocolo, sobrando assim um periférico UART para a interface entre o sistema e a *cloud*.

Analisando os microcontroladores mais conceituados, chegou-se à conclusão que o STM32F103 será o mais indicado. Pois este contém 3 periféricos UART e um periférico CAN. É a linha mais baixa de microcontroladores da ST Electronics mas mesmo sendo a mais baixa vai de encontro as necessidades do sistema. Em termos de *performance* pode operar até a uma frequência de 72 Mhz com 64 *Kbytes* de *Flash* e 20 *Kbytes* de *Static Random Access Memory* (SRAM). Tem também 4 *timers* 2 conversores A/D de 10 canais, 2 periféricos SPI e I2C, 3 USART e um periférico CAN e outro USB. Quanto aos GPIO tem disponível 37 para uso do utilizador. [95] Sendo um microcontrolador com um *package Surface Mounted Components* (SMD) surge a necessidade da aquisição de uma *breakout board* para este microcontrolador. A placa adquirida (figura 6.6) dispõem de um *Light Emitting Diode* (LED) ligado ao pino PC13 e um RTC (32.768KHz) tudo o resto que a placa incorpora (resistencias, condensadores, etc) é o necessário para o perfeito funcionamento do microcontrolador.

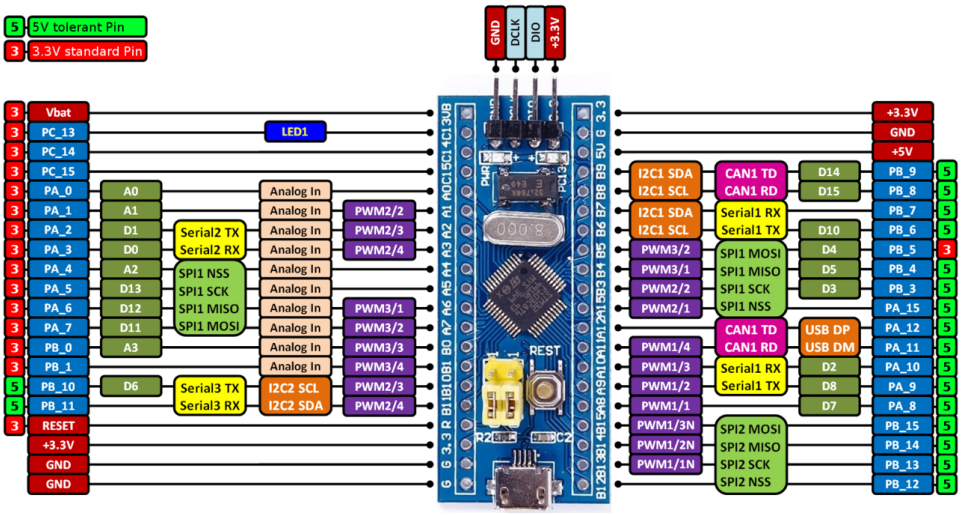


Figura 6.6: STM32f103 *Breakout Board* (adaptado de [96])

6.3.3 Sensores

A escolha de sensores depende diretamente da grandeza que se pretende medir. O facto de um sensor consuma pouca energia não faz por si só uma boa escolha. A característica mais importante de um sensor depende da sua aplicação. Se for necessário obter dados com uma elevada precisão na grandeza medida, é certo que a precisão é um fator decisivo. Se apenas se quer um sensor que seja robusto pois é para ser colocado no exterior onde está exposto a poeiras, elevadas temperaturas, agua, entre outros, talvez a precisão do sensor não será a característica determinante para a escolha.

No caso deste projeto, como já foi mencionado um dos grandes objetivos é o seu baixo custo. Então a escolha será sensores económicos no que toca ao custo nunca esquecendo a sua precisão. A facilidade de aquisição também é um fator a considerar.

Os sensores utilizados neste projeto são os seguintes:

GPS

O módulo escolhido é o uBlox NEO-6M (figura 6.7). Este módulo foi escolhido devido ao seu baixo custo e uma *performance* satisfatória para o projeto. O módulo adquirido está disponível numa *breakout board*, o que permite uma fácil utilização, já dispõe uma antena em cerâmica para captação do sinal. A comunicação deste módulo é feita através do protocolo UART a um *baudrate* de 9600 bps [97]. Depois de alimentado, o GPS irá despoletar uma série de mensagens pela sua porta série. Esta mensagem terá a seguinte sintaxe.

-$GPRMC,hhmmss,A,llll.ll,a,yyyyy.yy,b,vvv.v,ddd.d,ddmmaa,mm.m,d*ss

hhmmss -	Horas GMT +0
A -	Estado (A = OK, V = aviso)
llll.ll -	Latitude
a -	Latitude
yyyyy.yy -	Longitude
b -	E ou W
vvv.v -	Velocidade em nós
ddd.d -	*Course Made Good* (CMG)
ddmmaa -	Data
mm.m -	Variação magnética em graus
d -	E ou W
ss -	*checksum*

Inicialmente enquanto o GPS não obtiver sinal dos satélites a mensagem irá ser enviada pela UART com os parâmetros em branco, à medida que o GPS vai obtendo informações estas mesmas vão ser incluídas na trama de dados.

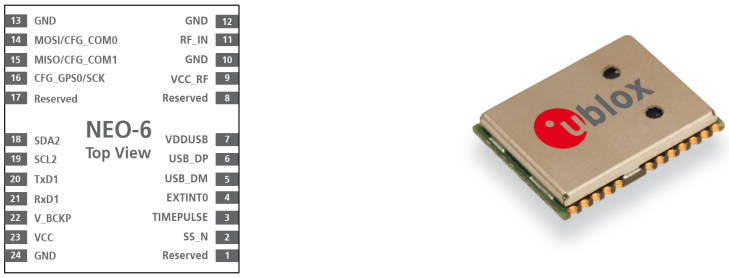


Figura 6.7: GPS uBlox NEO-6M [98]

Sensor de Temperatura

Para quantificar temperatura será utilizado o DS18B20, a escolha deste sensor deve-se ao facto de operar com o protocolo OneWire o que permite a ligação de vários sensores em série, figura 6.8. Com isto é possível adquirir dados referentes à temperatura de vários pontos no meio físico através de apenas um pino. Cada sensor ligado neste barramento tem um endereço e é a partir desse endereço que é possível fazer a aquisição de um sensor em especifico.

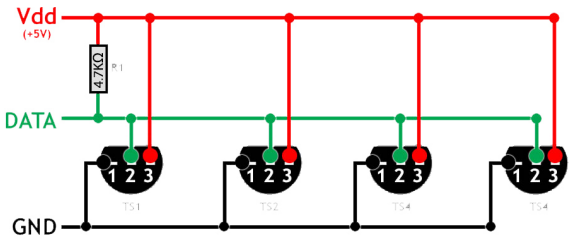


Figura 6.8: Barramento OneWire para o sensor DS18B20 (adaptado de [99])

A resolução do sensor é configurável de 9 a 12 bits, vindo configurado por defeito com 12 bit de resolução, isto corresponde a incrementos de 0.0625ºC. A gama de valores que o sensor é capaz de medir é de -55ºC a 125ºC.

Sensor de Luminosidade

Para quantificar a luminosidade presente irá ser utilizado um LDR. Este é um sensor resistivo que para uma variação na luminosidade que incide sobre

o sensor, resulta numa variação na sua resistência. Com sensores deste tipo pode-se ter duas implementações diferentes. A primeira é a ligação direta, através de um divisor resistivo, a um pino analógico/digital do microcontrolador. A segunda será ligar o LDR a um comparador, por exemplo, o LM324 e seguidamente ligar a saída do comparador a um pino digital do microcontrolador. A grande diferença entre estes dois casos é que no primeiro temos a leitura em dados brutos, já na segunda apenas temos a informação que a luminosidade está a cima ou abaixo de um valor predefinido. Outra grande diferença é a calibração, que no primeiro caso é feito em *software*, no segundo é feito em *hardware*. Não esquecendo que caso se opte pelo segundo caso, implica a adição de um comparador ao sistema.

Acelerómetro

Para registo de todos os movimentos do veículo durante a viagem, é necessário um acelerómetro. Existe uma variedade muito vasta deste tipo de sensores no mercado. Existem sensores que apenas captam movimento num sentido, estes são frequentemente utilizados em suspensões [100] onde na realidade apenas há interesse em analisar um eixo. Outros que possuem 3 eixos para uso genérico, tudo depende da aplicação. Outro aspeto a considerar na escolha neste tipo de sensor é a força g que o sensor vai ser submetido. No caso dos *airbags* nos automóveis existem soluções para este tipo de sensores que tem a capacidade de mensurar até 120 g. Mas por sua vez o *smartphone* não terá necessidade de contabilizar movimentos com 120 g então neste caso é utilizado um sensor com menor g e consequentemente mais pequenos[101].

Para este projeto foi escolhido o MPU-6050 que é um chip que contem um acelerómetro e um giroscópio juntos. É apresentado como um *Low Power* MEMS *Sensor* de 6 eixos, 3 para o acelerómetro e os restantes para o giroscópio. A escala depende da sensibilidade, mas pode ser de +/- 2, 4, 8 ou 16 g. O giroscópio de pode ter uma escala de 250, 500, 1000 ou 2000

°/sec [102].

Uma característica deste sensor é a capacidade de detetar acelerações especificas que caracterizem um determinado movimento devido ao *Digital Motion Processor* (DMP) presente no sensor. O sensor dispõe de um pino de interrupção, este é ativado quando a aceleração exceda determinado valor, valor este parametrizavél. É uma característica interessante para o projeto visto que deste modo o Nó *Slave* deixa de fazer leituras constantemente. Se as leituras do sensor descem a autonomia aumenta.

Este sensor tem interface I2C que permite aquisição de dados até 400kHz em *fast mode*. Para uma maior facilidade de uso foi adquirida uma *breakout board* (figura 6.9), esta dispõe de pinos com espaçamento de 2.54mm o que permite o uso em *breadboard* enquanto decorre a fase de protótipo.

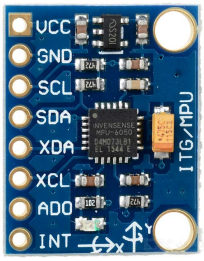


Figura 6.9: *Breakout Board* MPU-6050 [103])

Sensor de pressão atmosférica

Em alguns ambientes existe a necessidade de monitorizar a pressão atmosférica. Como é o caso de transporte de alimentos em câmaras frigoríficas [104]. Onde tem que ser tudo monitorizado para que a qualidade dos produtos seja garantida. De forma a controlar a pressão atmosférica vai ser utilizado o BMP280 da BOSCH que além de ser fornecer a pressão atmosférica, fornece também a temperatura [105]. Este sensor dispõe de um modo *Sleep* que promete gastar 0.1uA. O que vai de encontro a um dos objetivos deste

projeto, o baixo consumo de energia. Segundo o *datasheet* [105], este sensor pode fornecer dados sobre a pressão atmosférica de -500m até 9000m com um erro de precisão absoluta de 8m, já no que toca à temperatura o intervalo de valores é de -40°Caté 85°C. Os dados estão disponíveis na interface I2C ou SPI.

Como o sensor é no *package Land Grid Array* (LGA) foi necessário adquirir este sensor numa *breakout board*. Esta placa (figura 6.10) dispõe os pinos para comunicar com o sensor via I2C.

Figura 6.10: *Breakout Board* BMP280 [106])

Contactos mecânicos

Para simulação de alguns sensores mecânicos, como por exemplo fins de curso de alguma porta ou outro sensor que a sair seja digital, são utilizados botões do tipo on/off.

6.3.4 Transceiver CAN

A maior parte dos *transceivers* requerem 5V como tensão de alimentação como era requerido na norma *standard* ISO 11898. Contudo começou a surgir uma preocupação com a eficiência dos circuitos criados e já existem *transceivers* a operar com 3.3V. Esta alteração reduz em 50%, ou mais, da potencia consumida como se pode observar na figura 6.11 [88]. Da Microchip Technology existe o MCP2551, este é no *package* DIP o que facilita a mon-

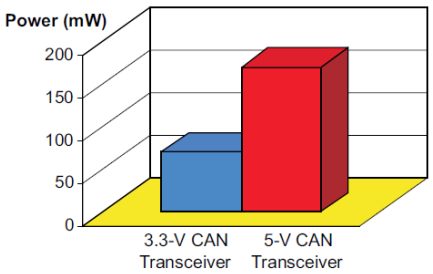


Figura 6.11: Potencia consumida pelos CAN *transceivers* [88]

tagem de *breadboards* por exemplo, tem mecanismo de desligar automático quando a temperatura aumenta, é anunciado como imune a curto circuitos, é possível ligar até 112 nós com este *transceiver*.

Por parte da Texas Instruments existe o VD230 que é um *transceiver* SMD que opera exclusivamente com microcontroladores que usam 3.3V como tensão de funcionamento, consegue funcionar numa rede de 120 nós. Se for necessário um *transceiver* mais robusto, a Texas Instruments oferece o ISO1050 que é semelhante ao VD230 mas este é *galvanically isolated* o que torna ainda mais imune ao ruído, na folha do fabricante é referido como um *transceiver* de alta duração. Na figura 6.12 é possível ver o diagrama de blocos do *transceiver* CAN ISO1050.

Estes 3 *transceivers* são os mais conhecidos e utilizados, existem outros *transceivers* para aplicações mais especificas, um exemplo disso é o ADM3053 da Analog Devices, é um *transceiver isolated* com um conversor DC-DC integrado. Utiliza a a tecnologia *iCoupler* da própria Analog Devices para o isolamento da parte *transceiver* CAN e a *isoPower* para o isolamento do DC-DC [107].

A escolha do *transceiver* cai para o ISO1050 porque devido a sua característica *galvanically isolated* permite isolar lado do microcontrolador protegendo assim o sistema de possíveis ruídos. Outro fator que levou a escolha deste *transceiver* foi o facto de já ter trabalhado com este *transceiver* em

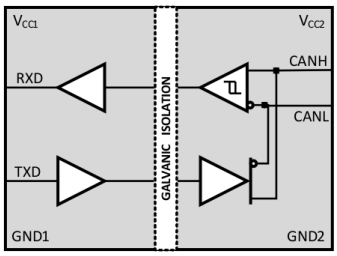


Figura 6.12: Diagrama de blocos do *transceiver* CAN ISO1050[108]

projetos anteriores.

Esta página foi intencionalmente deixada em branco.

Capítulo 7

Implementação

Neste capítulo será apresentado todo o processo de implementação. Primeiramente será abordado o *hardware*, seguidamente o *software* onde será explicado como é feita a aquisição dos dados e algumas configurações de sensores.

7.1 Hardware

Nesta secção é apresentado o *hardware* utilizado no sistema. Primeiramente será abordada a necessidade da criação de uma PCB para facilitar a sua utilização no protótipo. Seguidamente será apresentado o *hardware* utilizado nos nós, tanto nos *Slaves* tanto no *Master*, o *hardware* da Unidade Central também será apresentado.

7.1.1 Módulo BLE (Nó *Master* e Nó *Slave*)

Surgiu a necessidade da criação de uma PCB de modo a tornar as ligações mais fiáveis e mais imunes ao ruído. Outra razão pela qual foi feito uma PCB é devido ao espaçamento da *breakout board* ser 1.27mm e numa fase inicial, para facilidade de uso, é utilizada uma breadboard cujo o espaçamento é de 2.54mm. A PCB desenvolvida permite a utilização do SoC nrf51822 numa

breadboard

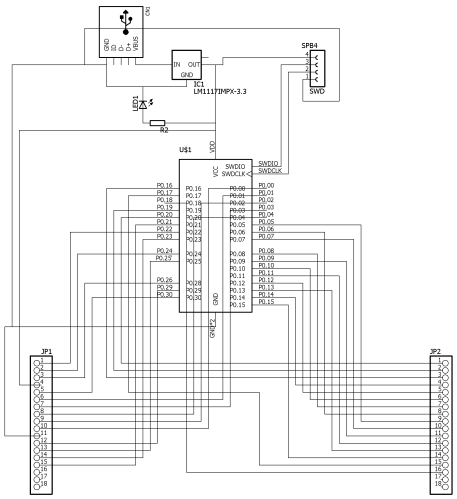
Este módulo BLE apenas será utilizado nos Nós *Slaves* e no Nó *Master* pois são os únicos que utilizam o nrf51822 no sistema. De modo a alimentar a placa de forma independente foi adicionado um conector mini *USB* apenas e somente para alimentação da placa. Foi disposta uma ficha de programação para que seja mais intuitivo programar o módulo. Na figura 7.1a está apresentado o esquemático onde é possível ver ainda, para além do que já foi descrito, um LED para identificar a alimentação da placa.

Após isso foi desenvolvido o *layout* da PCB. Um dos aspetos que se teve em conta da conceção desta PCB é a distancia de pinos que tem que ser aproximadamente 36.25 mm de modo a encaixar em 2 *breadboard* dispostas paralelamente. Sabe-se que o conector USB terá que ser colocado numa extremidade bem como a ficha de programação. Na figura 7.1b é apresentado o *layout* da PCB. No anexo A é apresentado como foi produzido esta PCB (figura 7.1c).

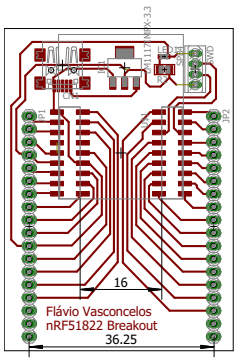
7.1.2 *Hardware Slave*

O *hardware* dos Nós *Slaves* varia à medida do sensor, mas a unidade de processamento mantém-se a mesma, o módulo BLE apresentado em 7.1.1 com o SoC nrf51822 é utilizado em todos os Nós *Slaves*. Todos os nós apresentados para este sistema dispõem de 4 LED's cuja função é indicar o estado de conexão e simular um atuador. Para além dos LED's todos os nós dispõe de 4 *tact switch* para interação com o utilizador.

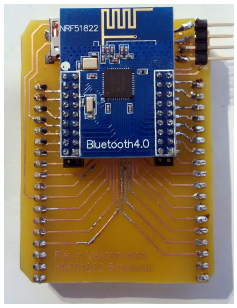
O Nó TEMP, cuja grandeza a medir é a temperatura, dispõe de um sensor de temperatura, o DS18B20, da Maxim Integrated. O sensor sendo digital e usando o protocolo de comunicação OneWire poderá ser ligado a um pino genérico. Os pinos para os LED's e os *tact switch* são os P0.21 ao P0.24 e P0.17 ao P0.20 respetivamente. Na figura 7.2 é apresentado o esquemático do Nó TEMP.



(a)



(b)



(c)

Figura 7.1: PCB Versão 1.0 (a) Esquema elétrico b)*Layout* c)Produto final)

O próximo nó é o Nó LUM que é responsável por mensurar a luminosidade durante a viagem. Para isso tem um LDR ligado através de um divisor de tensão ao microcontrolador. O divisor de tensão terá que ligar ao periférico *Analog to Digital Converter* (ADC) do microcontrolador. O periférico ADC terá de ser ativo uma vez que o sensor dá uma resposta em tensão. O periférico ADC, dispõe do canal 2 no pino P0.01 e esse foi o escolhido. Os LED's e os *tact switch* estão ligados da mesma forma que no Nó TEMP e todos os outros.

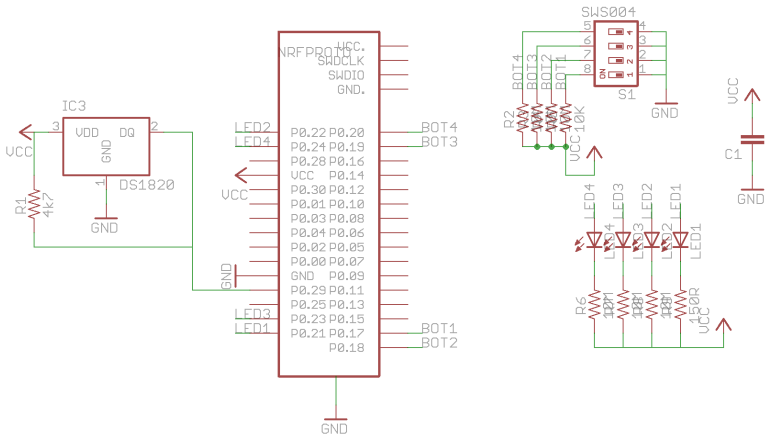


Figura 7.2: Esquema elétrico do Nó TEMP

O esquema elétrico do Nó LUM está apresentado na figura 7.3.

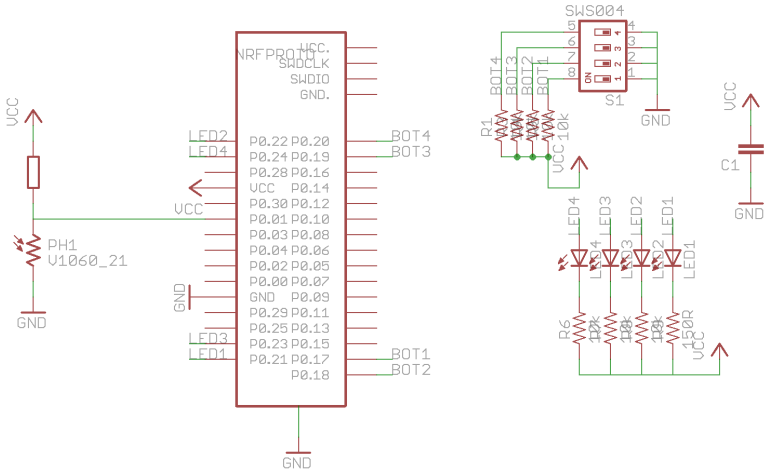


Figura 7.3: Esquema elétrico do Nó LUM

Outro dos 5 nós presentes neste sistema é o Nó ACC. Nó este que está responsável por determinar a aceleração que a carga sofre durante a viagem. O sensor utilizado utiliza o protocolo de comunicação I2C, então os pinos SCL e SCA estão ligados aos pinos P0.03 e P0.04 do SoC respetivamente. Este nó continua a ter os LED's e os *tact switch* como os nós anteriormente apresentados. Na figura 7.4 é apresentado o esquema elétrico do Nó ACC.

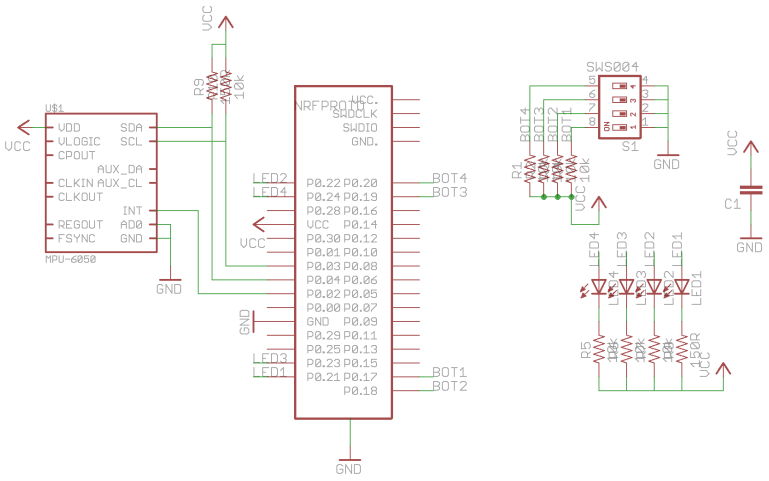


Figura 7.4: Esquema elétrico do Nó ACC

Outro nó que utiliza também como protocolo de comunicação I2C é o Nó PRESS. A grandeza que este contabiliza é a pressão atmosférica, esta grandeza é útil em cargas com mais sensíveis. Os pinos onde estão ligados são os mesmos do Nó ACC, ou seja SCL para o pino P0.03 e o SDA para o pino P0.04. Seguindo o mesmo formato, este nó tem 4 LED's e 4 *tact switch*, como os anteriores. A figura 7.5 apresenta o esquema elétrico deste nó.

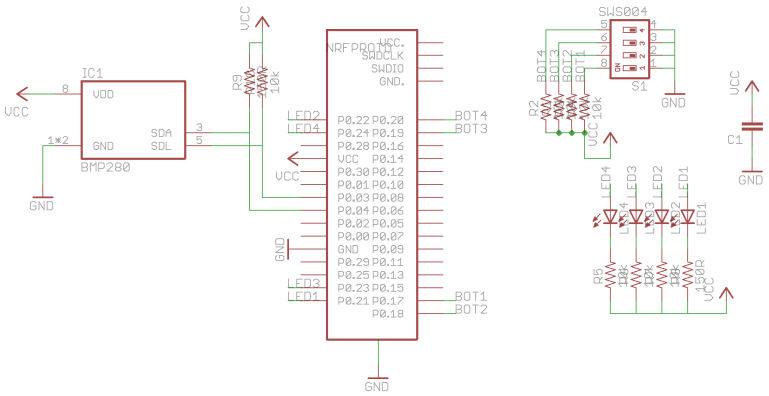


Figura 7.5: Esquema elétrico do Nó PRESS

Por ultimo, e o mais simples, tem-se o Nó IO. O objetivo deste nó não é medir nenhuma grandeza física mas sim simular por exemplo uma porta.

Idealmente é simulado com um contacto físico muito similar ao sensor de uma porta que tem a saída digital. Para isso é colocado um botão normalmente aberto no pino P0.16. Na figura 7.6 é apresentado o esquema elétrico deste nó.

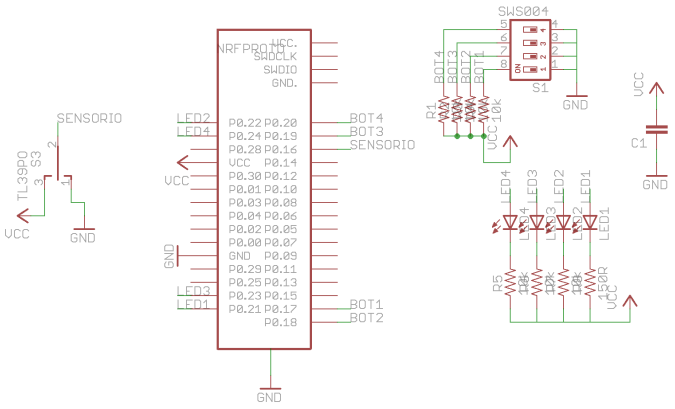


Figura 7.6: Esquema elétrico do Nó IO

7.1.3 *Hardware Master*

Este elemento do sistema tem como encargo reunir toda a informação vinda dos Nós *Slaves*. Não tem nenhum sensor agregado mas à semelhança dos Nós *Slaves* este nó possui, de igual forma, o SoC nrf51822. Apresenta ainda 4 LED's e 4 *tact switch*. Para além disso dispõe de um conector para a ligação entre este nó e a Unidade Central. O esquema elétrico deste nó está apresentado na figura 7.7.

7.1.4 *Hardware* Unidade Central

A Unidade Central é o elemento que centraliza toda a informação do sistema. De forma a acompanhar a descrição do *hardware* é apresentado o esquema elétrico na figura 7.8. Este elemento dispõe de um *transceiver* CAN, o ISO1050, que está ligado diretamente aos pinos do periférico CAN1 do microcontrolador. O microcontrolador utilizado é o ARM STM32f103.

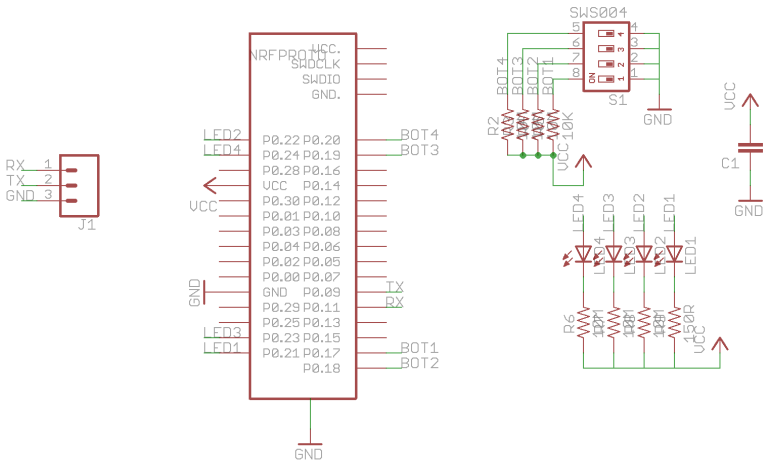


Figura 7.7: Esquema elétrico do Nó *Master*

A Unidade Central também dispõe de um GPS que utiliza UART como protocolo de comunicação, este sensor está ligado ao periférico UART1 do microcontrolador STM32f103. A comunicação entre o Nó *Master* e a Unidade Central é realizada, por UART, para isso a Unidade Central tem um conector que permite a ligação serie. Esse conector cria uma ligação física até ao periférico UART2 do microcontrolador. Para comunicação com o exterior este elemento do sistema dispõe toda a informação por serie através do periférico UART3 do microcontrolador. Esta informação será reencaminhada para um servidor. Este microcontrolador dispõe uma grande flexibilidade no que toca ao periférico UART. O microcontrolador possui de 3 periféricos podendo os pinos ser mapeados. Sabendo que a Unidade Central tem 3 periféricos que utilizam UART como protocolo de comunicação pode-se dizer que este microcontrolador não é *oversized* para o sistema.

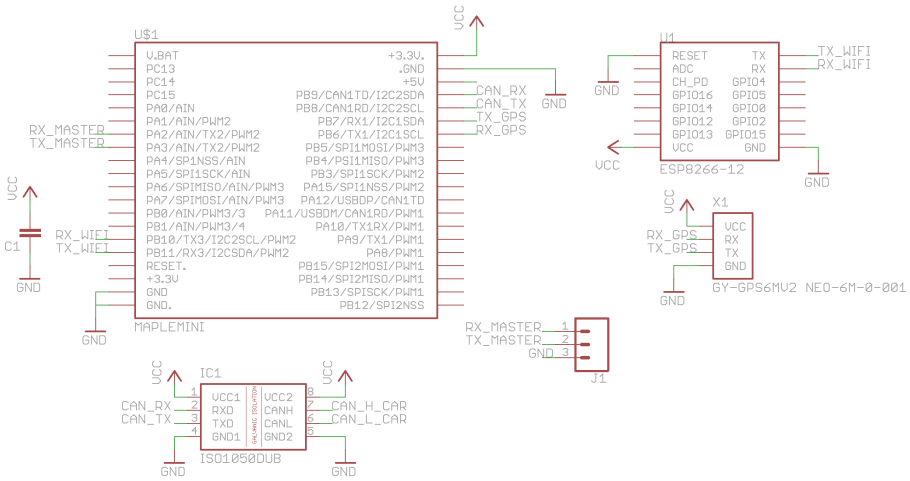


Figura 7.8: Esquema elétrico da Unidade Central

7.2 Software

Nesta secção será apresentado o desenvolvimento do projeto a nível de *software*. Serão apresentados alguns excertos de código C.

7.2.1 *Software Slave*

O *software* desenvolvido para os Nós *Slaves* é relativamente idêntico em todos os nós, pois todos eles funcionam de forma similar. A função destes nós é fazer a aquisição de uma determinada grandeza física e enviar essa informação para o Nó *Master* via BLE.

Inicializações

Como todo e qualquer nó tem que realizar uma serie de inicializações para para o seu correto funcionamento. Na figura 7.9 é apresentado um fluxograma onde identifica claramente cada passo no momento da inicialização do nó.

Primeiramente é inicializado um *timer* e sua respetiva função de interrupção (excerto 7.1) para auxiliar na tarefa de tornar periódica a aquisição

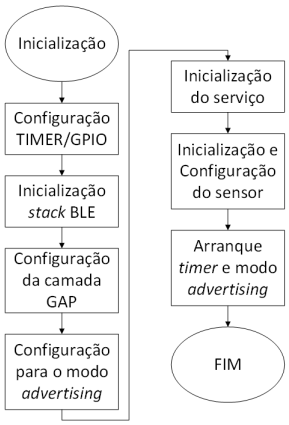


Figura 7.9: Fluxograma da inicialização do Nó *Slave*

do sinal. Para isso é necessário ativar o *timer* (linha 2) e só depois é que é feito a criação do *timer* (linha 3 a 5).

Excerto de Codigo 7.1: Inicialização do *timer*

```
1 // Inicializar o Modulo Timer
2 APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_OP_QUEUE_SIZE, false);
3 app_timer_create(&m_io_timer_id, // Criar Timer
4                APP_TIMER_MODE_REPEATED,
5                state_meas_timeout_handler);
```

Após a inicialização do *timer* é inicializado a entradas e saídas (excerto 7.2), pois o protótipo contém LED's e botões que permitem identificar o estado do módulo. Na linha 2 é configurado os pinos desde o BUTTON_START até ao BUTTON_STOP como entrada ativando as resistências de *pullup*. Na linha 3 é configurado os pinos de saída, do LED_START ao LED_STOP.

Excerto de Codigo 7.2: Configuração de entradas e saídas

```
1 // Configuracao dos botoes
2 nrf_gpio_range_cfg_input(BUTTON_START,BUTTON_STOP,NRF_GPIO_PIN_PULLUP);
3 // configuracao dos leds
4 nrf_gpio_range_cfg_output(LED_START,LED_STOP);
```

Após a configuração das entradas e saídas é inicializado a *stack* do BLE. Para isso é executada a função presente no excerto 7.3. Primeiramente é

configurado o *clock* do SoC (linha 4), a configuração presente na macro NRF_CLOCK_LFCLKSRC indica a origem do cristal, no caso é um cristal de baixa frequência (LFCLK). Seguidamente é inicializado o *handler* do *SoftDevice* (linha 6) com a configuração do *clock*. Depois é executada a função da linha 8 onde é indicado quais são o numero máximo de *slaves* e *master* que o SoC irá estar conectado. Seguidamente na linha 13 é verificado se a RAM é suficiente para o número de conexões que foram indicadas. Após essa verificação é ativado a stack do BLE (linha 15). Nas duas ultimas funções deste excerto de código é indicada a função de *handler* para eventos do BLE e eventos do sistema (linhas 18 e 21).

Excerto de Codigo 7.3: Inicialização da *stack* BLE

```
1  static void ble_stack_init(void)
2  {
3      uint32_t err_code;
4      nrf_clock_lf_cfg_t clock_lf_cfg = NRF_CLOCK_LFCLKSRC;
5      // Initialize the SoftDevice handler module.
6      SOFTDEVICE_HANDLER_INIT(&clock_lf_cfg, NULL);
7      ble_enable_params_t ble_enable_params;
8      err_code = softdevice_enable_get_default_config(CENTRAL_LINK_COUNT,
9                                                      PERIPHERAL_LINK_COUNT,
10                                                     &ble_enable_params);
11     APP_ERROR_CHECK(err_code);
12     // Check the ram settings against the used number of links
13     CHECK_RAM_START_ADDR(CENTRAL_LINK_COUNT, PERIPHERAL_LINK_COUNT);
14     // Enable BLE stack.
15     err_code = softdevice_enable(&ble_enable_params);
16     APP_ERROR_CHECK(err_code);
17     // Register with the SoftDevice handler module for BLE events.
18     err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
19     APP_ERROR_CHECK(err_code);
20     // Register with the SoftDevice handler module for SYS events.
21     err_code = softdevice_sys_evt_handler_set(sys_evt_dispatch);
22     APP_ERROR_CHECK(err_code);
23 }
```

Posteriormente à inicialização da *stack* do *BLE* são inicializado os parâmetros da camada GAP através da função presente no excerto 7.4. Na linha 8 é chamada uma função que permite definir o nome do dispositivo. É através do nome, que o Nó *Master* sabe qual o dispositivo a se conectar. O Nó *Master* só se irá conectar a Nós *Slaves* com determinado nome, nome este defi-

nido na macro DEVICE_NAME. Relativamente aos intervalos do conexão (linha 13 e 14) estes estão definidos nas macros MIN_CONN_INTERVAL e MAX_CONN_INTERVAL que corresponde ao intervalo mínimo e máximo entre cada conexão respetivamente. Na linha 15 é definida a latência (SLAVE_LATENCY) do Nó *Slave*, quando é diferente de zero o periférico opta por não responder quando o Nó *Master* requer dados até o tempo de latência do Nó *Slave*. No entanto, se o Nó *Slave* tiver dados para enviar, ele envia dados a qualquer momento. Na linha 16 é definido (CONN_SUP_TIMEOUT) o tempo limite desde a última troca de dados até que um nó seja considerado perdido, em todos os nós está definido como 4 segundos.

Excerto de Codigo 7.4: Inicialização da camada GAP

```
1  static void gap_params_init(void)
2  {
3      uint32_t              err_code;
4      ble_gap_conn_params_t gap_conn_params;
5      ble_gap_conn_sec_mode_t sec_mode;
6
7      BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
8      err_code = sd_ble_gap_device_name_set(&sec_mode,
9                                            (const uint8_t *)DEVICE_NAME,
10                                           strlen(DEVICE_NAME));
11     APP_ERROR_CHECK(err_code);
12     memset(&gap_conn_params, 0, sizeof(gap_conn_params));
13     gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
14     gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
15     gap_conn_params.slave_latency     = SLAVE_LATENCY;
16     gap_conn_params.conn_sup_timeout = CONN_SUP_TIMEOUT;
17     err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
18     APP_ERROR_CHECK(err_code);
19 }
```

Após a inicialização da camada GAP, o SoC é configurado o modo para *advertising* (excerto 7.5) de modo a que seja "visível"para o Nó *Master*. Primeiramente é definido de que modo é apresentado o nome, no caso, é de forma completa (linha 9). Seguidamente é configurado para que seja visível (linha 10). Após isso, é configurado o tempo que o Nó *Slave* está visível (linha 11), neste caso está sempre visível até que o Nó *Master* se conecte. Mas também há a possibilidade de estar apenas visível apenas 30 segundos

(BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE), mas para este projeto optou-se por estar sempre visível. Seguidamente é ativado o modo *fast advertising* (linha 14), a diferença é entre o *fast* ou *slow* é que o modo *slow* permite intervalos de *advertising* maiores, significa uma comunicação mais lenta mas por sua vez uma economia de energia maior. Foi escolhido o modo *fast*, apesar de o modo *slow* gastar menos energia porque a comunicação é mais lenta, é importante o Nó *Master* encontrar rapidamente o Nó *Slave*. Quando isso acontecer o Nó *Slave* deixa de estar no modo *advertising*, por isso este estado é momentâneo e é preferível uma conexão rápida do que lenta. Estes intervalos de tempo apenas são aplicados na fase inicial. Posteriormente é indicado os intervalos de tempo para o modo *fast advertising* (linhas 15 e 16). Por fim é inicializado o modo *advertising* com os parâmetros definidos anteriormente (linha 18).

Excerto de Codigo 7.5: Inicialização do modo *advertising*

```
1  static void advertising_init(void)
2  {
3      uint32_t            err_code;
4      ble_advdata_t       advdata;
5      ble_adv_modes_config_t options;
6
7      // Build advertising data struct to pass into @ref ble_advertising_init.
8      memset(&advdata, 0, sizeof(advdata));
9      advdata.name_type            = BLE_ADVDATA_FULL_NAME;
10     advdata.include_appearance   = true;
11     advdata.flags                = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
12
13     memset(&options, 0, sizeof(options));
14     options.ble_adv_fast_enabled = true;
15     options.ble_adv_fast_interval = APP_ADV_INTERVAL;
16     options.ble_adv_fast_timeout = APP_ADV_TIMEOUT_IN_SECONDS;
17
18     err_code = ble_advertising_init(&advdata, NULL, &options, on_adv_evt, NULL);
19     APP_ERROR_CHECK(err_code);
20 }
```

Depois da inicialização do modo *advertising* é inicializado o serviço do sensor (excerto 7.6) onde é dado permissões para a escrita e leitura da estrutura do serviço (linha 8 a 11). É possível também criar um *handler* para os eventos da *stack* do BLE associados a este serviço (linha 12) , por exemplo

sempre que o Nó *Master* escreve na estrutura deste serviço.

A ultima coisa que é feita na função é o arranque do serviço com as configurações anteriormente apresentadas (linha 16).

Excerto de Codigo 7.6: Inicialização do serviço (slave)

```
1  static void services_init(void)
2  {
3      uint32_t      err_code;
4      ble_io_init_t io_init;
5      // Initialize IO Service.
6      memset(&io_init, 0, sizeof(io_init));
7      // Here the sec level for the IO Service can be changed/increased.
8      BLE_GAP_CONN_SEC_MODE_SET_OPEN(&io_init.state_char_attr_md.cccd_write_perm);
9      BLE_GAP_CONN_SEC_MODE_SET_OPEN(&io_init.state_char_attr_md.read_perm);
10     BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&io_init.state_char_attr_md.write_perm);
11     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&io_init.state_report_read_perm);
12     io_init.evt_handler       = NULL;
13     io_init.support_notification = true;
14     io_init.p_report_ref      = NULL;
15
16     err_code = ble_io_init(&m_io, &io_init); // START SERVICE
17     APP_ERROR_CHECK(err_code);
18 }
```

A tarefa a seguir à inicialização do serviço é configurar tudo o que seja necessário para o funcionamento do sensor, isto passa por inicialização do periférico, configuração do sensor e calibração. No caso do Nó ACC, por exemplo, cujo protocolo de comunicação do sensor é o I2C deve-se inicializar primeiramente o periférico. Na linha 5 do excerto 7.7 é definido quais são os pinos de SCL e SDA referentes ao protocolo I2C, é indicado também a frequência de funcionamento do protocolo I2C, a prioridade da interrupção e se o *buffer* inicia limpo. Após isto é inicializado o periférico com as configurações anteriores (linha 13). Por fim dá inicio ao periférico através da função presente na linha 15.

Excerto de Codigo 7.7: Inicialização do periférico I2C

```
1  void twi_init (void)
2  {
3      ret_code_t err_code;
4
5      const nrf_drv_twi_config_t twi_mcp6050_config = {
6         .scl              = SCL_PIN,
7         .sda              = SDA_PIN,
8         .frequency        = NRF_TWI_FREQ_400K,
9         .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
10        .clear_bus_init   = false
11     };
12
13     err_code = nrf_drv_twi_init(&m_twi, &twi_mcp6050_config, twi_handler, NULL);
14     APP_ERROR_CHECK(err_code);
15     nrf_drv_twi_enable(&m_twi);
16 }
```

Depois de inicializar o periférico I2C dá se inicio à inicialização do sensor (excerto 7.8), no caso do Nó ACC é realizado as seguintes operações. Primeiramente é reiniciar os valores do acelerómetro giroscópio e sensor de temperatura (linha 8). Seguidamente é verificado o ID do chip através da função presente na linha 12. Caso estas funções sejam executadas com sucesso, a variável transfer_succeeded irá conter o valor booleano 1.

Excerto de Codigo 7.8: Inicialização do sensor

```
1  bool mpu6050_init(uint8_t device_address)
2  {
3      bool transfer_succeeded = true;
4      m_device_address = (uint8_t)(device_address << 1);
5
6      // Do a reset on signal paths
7      uint8_t reset_value = 0x04U | 0x02U | 0x01U; // Resets gyro,accelerometer and temp sensor signal paths.
8      transfer_succeeded &= mpu6050_register_write(ADDRESS_SIGNAL_PATH_RESET, reset_value);
9
10
11     // Read and verify product ID
12     transfer_succeeded &= mpu6050_verify_product_id();
13
14     return transfer_succeeded;
15 }
```

Após estas inicializações todas é dado inicio ou arranque, do *timer* e do modo *advertising* (excerto 7.9) pois até agora foi apenas configurado e só neste momento é dado o inicio. Na linha 5 é dado arranque ao *timer* e

indicado o período do mesmo. Esse período pode variar de Nó *Slave* em Nó *Slave*. O modo *advertising* também é dado arranque, no modo *fast* como previamente configurado (linha 12)

Excerto de Codigo 7.9: Arranque do timer e do modo *advertising*

```
1  static void application_timers_start(void)
2  {
3      uint32_t err_code;
4      // Start application timers.
5      err_code = app_timer_start(m_io_timer_id, STATE_MEAS_INTERVAL, NULL);
6      APP_ERROR_CHECK(err_code);
7
8  }
9  void advertising_start(void)
10 {
11     ret_code_t err_code;
12     err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
13     APP_ERROR_CHECK(err_code);
14 }
```

Programa

Após as inicializações realizadas nos Nós *Slaves*, o programa fica no ciclo principal. Na figura 7.10 é apresentado o fluxograma do ciclo principal e das interrupções geradas pela *stack* BLE.

No caso dos Nós *Slave* o ciclo principal (excerto 7.10) apenas chama uma função de gestão de energia (linha 4). Esta função é uma função não bloqueante que permite que o microcontrolador fique a aguardar algum evento, como por exemplo, interrupções geradas pela *stack* do BLE.

Excerto de Codigo 7.10: Ciclo principal

```
1  // Enter main loop.
2  for (;;)
3  {
4     power_manage();
5  }
```

A *stack* do *BLE* desencadeia várias funções de interrupção, das quais se destaca a função por interrupção *dispatch* (excerto 7.11) que ocorre assim que haja um evento BLE. Quando o evento BLE ocorre é executado uma

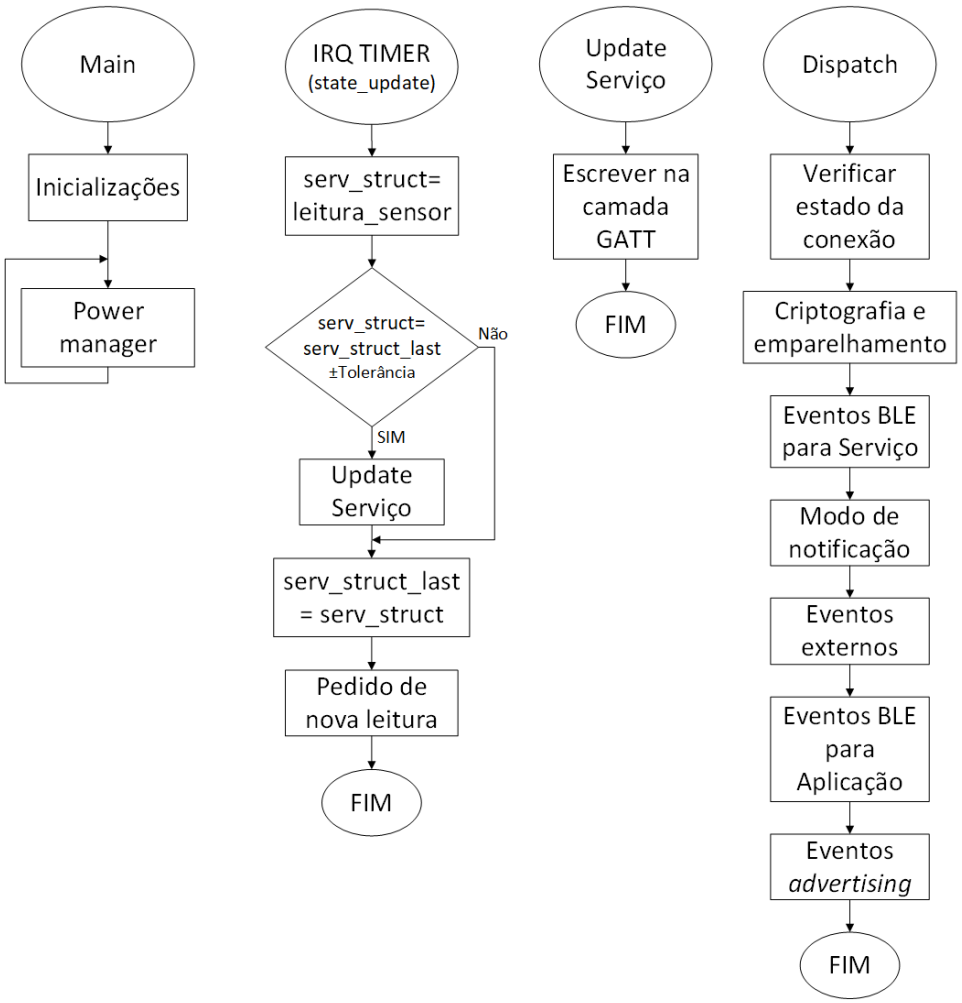


Figura 7.10: Fluxograma do sistema para filtrar dados repetidos

serie de funções necessárias para o correto funcionamento da transmissão de dados entre *Slave* e *Master*. Primeiramente é verificado o estado da conexão (linha 3), após isso é executada a função cujo objetivo é controlar processos de criptografia e emparelhamento (linha 4). Seguidamente é executada uma função manipula os eventos da *stack* do BLE para o interesse do serviço. A próxima função a ser chamada nesta rotina de interrupção (linha 6) é responsável por assegurar o modo de notificação como opção de transmissão. Seguidamente é executado a função (linha 7) que permite desencadear de-

terminados eventos BLE através de botões, como por exemplo, ao carregar num botão forçar uma nova transmissão de dados ou ao carregar no botão desemparelhar do Nó *Master*. É também necessário uma função para lidar com os eventos do BLE associados à aplicação, por exemplo é possível saber quando é que Nó *Slave* foi conectado, desconectado ou o tempo de espera expirou também é possível saber quando há um pedido de escrita ou leitura do serviço, tudo isto é feito chamando a função presente na linha 8. Por ultimo (linha 9), é executado o *handler* que gere eventos relativos ao *advertising*.

Excerto de Codigo 7.11: Função *dispatch*

```
1  static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
2  {
3      ble_conn_state_on_ble_evt(p_ble_evt);
4      pm_on_ble_evt(p_ble_evt);
5      ble_io_on_ble_evt(&m_io, p_ble_evt);
6      ble_conn_params_on_ble_evt(p_ble_evt);
7      bsp_btn_ble_on_ble_evt(p_ble_evt);
8      on_ble_evt(p_ble_evt);
9      ble_advertising_on_ble_evt(p_ble_evt);
10 }
```

Relativamente à aquisição de dados, esta é feita de forma periódica, o período da função que atualiza a estrutura do serviço é variável dependendo do tipo de sensor que o nó contém. Ao ocorrer a função de interrupção (state_update) é copiado o ultimo valor lido para a estrutura do serviço e feito o *update* do serviço através da função presente no excerto 7.12. Primeiramente é guardado uma nova amostra (linha 3) e seguidamente é verificado se o valor a atualizar é igual ao ultimo valor lido tendo sempre em conta uma determinada tolerância definida na macro (TOLUP). Caso se verifique que é diferente, é feito o *update* ao serviço (linha 6). Este vai escrever na camada GATT do *SoftDevice*. Posteriormente o *SoftDevice* fica encarregue de fazer o envio da informação para o Nó *Master*. Seguidamente é guardado a amostra atual de modo a que tenha uma amostra anterior numa próxima execução desta rotina de interrupção (linha 7). Após isto é iniciado o pedido

e uma nova amostra do valor do sensor, neste caso em analise o código é do Nó LUM, esse pedido vai desencadear a função de interrupção do periférico ADC (linha 10) , já configurado previamente (anexo B.1). Nessa rotina de interrupção do periférico ADC é atualizado uma variável global. Numa próxima vez que a função de interrupção (state_update) esse valor volta a ser copiado para a estrutura de dados. Criando assim um ciclo não bloqueante.

Excerto de Codigo 7.12: Função de interrupção do *timer*

```
1  static void state_update(void)
2  {
3  lum_measurement.sample32=sample;
4  if (sample > last_sample+TOLUP || sample < last_sample-TOLUP)
5    {
6       ble_lum_state_update(&m_lum, lum_measurement);
7       last_sample=sample
8     }
9     //new sample
10    nrf_adc_start();
11 }
```

Imagine-se o caso de uma viagem cuja duração é cerca de duas horas, caso não houvesse o mecanismo de verificação de mensagens iguais, iam ser transmitidas mais de 14 mil mensagens. A maior parte destas mensagens iam ser repetidas ou sem valor para o sistema, daí surgiu a necessidade para a criação deste mecanismo.

Cada nó pode ter mais que um serviço com mais ou menos características. Por exemplo o nó cujo sensor é o acelerómetro contém um serviço com 3 características, 3 para os eixos xyz respetivamente. Enquanto o nó com um sensor de saída digital apenas tem um serviço com uma característica.

A periodicidade da função de interrupção depende da variação da grandeza a ser medida. Luminosidade e a temperatura são os dois casos mais díspares. A luminosidade tem uma variação rápida pelo que a execução da função será feita com uma frequência alta mas que não comprometa o sistema. Imaginando o cenário em que Nó LUM esta no compartimento

de carga e sendo um veiculo com o compartimento de carga opaco e sem aberturas para o exterior ,como é o caso dos contentores, a variação da luminosidade irá ser instantânea sempre que a porta seja aberta, mas, irá se manter durante algum tempo no mesmo estado. Sabendo isto, é possível baixar a frequência de aquisição de modo a poupar energia. Analisando a temperatura, a frequência pode relativamente baixa devido à sua variação lenta, no projeto foi utilizado uma frequência de 1 Hz. Mesmo a aceleração que é uma grandeza de variação rápida está implementado com um sistema de interrupção evitando assim as constantes leituras e aumentando a sua autonomia.

7.2.2 *Software Master*

Este elemento da rede que é responsável por receber toda informação dos *Slaves* é denominado de *Master*. E o seu objetivo é encaminhar essa informação para a Unidade Central.

Inicializações

Muito à semelhança dos Nós *Slaves* da rede, o Nó *Master* também terá que executar uma serie de inicializações. O fluxograma está presente na figura 7.11.

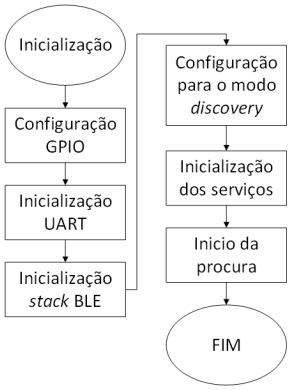


Figura 7.11: Fluxograma da inicialização do Nó *Master*

Primeiramente é configurado um conjunto de saídas para LED's e entradas para botões, pois o protótipo do Nó *Master* contem uma serie de botões e LED's para indicar o seu estado. Seguidamente é configurado o periférico UART como meio de comunicação entre o Nó *Master* e a Unidade Central. Esta função (excerto 7.13) utilizada configura o necessário para a utilização do periférico UART. Primeiramente é feito a configuração de parâmetros básicos, como pinos, controlo de fluxo e *baudrate* (linha 3 a 12). Seguidamente é configurado os aspetos referentes à *buffer* FIFO que vai ser utilizado juntamente com o periférico UART, indicando os tamanhos dos *buffer's* da FIFO (RX e TX). A função *handler* definida em 16 é executada sempre que um evento associado à UART ocorra.

Excerto de Codigo 7.13: Inicialização do periférico UART

```
1  static void uart_init(void){
2     uint32_t err_code;
3     const app_uart_comm_params_t comm_params =
4       {
5         .rx_pin_no   = RX_PIN_NUMBER,
6         .tx_pin_no   = TX_PIN_NUMBER,
7         .rts_pin_no  = RTS_PIN_NUMBER,
8         .cts_pin_no  = CTS_PIN_NUMBER,
9         .flow_control = APP_UART_FLOW_CONTROL_DISABLED,
10        .use_parity  = false,
11        .baud_rate   = UART_BAUDRATE_BAUDRATE_Baud9600
12      };
13    APP_UART_FIFO_INIT(&comm_params,
14                       UART_RX_BUF_SIZE,
15                       UART_TX_BUF_SIZE,
16                       uart_event_handle,
17                       APP_IRQ_PRIORITY_LOW,
18                       err_code);
19    APP_ERROR_CHECK(err_code);
20 }
```

Posteriormente à configuração do periférico UART é dado a inicialização da *stack* BLE, para isso é executada a mesma função que é utilizada nos Nós *Slaves* com uma pequena alteração (excerto 7.14). A certa altura é chamada a seguinte função

Excerto de Codigo 7.14: Excerto da função de inicialização da *stack* do BLE

```
1    err_code = softdevice_enable_get_default_config(CENTRAL_LINK_COUNT,
2                                                    PERIPHERAL_LINK_COUNT,
3                                                    &ble_enable_params);
```

Nessa função é utilizada uma macro, a PERIPHERAL_LINK_COUNT, que permite definir o numero máximo de *Slaves* que este nó se irá conectar no caso do Nó *Master* a macro toma o valor de 5, correspondente aos Nós *Slaves*.

Imediatamente a seguir à inicialização da *stack* BLE é configurado o módulo Database Discovery que lida com a descobrimento dos serviços (excerto 7.15). Quando um *Slave* se conecta a um *Master* é necessário que ambos tenham o mesmo serviço com as mesmas características para que possam interagir um com o outro. O *handler* apresentado na linha 3 é executado sempre que é algum *Slave* é descoberto. O objetivo desse *handler* é fazer a descoberta do serviço.

Excerto de Codigo 7.15: Inicialização do modo *discovery*

```
1  static void db_discovery_init(void)
2  {
3      ret_code_t err_code = ble_db_discovery_init(db_disc_handler);
4      APP_ERROR_CHECK(err_code);
5  }
```

Para que exista uma interação entre o *Master* e o *Slave* é necessário que ambos tenham o mesmo serviço, isto é, todos os serviços que os *Slaves* utilizam terão que ser inicializado no *Master*. Este nó é inicializado com 5 serviços correspondendo a cada Nó *Slave* utilizado neste projeto (TEMP, LUM, ACC, PRESS e IO). No excerto 7.16 é apresentado a função para inicializar o serviço IO. Na linha 5 é indicado o *handler* do serviço. Sempre que o *Master* receba alguma informação referente a este serviço esse *handler* irá ser executado. Seguidamente é feito um ciclo para que sejam iniciados tantos serviços quanto aqueles que possa vir a existir (numero máximo de conexões).

Excerto de Codigo 7.16: Inicialização do serviço IO (*Master*)

```
1  static void io_c_init(void)
2  {
3      uint32_t       err_code;
4      ble_io_c_init_t io_c_init_obj;
5
6      io_c_init_obj.evt_handler = io_c_evt_handler;
7
8      for (m_ble_io_c_count = 0; m_ble_io_c_count < TOTAL_LINK_COUNT; m_ble_io_c_count++)
9      {
10         err_code = ble_io_c_init(&m_ble_io_c[m_ble_io_c_count], &io_c_init_obj);
11         APP_ERROR_CHECK(err_code);
12     }
13     m_ble_io_c_count = 0;
14 }
```

Após a inicialização dos serviços para todos os nós é colocado no modo *scan* para iniciar a procura de *Slaves* na rede. Isso é feito através da função presente no excerto 7.17. Primeiramente é parado algum eventual processo de procura. Na linha 6 é dado inicio à procura de Nós *Slaves*. Todas as funções começadas por **sd**, como é o caso da **sd**_ble_gap_scan_start fazem parte do *SoftDevice* e não está acessível. A função que está na linha 9 serve para mudar os estados dos LED's presentes na placa.

Excerto de Codigo 7.17: Função para iniciar a procura de dispositivos

```
1  static void scan_start(void)
2  {
3      ret_code_t ret;
4      (void) sd_ble_gap_scan_stop();
5     //Start scan with pred window
6      ret = sd_ble_gap_scan_start(&m_scan_params);
7      APP_ERROR_CHECK(ret);
8     //LED Indications
9      ret = bsp_indication_set(BSP_INDICATE_SCANNING);
10     APP_ERROR_CHECK(ret);
11 }
```

Programa

Após feito todas as inicializações é dado o arranque da aplicação propriamente dita. Na figura 7.12 é apresentado um fluxograma onde apresenta o ciclo principal e o *handler* para mensagens recebidas via BLE.

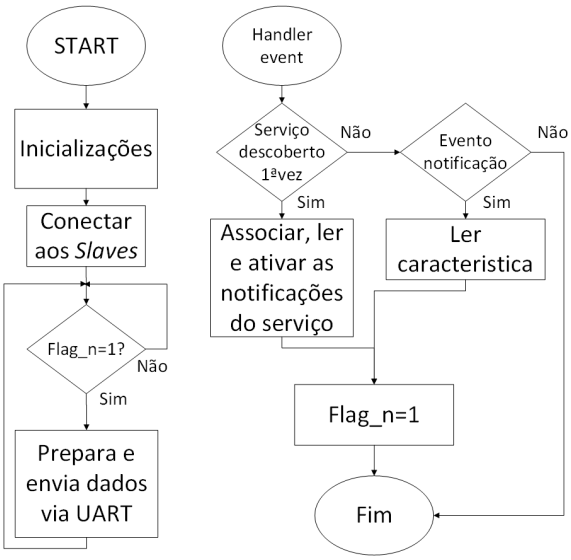


Figura 7.12: Fluxograma do Nó *Master*

É feito o *scan* de *Slaves* enquanto o nível máximo de conexões definidos na macro PERIPHERAL_LINK_COUNT for atingido. No ciclo principal é testado uma serie de *flags* que indicam se determinadas mensagens estão prontas para serem enviadas. No excerto 7.18 é apresentado o código de teste de apenas uma *flag* as restantes serão iguais. Caso alguma dessas *flags* estejam no seu estado alto (linha 1), a informação desse serviço é enviado via UART (linha 5) para a Unidade Central, seguindo uma trama predefinida.

Excerto de Codigo 7.18: Excerto do ciclo principal (*Master*)

```
1 if(flag_usart_ldr==1)
2 {
3   for (uint32_t i = 0; i < strlen(buffer_data_ble_ldr); i++)
4    {
5       while (app_uart_put(buffer_data_ble_ldr[i]) != NRF_SUCCESS);
6   }
7   flag_usart_ldr=0;
8 }
```

A trama tem a seguinte estrutura:

N	:	D	!

N - Indica o numero do nó que deu a origem à informação (1 byte);

: - Separação entre o numero do nó que deu a origem à informação e a informação do sensor (1 byte);

D - Informação do sensor do nó N (3 bytes);

! - Indica o final da trama (1 byte).

Quando se pretende enviar dados via UART, é chamada a função app_uart_put para colocar dados no *buffer* FIFO TX. Quando os dados chegam pela UART é colocado no buffer FIFO RX e, de seguida, é executado o *handler* associado ao periférico UART (uart_event_handle) com o evento APP_UART_DATA_READY. Quando isto ocorrer basta executar a função app_uart_get para ler dados do *buffer* FIFO RX.

Quando algum *Slave* se conectar, e este contenha um serviço inicializado no *Master*, o programa executa o *handler* (excerto 7.19) associado a esse mesmo serviço. Este *handler* é definido na função de inicialização do serviço. Quando houver um evento BLE que obrigue o *handler* a ser executado pela primeira vez o tipo de evento (p_io_c_evt->cvt_type) será a informar que o serviço foi descoberto, por isso nesta função, o programa irá executar o "case"presente na linha 6. Serve como configuração do serviço, primeiramente é o serviço é associado à *stack* do BLE, é lido o serviço e logo de pois é ativado as notificações da característica do serviço. As próximas vezes que ocorrer um evento BLE para que este *handler* seja executado o tipo desse evento será a notificação, isto é, o programa irá executar o "case"presente na linha 19. Irá ser construida a mensagem para ser enviada por UART e ativado uma *flag*. No ciclo *main* é verificado essa *flag* e caso seja verdadeira é realizado o envio.

Excerto de Codigo 7.19: *Handler* para o serviço

```
static void io_c_evt_handler(ble_io_c_t * p_io_c, ble_io_c_evt_t * p_io_c_evt)
{
    uint32_t err_code;
    switch (p_io_c_evt->evt_type)
    {
        case BLE_IO_C_EVT_DISCOVERY_COMPLETE:
            err_code = ble_io_c_handles_assign(p_io_c,
                                               p_io_c_evt->conn_handle,
                                               &p_io_c_evt->params.io_db);
            APP_ERROR_CHECK(err_code);
            //Read SERVICE IO
            err_code = ble_io_c_bl_read(p_io_c);
            APP_ERROR_CHECK(err_code);
         //Activate Notifications
            err_code = ble_io_c_bl_notif_enable(p_io_c);
            APP_ERROR_CHECK(err_code);
            break;

        case BLE_IO_C_EVT_IOCAR_NOTIFICATION:
        {
                 //READ the caracteristic
              sprintf(buffer_data_ble_io, "5:%d!", p_io_c_evt->params.state);// build the message
              flag_usart_io=1; //main cicle flag
            break;
        }
        default:
            break;
    }
}
```

7.2.3 Unidade Central

Este elemento da rede é responsável por centralizar toda a informação. Juntamente com a informação dos sensores presente na rede BLE será adicionada ainda a informação do sensor de GPS e alguma informação relevante como velocidade instantânea e as rotações por minuto. Toda essa informação é centralizada ficará pronta para ser enviada para um servidor.

Inicializações

O microcontrolador utilizado na Unidade Central, o stm32f103, tem 3 periféricos UART e todos eles irão ser necessários. Para o Nó *Master*, para o sensor GPS e para o módulo que comunica com o servidor. Contém também

o periférico CAN que irá ser utilizado para adquirir informação direta do carro através da ficha OBD-II. Primeiramente o programa começa por inicializar o periférico UART. Na figura 7.13a é apresentado o fluxograma que apresenta a inicialização da UART.

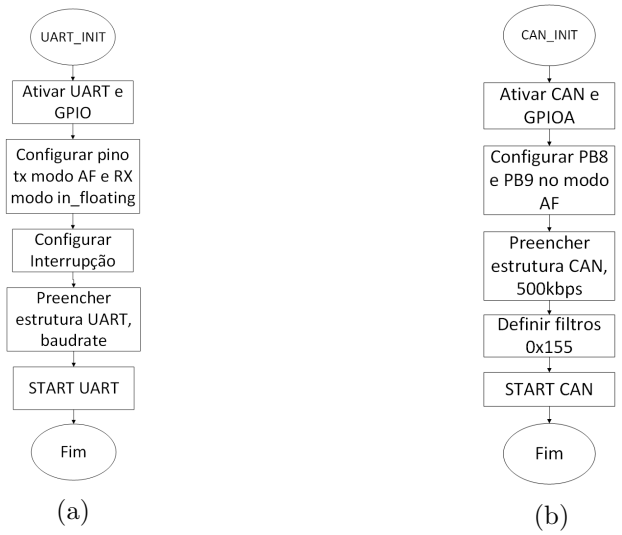


Figura 7.13: a)Fluxograma da inicialização da UART b)Fluxograma da inicialização do CAN

A primeira tarefa a realizar é ativar o *clock* do periférico e ativar os pinos onde esse periférico é ligado. Seguidamente é configurado os pinos TX e RX do microcontrolador. De forma a tornar-se fiável, é utilizado interrupções para aquisição de dados por UART, esse é o processo realizado a seguir. Após a configuração da interrupção, onde é indicado a prioridade, é configurado o periférico UART onde é, por exemplo, indicado o *baudrate*, controlo de fluxo, paridade entre outros. Por fim é feito o arranque do periférico e da interrupção. No anexo B.3 é apresentado o código de inicialização da USART2.

Para além do periférico UART também é necessário inicializar também o periférico CAN. É possível visualizar na figura 7.13b o fluxograma correspondente à inicialização do CAN. A primeira tarefa é inicializar os perifericos, CAN e o porto onde os pinos estão ligados. Seguidamente são

configurados os pinos como *alternate function* pois deixarão de ser pinos de caráter normal mas sim para uso exclusivo do CAN. Posteriormente é configurado o periférico CAN preenchendo uma estrutura onde é indicado o modo de funcionamento do CAN, *Prescaler*, prioridades dos *buffers First In, First Out* (FIFO), modo *automatic wakeup*, entre outros. Seguidamente é configurado os filtros CAN, o processo de configuração é feito por *software* mas a filtragem propriamente dia é feita por *hardware* poupando assim tempo de processamento [95]. Os filtros podem ser configurados no modo *mask* ou *identifier list*. No modo *mask* é utilizado uma máscara para de modo a fazer filtrar as mensagens aceitando mensagens com uma gama de identificadores. Já no modo *identifier list* é definido um identificador especifico de modo a só aceitar determinadas mensagens [95]. Como nesta aplicação são apenas duas informações (velocidade e as Rotações Por Minuto (RPM) do motor) relevantes e que estão contidas na mesma mensagem, então foi optado pelo método de ID. Por fim é dado o arranque ao periférico já configurado. A função que executa a inicialização do periférico CAN está na integra no anexo B.4.

Após feitas as inicializações, o microcontrolador estará pronto para executar o ciclo principal.

Programa

Após a inicialização de todos os periféricos o programa executado o ciclo principal. Na figura 7.14 é apresentado o fluxograma da Unidade Central.

No seu ciclo principal (excerto 7.20), é verificado se existe alguma informação pronta a ser enviada para o servidor.

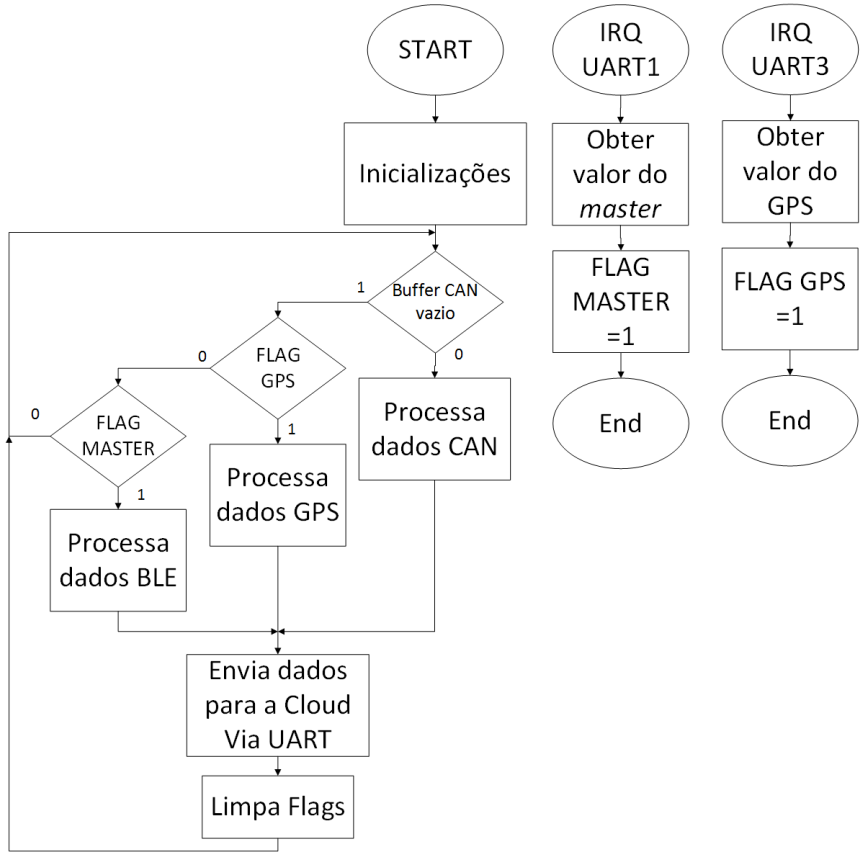


Figura 7.14: Fluxograma da Unidade Central

Excerto de Codigo 7.20: Ciclo principal (Unidade Central)

```
if (CAN1->RF0R & CAN_RF0R_FMP0){
   CAN_Receive(CAN1, CAN_FIFO0, &rx_message);
   cnt=0;
   if (rx_message.StdId==0x155){
      sprintf(buffervel, "VL:%d!", (rx_message.Data[4] << 4 ) | ( rx_message.Data[3] & 0xff));
      sprintf(bufferrpm, "RP:%d!", (rx_message.Data[1] << 4 ) | ( rx_message.Data[0] & 0xff));
      }
      do{
         while(USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET);
         USART_SendData(USART3, buffervel[cnt]);
         cnt++
      }while( buffervel[cnt-1]!=TERMINATOR);
      cnt=0;
      do{
         while(USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET);
         USART_SendData(USART3, bufferrpm[cnt]);
         cnt++
      }while( buffervel[cnt-1]!=TERMINATOR);
}
```

No caso do protocolo CAN apenas é recebida a informação da velocidade o veículo e as rotações do motor. Para isso é verificado o estado do *buffer* presente no periférico CAN do microcontrolador (linha 1). Caso o *buffer* contenha informação, essa informação é lida. Após a leitura é feito o tratamento da mensagem (linhas 5 e 6). Após isso as mensagens são enviadas via UART para o módulo que fará a conectividade com o servidor, no caso a instância da UART é a numero 3 (linha 8 a 18).

Existem 2 rotinas de interrupção para os 2 periféricos UART (Nó *Master* e GPS), quando alguma dessas rotinas de interrupção ocorrer irá ser ativado uma *flag* e guardada a mensagem recebida numa variável para ser processada no ciclo principal.

Conhecendo o tipo de trama que cada elemento pode enviar para a Unidade Central torna-se fácil separar a informação de forma clara. O objetivo será conceber uma nova trama que posteriormente seja enviada para o módulo que fará a conectividade com este sistema e o servidor.

A trama de dados que serão enviados para a *cloud* terá o seguinte aspeto.

T	:	D	!

T - Indica a grandeza da informação, na tabela 7.1 é apresentado as várias possibilidades (2 bytes);

: - Separação entre a grandeza da informação e a informação propriamente dita (1 byte);

D - Informação do sensor;

! - Indica o final da trama (1 byte).

Tabela 7.1: Tipos de mensagens a ser enviadas para a *Cloud*

T	Grandeza	Gama de Valores	Bytes
01	Temperatura	-55,+125	1
02	Luminosidade	0,+255	1
3X	Aceleração eixo X	-32768, +32767	2
3Y	Aceleração eixo Y	-32768, +32767	2
3Z	Aceleração eixo Z	-32768, +32767	2
04	Pressão Atmosférica	300, 1100	2
05	Sinal Digital	0,1	1
LA	Latitude	000000N, 905959S	7
LO	Longitude	0000000E, 1805959W	8
TH	Relogio, Horas	0,23	1
TM	Relogio, Minutos	0,59	1
DT	Data(DDMMAA	000000,311299	6
VL	Velocidade Instantânea	0-200	1
RP	RPM	0-10000	2

Capítulo 8

Resultados

Neste capítulo são descritos alguns testes que permitem verificar o correto funcionamento e avaliar o desempenho do sistema desenvolvido. Na figura 8.1 é apresentado a arquitetura do cenário de testes.

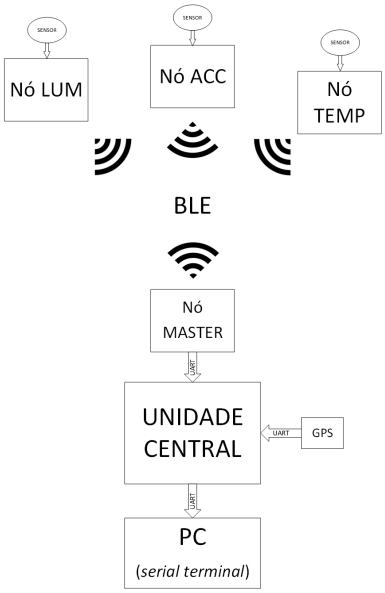


Figura 8.1: Arquitetura do cenário de testes

Foi realizado um ensaio com os Nós TEMP, LUM e ACC. Foi feito um *scan* à porta série da Unidade Central e foi obtido um log de dados presente

no anexo C. Para uma explicação é apresentado apenas um excerto desse log. Para compreensão destes dados deve-se ter em conta a tabela 6.1 presente na secção 6.2.

LA:41.11314N!
LO:-8.58688W!
DT:251017!
TH:14!
TM:32!
01:24!
02:107!
3X:1132!
3Y:-52!
3Z:16832!

Logo no inicio no excerto acima é possível verificar que os dados do GPS são apresentados, latitude (41.11314N) e a longitude (-8.58688W). Seguidamente a data (251017) e a hora (14) e por fim os minutos (32). Após os dados do GPS surge os informação referente ao Nó TEMP, Nó LUM e Nó ACC respetivamente. Com estes dados podemos concluir que estão 24 graus celsius e que a luminosidade numa escada de 0 a 255 é de 107 e praticamente não existe aceleração. Os dados obtidos do Nó ACC estão na sua forma bruta. O sensor tem 16 bits, sabendo que o bit mais significativo corresponde ao sinal (aceleração positiva e aceleração negativa) tem-se 15 bits para o modulo da aceleração (32768), outro dado importante é que o sensor está configurado para uma sensibilidade de 2g. Com isto pode-se concluir que o valor 16384 corresponde a 1g. Pode-se afirmar que nó ACC está a sofrer uma aceleração no eixo dos X de cerça de 0.07g e no eixo dos Y de 0.003g no eixo dos Z cera de 1 g, isto é, nos eixos X e Y o corpo encontra-se parado no eixo Z está sob a força da gravidade.

É de verificar que nas linhas seguintes apenas apresenta os dados relativos ao GPS, uma vez que as alterações da temperatura da luminosidade e da aceleração são relativamente pequenas e serão consideradas como ruído. A dada altura o Nó LUM recebe uma diferença de luminosidade substancialmente elevada, então de 107 que era o ultimo valor registado até à data passou para 180 que foi o valor registado na nova leitura. O mesmo aconteceu com o Nó ACC que numa leitura relevante registou valores de aceleração referentes ao eixo do X de 1.14 g e no eixo do Y de 0.29 g e no eixo do Z de 0.70 g. Não esquecendo do Nó TEMP que registou 24 graus do inicio ao fim desta captura de dados.

Relativamente ao consumo energético foi realizado uma medição da corrente consumida dos vários elementos da rede. Para a realização da medição foi utilizado um multímetro. Deveras não é a melhor solução, pois o intervalo de tempo que há transmissão o consumo dispara mas o multímetro tem uma reação lenta, mascarando assim esse pico de corrente. A solução ideal seria a utilização de um osciloscópio com o respetivo log de dados, de modo a analisar a corrente consumida. Apesar disso foi feito um ensaio com o multímetro de forma a estimar a corrente consumida.

Para demonstrar os resultados é apresentada a tabela 8.1. É possível verificar que o Nó ACC consome em média 8 mA, mais 1 mA face ao Nó IO, isto deve-se ao facto de o nó ACC conter um sensor (MPU-6050) e a informação a enviar será 3 vezes superior e o Nó IO apenas ter um *tact switch* para simulação de um fim de curso. O Nó *Master* apresenta um consumo energético mais elevado, isto deve-se ao facto do Nó *Master* ter que lidar com um numero mais elevado de conexões. Relativamente à Unidade Central, esta é deveras a que consome mais energia. Isto deve-se ao facto de este elemento conter GPS. O consumo do GPS isolado é cerca de 65mA.

Os Nós *Slaves* contêm uma bateria de 1750 mAh. Sabendo que a bateria é capaz de fornecer 1750 mA por hora e cada Nó *Slave* consome entre 7 e 8

Tabela 8.1: Consumo energético dos elementos da rede

Nome	Consumo (mA)
Nó TEMP	7.1
Nó LUM	7
Nó ACC	8
Nó PRESS	7.35
Nó IO	7
Nó Master	14.74
Unidade Central	103.2

mA, através da equação 8.1 obteve-se que a bateria dura até 10 dias com a exceção dos Nós ACC e PRESS que duram cerca de 9 dias.

$$h = \frac{mAh}{mA}(=)h = \frac{1750}{8} = 218.75\ horas \therefore \approx 9\ dias \tag{8.1}$$

O Nó *Master* e a Unidade Central podem ser alimentados pelo veiculo uma vez que este conjunto está ligado fisicamente um ao outro e que por sua vez está ligado ao conector OBD-II do veículo.

Capítulo 9

Conclusão e Trabalho Futuro

Após o término deste projeto e alguma reflexão sobre o trabalho desenvolvido, chega-se à conclusão que a utilização de uma rede de sensores sem fios torna-se vantajoso pela sua flexibilidade e simplicidade. No que toca à poupança de energia, ao utilizar uma rede de sensores sem fios utilizando o BLE como tecnologia, os resultados são bastante satisfatórios. Consumos médios na casa dos 8mA para os Nós *Slaves* e de 15mA para o Nó *Master*. Isto permite que os Nós *Slaves* estejam operacionais até 10 dias com uma bateria de 1750 mAh. A este conjunto soma-se ainda a parte de IoT o que permite que o sistema esteja acessível remotamente, permitindo ao utilizador monitorizar dados da viagem do outro lado do mundo.

O baixo consumo de energia é fundamental para que uma RSSF se mantenha a funcionar, e esse era um dos principais objetivos do projeto e foram conseguidos de forma satisfatória. Os sensores escolhidos realizaram medições suficientemente precisas para o sistema, não comprometendo o mesmo. As vantagens de utilizar uma rede de sensores sem fios foram evidenciadas neste projeto. A recolha dos dados do veículo também foi realizada com sucesso, mas para isso foi necessário fazer engenharia inversa para descobrir is ID's das mensagens mais relevantes para o sistema o que não foi uma tarefa fácil.

Atualmente este tipo de redes são utilizadas em muitas outras aplicações. A agricultura é exemplo disso, é fácil imaginar um agricultor com inúmeros hectares de área verde que precisa ser monitorizada. Com um sistema composto por vários sensores e/ou atuadores é possível, por exemplo, a prevenção de incêndios, realização de estudos de alguma cultura agriculta ou mesmo para automatizar alguns processos como a rega. Este é um exemplo real onde as redes de sensores sem fios são extremamente úteis, torna-se muito simples, rápido e eficaz a monitorização de uma área verde.

Para trabalho futuro seria interessante realizar uma interface gráfica para uma utilização do sistema via web. Outro melhoramento a ter em conta seria a criação de um invólucro resistente a poeiras e humidade de modo a possibilitar a implementação deste sistema em locais mais adversos.

O resultado deste trabalho levou à realização de um *paper* [109], este foi apresentado na conferência 2017 SOLI - IEEE em Itália. É possível ver o *paper* no anexo D

Bibliografia

[1] Martin Leopold. Sensor Network Motes :Portability & Performance. (December):1–123, 2007.

[2] Kahina Chelli. Security Issues in Wireless Sensor Networks : Attacks and Countermeasures. I, 2015.

[3] David Culler, Deborah Estrin, and Mani Srivastava. Overview of sensor networks. *Computer*, 37(August):41–49, 2004.

[4] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[5] Data Communication. *Wireless Sensor and Actuator Networks Algorithms and Protocols for Scalable Coordination*. Wiley, 2010.

[6] Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, Rasit Eskicioglu, Peter Graham, and Frank Sommers. Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*, pages 521–551, 2006.

[7] Thomas Schoellhammer, Ben Greenstein, and Deborah Estrin. Hyper: A Routing Protocol To Support Mobile Users of Sensor Networks. *Center for Embedded Network Sensing*, jan 2006.

[8] Marcel Staroswiecki. On fault tolerant estimation in sensor networks. *... of European Control Conference, Cambridge, UK*, (5):1–6, 2003.

[9] G. Hoblos, M. Staroswiecki, and A. Aitouche. Optimal design of fault tolerant sensor networks. *IEEE International Conference on Control Applications.*, pages 467–472, 2000.

[10] J M Kahn, R H Katz a C M Fellow, and K S J Pister. Next Century Challenges : Mobile Networking for " Smart Dust. *Advances*, pages 271–278, 1999.

[11] Ian F Akyildiz, Dario Pompili, and Tommaso Melodia. Underwater acoustic sensor networks : research challenges. 3:257–279, 2005.

[12] Alper Bereketli and Ozgur B. Akan. Communication coverage in wireless passive sensor networks. *IEEE Communications Letters*, 13(2):133–135, 2009.

[13] Anis Koubâa, Mário Alves, and Eduardo Tovar. IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview. *Architecture*, (July), 2005.

[14] Jason L. Hill and David E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, 2002.

[15] Seth Edward-austin Hollar, David M Auslander, Kristofer S J Pister, David E Culler, and Albert P Pisano. COTS Dust. 2000.

[16] A.G Fallis. The family of motes preceeding Telos and their capabilities. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013.

[17] Jan Beutel and E T H Zurich. The BTnode Story Reflections on Almost a Decade of Mote Class Devices. *Computing*, pages 1–44.

[18] Atmel. 8-bit Atmel Microcontroller Programmable ATmega128L. *Www.Atmel.Com/Atmel/Acrobat/2467S.Pdf*, (8-bit Atmel Microcontroller):384, 2011.

[19] Crossbow Technology. iMote 2. *Datasheet*, pages 2–4, 2012.

[20] Crossbow Technology Inc. TelosB Mote Platform - Datasheet. pages 1–2, 2004.

[21] Prof Margaret Martonosi. The Princeton ZebraNet Project : Sensor Networks for Wildlife Tracking ZebraNet. pages 1–14.

[22] Pei Zhang, Christopher M. Sadler, Stephen a Lyon, and Margaret Martonosi. Hardware design experiences in ZebraNet. *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, 7:227, 2004.

[23] The Samraksh Company. eMote User ' s Manual for the . NOW. *Manual*, (September), 2015.

[24] STM32F103 - STMicroelectronics - Web Page.

[25] Maxim Buevich, Niranjini Rajagopal, and Anthony Rowe. Hardware assisted clock synchronization for real-time sensor networks. *Proceedings - Real-Time Systems Symposium*, pages 268–277, 2013.

[26] Atmel. ATmega128RFA1 Microcontroller with Low Power Transceiver for ZigBee and 1 Pin Configurations ATmega128RFA1. *Configurations*, pages 1–524, 2006.

[27] Eugene Shih, Seong-Hwan Cho, Nathan Ickes, Rex Min, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks.

Proceedings of the 7th annual international conference on Mobile computing and networking - MobiCom '01, pages 272–287, 2001.

[28] Alec Woo and David E. Culler. A transmission control scheme for media access in sensor networks. *Proc. ACM 9th International Conference on Mobile Computing and Networking (MOBICOM'01)*, pages 211–235, 2001.

[29] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[30] Rex Min, Travis Furrer, and Anantha Chandrakasan. Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks Massachusetts Institute of Technology.

[31] José Chilo, Carl Karlsson, Per Ängskog, and Peter Stenumgaard. EMI disruptive effect on wireless industrial communication systems in a paper plant. *IEEE International Symposium on Electromagnetic Compatibility*, pages 221–224, 2009.

[32] Abel C. Lima-Filho, Ruan D. Gomes, Marc??u O. Adissi, Tssio Alessandro Borges Da Silva, Francisco A. Belo, and Marco A. Spohn. Embedded system integrated into a wireless sensor network for online dynamic torque and efficiency monitoring in induction motors. *IEEE/ASME Transactions on Mechatronics*, 17(3):404–414, 2012.

[33] Timothy J Lynham, Charles W Dull, Ashbindu Singh, U N Environmental Program, Mundt Federal Building, and Sioux Falls. Requirements for Space-based Observations in Fire Management : A report by the Wildland Fire Hazard Team , Committee on Earth Observation Satellites (CEOS) Disaster Management Support Group (DMSG). *Group*, 00(C):762–764, 2002.

[34] Çağdaş Döner, Gökhan Şimşek, Kasım Sinan Yıldırım, and Aylin Kantarcı. Forest Fire Detection with Wireless Sensor Networks.

[35] Yu Liyang, Wang Neng, and Meng Xiaoqiao. Real-time forest fire detection with wireless sensor networks. *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, 2:1214–1217, 2005.

[36] H.R. Bogena, M. Herbst, J.a. Huisman, U. Rosenbaum, a. Weuthen, and H. Vereecken. Potential of Wireless Sensor Networks for Measuring Soil Water Content Variability. *Vadose Zone Journal*, 9(4):1002, 2010.

[37] R Beckwith, D Teibel, and P Bowen. Report from the field: results from an agricultural wireless sensor network. *29th Annual IEEE International Conference on Local Computer Networks*, pages 471–478, 2004.

[38] Alessandro Matese and Salvatore Filippo Di Gennaro. Technology in precision viticulture: a state of the art review. *International Journal of Wine Research*, 7:69, may 2015.

[39] Q. Shan, Y. Liu, G. Prosser, and D. Brown. Wireless intelligent sensor networks for refrigerated vehicle. *Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication (IEEE Cat. No.04EX710)*, 2:525–528, 2004.

[40] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, 2002.

[41] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring

application. *Proceedings of the 2nd international conference on Embedded networked sensor systems SenSys 04*, 2:214–226, 2004.

[42] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Peh Peh Li-Shiuan, and Daniel Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002)*, pages 96–107, 2002.

[43] Pei Zhang, Christopher M. Sadler, Stephen a Lyon, and Margaret Martonosi. Hardware design experiences in ZebraNet. *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, 7:227, 2004.

[44] Wi-fi Bluetooth and Smart Sub. Wireless connectivity for IoT applications. *Application Paper*, 2015.

[45] Ciara O'Brien. Wearables: Samsung chases fitness fans with Gear Fit 2 - Web Page, 2016.

[46] TIM HAWKINS. An edge-to-cloud continuum: Inside GE Digital's Predix Industrial Internet of Things platform, 2016.

[47] Li Li, Xiaoguang Hu, Ke Chen, and Ketai He. The applications of WiFi-based Wireless Sensor Network in Internet of Things and Smart Grid. *Proceedings of the 2011 6th IEEE Conference on Industrial Electronics and Applications, ICIEA 2011*, pages 789–793, 2011.

[48] Adam Stone. What is Wibree? - Web Page, 2007.

[49] Dynastream. ANT Protocol — Dynastream Innovations - Web Page.

[50] Phil Smith. Comparing Low-Power Wireless Technologies, 2011.

[51] Dynastream Innovations Inc. ANT Message Protocol and Usage. pages 1–134, 2013.

[52] RS Components. 11 Internet of Things (IoT) Protocols You Need to Know About DesignSpark - Web Page, 2012.

[53] Eleanor Bash. *Protocols and Architectures for Wireless Sensor Networks*, volume 1. 2015.

[54] Artem Dementyev, Steve Hodges, Stuart Taylor, and Joshua Smith. Power Consumption Analysis of Bluetooth Low Energy, ZigBee and ANT Sensor Nodes in a Cyclic Sleep Scenario. *Paper*, pages 2–5.

[55] IEEE Computer Society. *Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*, volume 2005. 2005.

[56] L a N Man and Standards Committee. *IEEE Standard for Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements– Part 15.4: Wireless MAC and PHY Specifications for Low-Rate WPANs*, volume 2006. 2006.

[57] Bluetooth Special Interest Group. Specification of the Bluetooth System Covered Core Package Version 4.0. *Specification paper*, 0(June):2302, 2010.

[58] Digi International. Demystifying 802.15.4 and ZigBee. *Zoonoses and Public Health*, 54(1):66, 2007.

[59] Link-LABS. ZigBee Vs. Bluetooth: A Use Case With Range Calculations - Web Page.

[60] Steven R Strain and G Jerry. Generation and Analysis of ZigBee^{TM} IEEE802.15.4 signals in the 2.4 GHz band. 30(6):1–22, 2001.

[61] R Heydon and N Hunn. Bluetooth Low Energy - Web Page. *Bluetooth SIG*, 2012.

[62] Óscar Pérez Domínguez. ZigBee : Overview. *Presentation*, pages 1–18, 2009.

[63] Raoul Van Bergen and Digi International. Zigbee Mesh Networking 10. *Channels*, (September), 2008.

[64] JeeNode SMD - Hardware - JeeLabs . net.

[65] WiSense. System Overview - WiSense - Web Page, 2014.

[66] Rkris2013. WiSense nodes in a greenhouse — WISENSE - Web Page, 2016.

[67] Rkris2013. Smart community use case – Wirelessly controlled watering of green areas — WISENSE, 2015.

[68] Rkris2013. The LPG cylinder enters the IOT era — WISENSE - Web Page, 2015.

[69] Antonio Lignan. The RE-Mote platform — Zolertia, 2016.

[70] Libelium. Waspmote - Wireless Sensor Networks Open Source Platform, 2016.

[71] Libellium. Waspmote datasheet. *Journal of Chemical Information and Modeling*, 53:160, 2013.

[72] Terrence O'Brien. iPhone 4S claims title of first Bluetooth 4.0 smartphone, ready to stream data from your cat - Web Page, 2011.

[73] Mike Ryan. Outsmarting Bluetooth Smart - Quick Note. *CanSecWest*, 2014.

[74] Adopted Specifications — Bluetooth Technology Website.

[75] A.G Fallis. Final Report on the Inter-Agency Workshop on Research Issues for Smart Environments. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013.

[76] Lynch JP. Chapter 2 Bluetooth Transmission Technology. pages 6–21, 2002.

[77] Carles Cufí Kevin Townsend and Robert Davidson Akiba. *Getting Started with Bluetooth Low Energy* . O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2014.

[78] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors (Switzerland)*, 12(9):11734–11753, 2012.

[79] NXP Semiconductors. UM10204 I2C-bus specification and user manual. (April):64, 2014.

[80] Philips Semiconductors. AN10216-01 I2C manual. page 51, 2003.

[81] Cnet. Maxim buys Dallas Semi for $2.5 bln - CNET - Web Page.

[82] Wire Application Guide. 1-Wire Application Guide midon design. *Application Guide*, 2009.

[83] Design Support and App Notes. Guidelines for Reliable Long Line 1-Wire ® Networks 1-Wire Network Topologies. *Technical Document*, pages 1–18, 2008.

[84] Maxim integrated. AN126:1-Wire Communication Through Software. *Application Note*, pages 1–12, 2002.

[85] C A N in Automation (CiA). CAN physical layer - Web Page.

[86] C A N in Automation (CiA). CAN Data Link Layer - Web Page.

[87] Eugen Maye. Reliable data exchange in the automobile with CAN - Web Page. 2008.

[88] Steve Corrigan. Introduction to the Controller Area Network (CAN). Technical report, Texas Instruments, 2008.

[89] Dr. Conal Watterson. Controller Area Network (CAN) Implementation Guide, 2012.

[90] David H Deans. How M2M and IoT Enable New Data Intensive Apps - Web Page, 2016.

[91] Bus Reset. 1-Wire Overview 1-Wire Support on the Desktop Nano-Board NB2DSK01. (Figure 1):1–2, 2013.

[92] Texas Instruments. CC2640 SimpleLink™ Bluetooth® Smart Wireless MCU. 2015.

[93] Bluetooth Smart Power, System Cost, and Without Compromise. SmartBond ™ DA1458x Product Family SmartBond DA14580 flagship.

[94] Mouser Electronics. nRF51822 Bluetooth Low Energy 2.4GHz SoC - Nordic — Mouser Portugal.

[95] *STM32F101xx, STM32F102xx, ST M32F103xx, STM32F105xx and STM32F107xx advanced ARM -based 32-bit MCUs Datasheet.*

[96] Electronicsgru. Stm32f103 Blink Led using keil and stmcubemx - MIcrocontroller Projects - Web Page.

[97] Addicore. u-blox NEO-6M GY-GPS6MV2 GPS module with on board EEPROM - Web Page, 2016.

[98] Onboard Rtc, Assistnow Autonomous, Hybrid Gps, Assistnow Online, Assistnow Offline, O M A Supl, G P S Solution, The Neo, The Ddc,

All Neo, Model Type, and Supply Interfaces. NEO-6 series. *Product Summary*, pages 6–7.

[99] Tweaking4All. How to measure temperature with your Arduino and a DS18B20, 2014.

[100] Bosch Semiconductors. Active suspension systems - Web Page, 2017.

[101] STMicroelectronics. MEMS acceleration sensor: single-axis for central airbag applications, 2017.

[102] InvenSense Inc. MPU-6000 and MPU-6050 Product Specification. *InvenSense Inc.*, 1(408):1–57, 2013.

[103] Electronoobs. PID control drones mpu6050 mpu9250 gyro accelerometer euler msp432 rtos - Web Page.

[104] Vítor De Freitas Pereira, Eduardo Castello Branco Doria, Bento Da Costa Carvalho Júnior, Lincoln De Camargo Neves Filho, and Vivaldo Silveira Júnior. Avaliação de temperaturas em câmaras frigoríficas de transporte urbano de alimentos resfriados e congelados. *Ciência e Tecnologia de Alimentos*, 30(1):158–165, 2010.

[105] Bosch Sensortec. Digital pressure sensor BMP280 - Datasheet. 2015.

[106] Adafruit. Adafruit BMP280 I2C or SPI Barometric Pressure Altitude - Adafruit Industries - Web Page.

[107] Analog Devices. *ADM3053 - Datasheet*, 2004.

[108] Texas Instruments. Isolated CAN Transceiver - ISO1050. *Datasheet*, (June 2009):33, 2009.

[109] Ana Almeida Flávio Vasconcelos, Lino Figueiredo, João Ferreira. SMART Sensor Network. *2017 SOLI - IEEE*, 2017.

Esta página foi intencionalmente deixada em branco.

Apêndice A

Anexo 1 - Produção de PCB's

Neste projeto surgiu a necessidade da criação de uma PCB para o primeiro protótipo. Este anexo tem como objetivo a explicação dos passos que foram realizados da produção dessas PCB's. Existem várias formas de produzir PCB's, desde as mais arcaicas até às mais profissionais. Como objetivo era produzir um numero relativamente baixo de PCB's era impensável produzir as PCB's numa empresa dedicada ao serviço. Então foi realizado uma pesquisa de métodos alternativos para a produção de PCB's em casa. Após alguma pesquisa e relatos de pessoas que já tentaram vários métodos chegou-se à conclusão que o método de transferência de *toner* é o mais fácil, eficaz e com equipamento relativamente fácil de adquirir.

A.1 Material necessário

Existe uma serie de itens que são necessários para a confeção de PCB's por este método.

Placa PCB - Placa PCB virgem. Este método não usa placas PCB já pré sensibilizadas.

Papel - O papel tem que ser revestido com uma fina camada de cera de um lado e facilmente dissolúvel em água. Não há nenhum papel especifico, mas o melhor será papel fotográfico brilhante. Mas nem todo o papel é igual varia de marca para marca.

Impressora - Quanto à impressora, esta tem que ser a laser e a impressão tem que ser com máxima qualidade.

Ferro de engomar - Será necessário para ajudar a realizar a transferência do *toner* para a placa PCB.

Acetona - Para limpar a PCB numa fase inicial e final.

Palha de aço - Serve para polir a placa PCB antes de todo o processo.

Escova macia - O ideal é uma escova de dentes e irá ser necessário durante a remoção do papel.

Cloreto de hidrogénio (HCl) - Talvez é um dos itens mais difíceis de adquirir mas nas drogarias locais facilmente se encontra. O cloreto utilizado neste processo foi de 35 vol.

Agua Oxigenada (H2O2) - Esta água oxigenada não é a tradicional que se costuma adquirir na farmácia, a que de adquire nas farmácias é de 10vol, a utilizada neste processo foi de 30vol. Este item também se encontra em drogarias locais.

Água (H2O) - Para criar a solução juntamente com HCl + H2O2.

Óculos de luvas de segurança - A solução que irá ser preparada será extremamente forte, pois irá corroer cobre. Deve-se evitar o contacto com a pele ou olhos.

A.2 Impressão

É inútil indicar uma lista de bons tipos de papel porque os fabricantes tendem a alterá-los em momentos aleatórios. Então o que melhor se pode fazer é ir a uma gráfica e imprimir, ir experimentando outras gráficas e ver o papel que resulta melhor. Há relatos que em papel autocolante ou de revista também se obtém bons resultados.

Deve-se imprimir o *layout* no lado brilhante do papel fotográfico. Não tocar neste lado antes ou depois da impressão. A impressão terá que ser espelhada para que quando o *toner* passar para a PCB não fique invertido. A maioria das impressoras a laser permitem ajustar a densidade do *toner*. Ao ajustar a densidade ao máximo, mais *toner* é depositado no papel e isso será melhor para este caso. Deve-se imprimir pelo menos 2 cópias do *layout* na mesma página, pois se danificar o primeiro, teremos um *backup*. Pode-se também imprimir vários projetos na mesma página, visto que o papel fotográfico é relativamente caro.

A.3 Preparação da placa PCB

Se o cobre da PCB parecer oxidado, deve-se primeiro lavá-lo com um pouco de vinagre (ou um ácido mais forte como HCl), o que retornará o metal para um estado não oxidado. Esfregar o lado de cobre com a palha de aço até que o cobre seja brilhante e limpo. Em seguida, esfregar o lado de cobre com uma toalha de papel embebida em acetona. Este passo é essencial! A acetona removerá a sujeira e a gordura remanescentes. Se houver algo no cobre, o *toner* não ficará bem nesses locais.

A.4 Transferência do *toner*

Recortar o *layout*, deixando margens mínimas, exceto de um lado. Para facilitar, pode-se cortar pelo menos dois lados do papel para que eles correspondam com os lados do PCB. Isso torna o alinhamento trivial.

Colocar o ferro de engomar na potência mais alta. Coloque a placa PCB com o cobre virado para cima, sobre uma superfície resistente ao calor. Seguidamente colocar o *layout* em cima do cobre, com a parte impressa virado para baixo, alinhando-o adequadamente. Assegurando que o papel não se move, colocar o ferro quente em cima dele. Pressionar forte e de seguida retirar o ferro a cada 5 segundos, garantindo que a placa inteira seja aquecida uniformemente. Repetindo esse processo algumas vezes, não há nenhum numero certo de vezes porque depende muito da potencia do ferro e do papel em si. É certo que o papel pode ficar um pouco acastanhado durante todo esse processo. Isto requer alguma pratica para encontrar a quantidade certa de pressão que é necessária: muito pouca pressão causará uma transferência fraca, muita pressão fará com que o *toner* desapareça ou fique borratado.

A.5 Remover o papel

Colocar a placa num reservatório com água quente. Deixar embeber por pelo menos 10 minutos. Após isso deve-se poder descascar as primeiras camadas do papel. Se houver problemas nesta etapa pode-se recorrer à escova de dentes. Assim que se chegar a uma camada seca, deve-se mergulhar novamente. Quando se alcançar a última camada, que será um pouco mais difícil para sair, a escova de dentes é a melhor ferramenta para ajudar na remoção. Para partes realmente difíceis, como o espaço entre traços paralelos, pode-se quebrar a ultima camada com uma agulha. Na figura A.1a é possível ver o aspeto que deverá ter a PCB neste ponto.

Quando tudo tiver removido, pode-se deixar secar a placa. Se alguma das camadas ainda permanecer dentro de pequenos orifícios ou entre traços paralelos aparecerão com cor branca. Neste caso deve-se esfregar essas áreas com a escova de dentes, ou usar uma agulha para remover as áreas brancas, mesmo quando estão secas. Não há problema se houver alguma camada na parte superior das áreas de *toner*. Na Figura A.1b é possível ver a PCB já sem o papel, apenas com o *toner*.

Caso uma pista fique interrompida e se não for uma parte muito grande ou delicada, pode-se repará-la com um bom marcador permanente. Se, por outro lado, o *toner* estiver borratado, pode ter sido por exercer demasiada pressão durante o passo de transferência do *toner*, o papel pode ser de fraca qualidade ou talvez seja necessário tentar usar uma configuração de densidade de *toner* mais baixa na impressora.

Se, em algum momento, for necessário recomeçar o processo pode-se remover o *toner* com acetona e começar de novo. O *toner* pode sair devido ao papel ser de fraca qualidade, limpeza insuficiente com acetona ou insuficiente calor no ferro.

(a) Remoção do papel

(b) Aspeto final

Figura A.1: Remoção do papel da PCB

A.6 Revelação

Deve-se usar luvas de segurança, porque ninguém quer ter qualquer substância que dissolva metal em contacto com a pele. Os óculos de segurança também são uma boa pratica para evitar que salpicos acidentais. O recipiente não pode ser de metal.

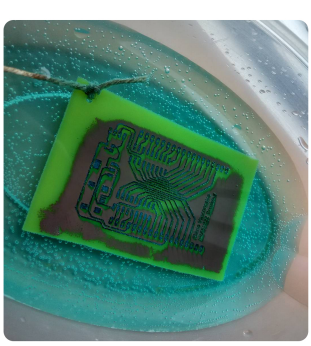
A mistura HCl + H2O2 funciona bem, é transparente e também é muito económica. Por exemplo, as concentrações com 30% H2O2 e 35% HCl e agua serão 8/27/66 porções respetivamente.

Para criar a mistura sem o risco de salpicar, deve-se começar com a água, adicionar o HCl e finalmente o H2O2. A água deve ser destilada, embora a água da torneira também funcione mas não tão bem.

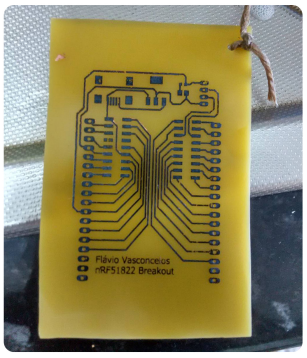
A revelação com a solução de HCl+H2O2+H2O dura poucos minutos. A solução ficará verde devido ao óxido de cobre que se dissolve. Pode-se agitar a solução um pouco para acelerar o processo mas não é necessário utilizar um tanque de bolhas. Na figura A.2a é possível ver que a solução já está um pouco esverdeada e o cobre já começa a desaparecer nas extremidades.

Deve-se ter um reservatório à mão para lavar a PCB quando todo o cobre desaparecer. Na figura A.2b é apresentado o resultado final da PCB já lavada. O H2O2 perderá as suas propriedades rapidamente em contacto com a água e com o oxigénio. Isso significa que a solução perderá suas propriedades após alguns dias ou mesmo horas, e o calor acelerará o processo. Isso também significa que se houver necessidade de armazenar a solução, deve-se usar uma garrafa com uma tampa que não seja perfeitamente hermética. No futuro quando for para voltar a utilizar a solução pode-se reativar a solução com H2O2 novo. Caso não se queira reutilizar a solução, a melhor maneira de descarta-la é levar para uma estação de eliminação química. Deve-se indicar no frasco o que contém, para que possa ser processado corretamente. Uma alternativa menos ecológica é derramar a solução no ralo depois de diluída com uma enorme quantidade de água ou (de preferência)

neutraliza-la. Neutralização pode ser feita adicionando a quantidade correta de NaOH (hidróxido de sódio), que transformará a solução em água + sal + resíduos do óxido de cobre. Deve-se usar indicadores de pH para testar quando atingir um valor neutro (cerca de 7).



(a) Remoção do cobre da PCB



(b) Aspeto final após revelação

Figura A.2: Remoção do cobre da PCB

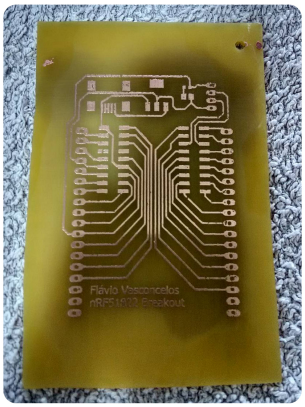
A.7 Acabamento

Depois de lavar e secar a PCB, deve-se remover o *toner* imediatamente. É relativamente mais fácil remover o *toner* antes furar a PCB, mas deixar o *toner* irá proteger o cobre durante as etapas a seguir. Sempre que necessário, use uma toalha embebida em acetona para limpar o *toner*. O próximo passo é furar a PCB, a menos que seja para componentes SMD. Para a maioria dos furos de componentes comuns, uma broca de 0,8 mm é a ideal. Na figura A.3a é possível ver a PCB já limpa sem qualquer resido de *toner*.

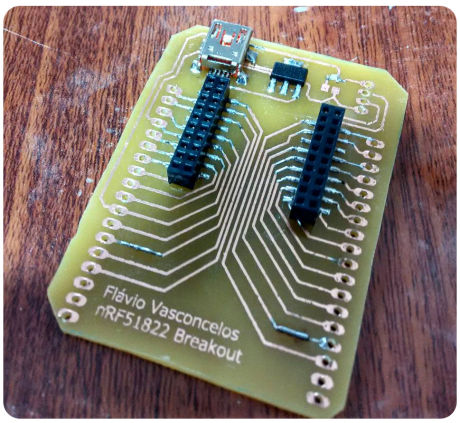
Para melhorar a aparência da PCB e facilitar a soldadura, pode-se imprimir um diagrama de componentes no lado dos componentes, da mesma maneira que se transferiu o *toner* no lado do cobre. É muito mais fácil porque o *toner* fica melhor ainda do que no lado do cobre. Não há necessidade de utilizar papel brilhante, o papel de escritório comum fará a sua tarefa. De lembrar que é necessário espelhar a imagem antes de imprimir. Esfre-

gue a superfície com acetona, limpar qualquer sujidade dos furos e alinhe o padrão usando os furos, segurando a placa contra uma luz brilhante. Logo a seguir utilizar o ferro como de anteriormente, e de seguida mergulhar na água. Remover o papel, descascando-o, o *toner* deve ficar forte o suficiente. Por fim pode-se remover qualquer papel residual com a escova de dentes.

Antes de soldar, deve-se certificar de que as pistas estão limpas e livres de óxido. Uma limpeza rápida com acetona deve ser feita. Se o PCB é apenas um protótipo descartável, pode-se simplesmente soldar e utilizar. Na figura A.3b é possível visualizar a PCB durante o processo de soldadura. No entanto, se for necessário durar mais tempo, é necessário proteger o PCB. A proteção mais básica é aplicar fluxo sobre as pistas de cobre. Isso funciona como um tipo de verniz que limpa o cobre e facilita a soldadura. No entanto, a longo prazo pode desgastar. Então, para aqueles PCB's que realmente precisam durar muito ou resistir a ambientes mais extremos, é indispensável proteger o cobre contra oxidação. Em suma pode-se simplesmente soldar a placa utilizando o fluxo para tornar a soldadura mais fácil, depois deve-se remover antes do envernizamento, para que o verniz adira com mais eficácia. Por fim, aplicar o verniz para proteger o cobre do PCB e dar um acabamento brilhante. Na figura A.4 é possível ver o aspeto final da PCB desenvolvida para este projeto.



(a) PCB sem o *toner*



(b) PCB durante o processo de soldadura

Figura A.3: Acabamento final da PCB

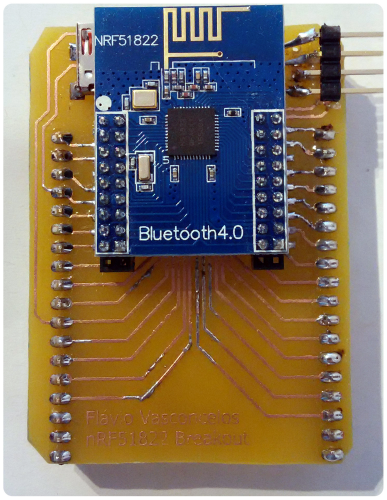


Figura A.4: PCB finalizada

Esta página foi intencionalmente deixada em branco.

Apêndice B

Funções de Inicialização

B.1 ADC INIT

Esta função permite configurar o periférico ADC com 10bit de resolução esta informação e outra está definida na estrutura presente na 2ª linha, é escolhido o canal que no caso é o 2, bem como uma rotina de interrupção.

```
1  void ldrconfig(void){
2      const nrf_adc_config_t nrf_adc_config = NRF_ADC_CONFIG_DEFAULT;
3      // Initialize and configure ADC
4      nrf_adc_configure( (nrf_adc_config_t *)&nrf_adc_config)
5      nrf_adc_input_select(NRF_ADC_CONFIG_INPUT_2);
6      nrf_adc_int_enable(ADC_INTENSET_END_Enabled << ADC_INTENSET_END_Pos);
7      NVIC_SetPriority(ADC_IRQn, 2); //Prioridade 2
8      NVIC_EnableIRQ(ADC_IRQn);
9  }
```

B.2 Serviço IO INTIT

O ciclo *for* nesta função, permite que seja conectado mais que um *slave* com o mesmo serviço. Isto permite que na rede seja possível ter mais que um sensor digital. A macro TOTAL_LINK_COUNT presente na linha 6 contém o numero máximo de conexões da rede.

```
1  static void io_c_init(void)
2  {
3      uint32_t       err_code;
4      ble_io_c_init_t io_c_init_obj;
5      io_c_init_obj.evt_handler = io_c_evt_handler;
6      for (m_ble_io_c_count = 0; m_ble_io_c_count < TOTAL_LINK_COUNT; m_ble_io_c_count++)
7      {
8          err_code = ble_io_c_init(&m_ble_io_c[m_ble_io_c_count], &io_c_init_obj);
9          APP_ERROR_CHECK(err_code);
10     }
11     m_ble_io_c_count = 0;
12 }
```

B.3 USART_init

```
1  void usart_init(uint32_t baud) {
2
3    RCC_APB1PeriphClockCmd( RCC_APB1Periph_USART2, ENABLE);
4    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);
5    //PA2 USART TX Alternate Function Push-Pull
6    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2; //pino 2
7    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
8    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
9    GPIO_Init(GPIOA, &GPIO_InitStructure);//inicia o GPIOA2
10   //PA3 USART RX GPIO_Mode_IN_FLOATING
11   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3; //pino 3
12   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
13   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
14   GPIO_Init(GPIOA, &GPIO_InitStructure);//inicia o GPIOA3
15   /* Configura o Priority Group com 1 bit */
16   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
17   NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
18   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
19   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
20   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
21   NVIC_Init(&NVIC_InitStructure);
22
23   USART_DeInit(USART2);
24   USART_InitStructure.USART_BaudRate = baud;
25   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
26   USART_InitStructure.USART_StopBits = USART_StopBits_1;
27   USART_InitStructure.USART_Parity = USART_Parity_No;
28   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
29   USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
30   USART_Init(USART2, &USART_InitStructure);
31   USART_Cmd(USART2, ENABLE);
32
33   USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
34 }
```

B.4 can_init

```
1  void can_init(void)
2  {
3    /* Enable GPIO clock */
4    RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1, ENABLE);
5    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB , ENABLE);
6    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO , ENABLE);
7    /* Configure CAN pin: RX */
8    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
9    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
10   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
11   GPIO_Init(GPIOB, &GPIO_InitStructure);
12   /* Configure CAN pin: TX */
13   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
14   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
15   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
16   GPIO_Init(GPIOB, &GPIO_InitStructure);
17   // Remap2 is for PB8 and PB9
18   GPIO_PinRemapConfig(GPIO_Remap1_CAN1 , ENABLE);
19   /* CAN 1 cell init */
20   CAN_InitStructure.CAN_Prescaler = 18; //64
21   CAN_InitStructure.CAN_Mode = CAN_Mode_Normal;
22   CAN_InitStructure.CAN_SJW = CAN_SJW_1tq; //2
23   CAN_InitStructure.CAN_BS1 = CAN_BS1_13tq; //11
24   CAN_InitStructure.CAN_BS2 = CAN_BS2_2tq; //4
25   CAN_InitStructure.CAN_TTCM = DISABLE;
26   CAN_InitStructure.CAN_ABOM = DISABLE;
27   CAN_InitStructure.CAN_AWUM = DISABLE;
28   CAN_InitStructure.CAN_NART = ENABLE;
29   CAN_InitStructure.CAN_RFLM = DISABLE;
30   CAN_InitStructure.CAN_TXFP = DISABLE;
31   /* CAN1 filter init, accept every message */
32   CAN_FilterInitStructure.CAN_FilterNumber = 0; // 0..13 for CAN1, 14..27 for CAN2
33   CAN_FilterInitStructure.CAN_FilterMode = CAN_FilterMode_IdList;
34   CAN_FilterInitStructure.CAN_FilterScale = CAN_FilterScale_16bit;
35   CAN_FilterInitStructure.CAN_FilterIdHigh = (0x155 << 5);
36   CAN_FilterInitStructure.CAN_FilterIdLow = (0x155 << 5);
37   CAN_FilterInitStructure.CAN_FilterFIFOAssignment = 0;
38   CAN_FilterInitStructure.CAN_FilterActivation = ENABLE;
39   CAN_FilterInit(&CAN_FilterInitStructure);
40   CAN_Init(CAN1, &CAN_InitStructure );
41 }
```

Apêndice C

Log de Resultados

LA:41.11314N!
LO:-8.58688W!
DT:251017!
TH:14!
TM:32!
01:24!
02:107!
3X:1132!
3Y:-52!
3Z:16832!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58693W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:6!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!

```
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58692W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
3X:18654!
3Y:4745!
3Z:11566!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11314N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11315N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58691W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
02:180!
LA:41.11317N!
LO:-8.58690W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
LA:41.11317N!
LO:-8.58689W!
DT:251017!
TH:15!
TM:7!
```

Apêndice D

Paper

Na página seguinte

SMART Sensor Network

with Bluetooth Low Energy and CAN-BUS

Flávio Vasconcelos, Lino Figueiredo, Ana Almeida
GECAD Research Center
Instituto Superior de Engenharia do Porto
Porto, Portugal
1100419@isep.ipp.pt, lbf@isep.ipp.pt, amn@isep.ipp.pt

Joao C. Ferreira
Information Sciences, Technologies and Architecture
Research Center Instituto Universitário de Lisboa (ISCTE-IUL), Lisbon Portugal
Joao.carlos.ferreira@iscte.pt

***Abstract*— This paper proposes a system to monitor, through the internet, the data of a logistics distribution truck. For this, it was implemented a reliable and flexible wireless sensor network with low energy consumption. The technology used for the radio system was Bluetooth Low Energy (BLE). Each node in the network contains one type of sensor. The sensors information together with GPS and On-Board Diagnostics (OBD) data collected by the central unit, and later transmitted to the cloud by GSM or Wi-Fi.**

Keywords*— *Bluetooth Low Energy; Internet of Things; Wireless Sensor Network; Controller Area Network; Piconet; Cloud.

I. INTRODUCTION

Advances in the areas of micro processing, Micro-Electro-Mechanical Systems (MEMS) and wireless communication stimulate the development of sensory equipment that can be used in a variety of application areas. Representing a subclass of ad-hoc wireless networks, Wireless Sensors Networks (WSN) are considered the new generation of real-time embedded systems with limited computing, power and memory resources. These networks can have large numbers of wireless nodes (also known as motes [1]) and are projected to collect and transmit data.

Usually nodes have low processing power capability to increase their autonomy. The fact that they are portable require the use of batteries. The radio module is the largest wear factor of the battery at the time of data transmission. There are other factors, such as, acquisition time, active time of the microcontroller, that are also important factors. One of the biggest challenges in WSN is to create mechanisms for cooperation between the network nodes so that the network has the capability to solve problems efficiently.

This type of solution is widely used in several scenarios because it is a low-cost solution for many applications. Other factors that lead to the use of WSN are: self-organization, node mobility in the geographic space, autonomous operation, ability to withstand poor environmental conditions, scalability, ease of use, among others [2].

A. Background and Motivation

The implementation of reliable, fault-tolerant, and robust computational systems is difficult due to external factors that cannot be controlled or predicted. With the advance of technology there is a need to create systems that can monitor a large area, but at a relatively low cost. To fill this gap, the development of microelectronics provides the development of components with increasingly smaller dimensions, lower prices, greater functional capacity and lower energy consumption. This trend makes it possible to develop more complex and economically viable applications [3].

B. Internet of Things

Internet of Things (IoT) can be defined as a network of small, low-cost, low-power electronic devices where communications occur without human intervention. Each device acts as a "smart node" on the network, detecting information and performing low-level signal processing to filter out noise signals and to reduce the bandwidth required for node-to-node communications. Finally, nodes send this information to the "cloud" in a secure way to protect, store and process data. Analysts expect IoT to grow to approximately 36 billion connected devices by 2020 [4]. Currently there are numerous applications for IoT of which the smart homes, wearables, among others stand out.

Smart home is the most searched IoT related subject on Google. In smart homes, you can turn on the air conditioning before you get home or turn off the lights after you have left home. It is also possible to unlock the doors to friends for temporary access, even when we are not at home. Smart homes are making life simpler and more convenient. Products associated with Smart Home are promised to save time, energy and money.

Another great subject is Wearables. Wearable devices contain sensors and software that collect data and information about users. This data is subsequently processed to extract a user profile. These devices mainly cover fitness, health and entertainment. The IoT prerequisite for wearable applications is to be highly energy efficient or ultra-low power and small [5].

In the industry there is Industrial Internet of Things (IIoT). It is to enable industrial engineering with sensors, software and large analytical data to create intelligent machines. The big goal behind IIOT is that, make machines smarter and more accurate and consistent than humans in data communication. This can help companies solve inefficiencies and problems more easily [6].

II. Wireless Sensor Network

A sensor network is characterized by the ability to monitor one or more variables of interest in a particular event, such as distance, direction, speed, humidity, wind speed, temperature, motion, vibration, light intensity, seismic activity, Weight, pressure, among others [8]. WSN are suitable for situations where a wired solution cannot be implemented and where instant access to information is desired. Nodes can move along with observed phenomenon. As an example of this, we have sensors that are placed on animals to observe their behavior.

In the following sections it will be presented some features of an WSN as well as some factors that determine or influence the implementation of an WSN, some applications and practical cases.

A. Characteristics

The WSN are quite comprehensive and dynamic. This high level of flexibility requires mechanisms with large adaptability. For this to happen, the cluster architecture is used [7]. Cluster by definition, in the computational area, is a system consisting of a set of computers or autonomous processing units interconnected in order to work together for a purpose [9]. Clusters are ideal for organizing large amounts of elements that are located in a large area, thus providing excellent flexibility. Usually in a WSN there are 3 types of elements [7]: 1) Slave; 2) Master; and 3) Sink.

The slave aims to collect data from the sensors and transmit this data to the network.

The master node has the function of monitoring all existing slave nodes. The great feature of these elements is the ability to cross data from various sensors. These nodes can receive a massive amount of information from their slaves which will cause a significant drop in network performance. This implies that the processing of data has to be optimized [10].

The sink node is an element that connects to another network with the WSN. In this way the sink is a very important element because if this node stops working, the whole network is incommunicable with the outside [7].

B. Aplications

WSN can be used for a wide range of applications from the most diverse areas such as agriculture, military, health, domestic applications, among others.

Smart Homes - When talking about sensor networks in homes, IoT is soon associated. It is very common to apply a smart network to home automation. It is very comfortable to receive notification that the lights in your house are still on when you leave your home. Another possibility is to allow easily monitor the house and control the locks of the doors, making the residence theft proof. The system can be configured to automatically lock the doors when you are leaving the house or unlock them when you are approaching the main entrance. It is possible to monitor any type of temperature increase in the house using thermostats. The hardware is placed in and around home, the nodes interact with smartphones and define the appropriate applications based on their proximity to installed nodes [16].

Agriculture - Another large area where WSN is applied is in precision agriculture. With a WSN it increases efficiency, productivity and profitability, minimizing impacts on wildlife and the environment [17]. Real-time information provides a solid foundation for farmers' strategies.

Beckwith et al., Implemented a WSN in a vineyard. This network contained 65 nodes whose goal was to collect temperatures for six months. The information was used to analyze two important parameters in wine production: sum of heat and damages caused by frost [18].

Industry - The food industry should follow high quality standards in order to be competitive and to meet the needs of consumers. The quality of food products can change rapidly because they are subjected to a variety of hazards during production, transportation and storage, which compromises quality [19]. Perishable food products such as vegetables, fruits, meat or fish require refrigerated transports. Therefore, temperature is the most important factor when extending the useful life of this type of food. Studying and analyzing the temperature inside refrigeration rooms, containers and trucks is the main concern of the industry. The use of a WSN in refrigerated vehicles is a solution to end the lack of temperature information during the trip [19]. The vehicles contain a variety of sensors to detect, identify, record and communicate what happens during the journey in order to control the status of perishable goods in transport.

III. Bluetooth Low Energy

Bluetooth is now an indispensable technology in a mobile phone, thus allowing mobile devices to communicate with each other over short distances. In particular, Bluetooth Low Energy (BLE) is essential for small mobile devices because it requires very little power to operate.

BLE main characteristics -The purpose of BLE is to create a new class of small devices, which only need to send or receive a small amount of data frequently. To maintain power consumption relatively low, BLE has a transmit power (0.01 to 10mW) lower than standard Bluetooth (100mW for Class 1 devices and 1mW for Class 3) [21]. The data is sent in the same way, but with a slower speed, the maximum of 100kbps. BLE devices can switch between standby mode and active mode faster than standard Bluetooth, thus saving energy, allowing small bursts of data to be transmitted.

Many more benefits are behind the BLE. Currently there is a huge market for the ultimate consumer, as is the case with fitness wristbands. To get an idea this is the most common use for BLE, 96% of fitness devices have BLE [22].

Compared with other wireless technologies, the rapid growth of BLE is relatively easy to explain. This growth is directly related to the evolution of smartphones, tablets and mobile communications. The adoption of BLE from the beginning was made by the two biggest market leaders in the area of smartphones: Apple and Samsung. The first smartphone

that makes use of BLE was the iPhone 4S, released in October 2011 [20]. Although the market for smartphones and tablets is perfectly mature, there is a need to create connectivity with the outside world.

BLE Stack - Like the classic Bluetooth [23], the stack of BLE is composed of 3 parts [24]: 1) Controller; 2) Host; and 3) Application. The controller is typically the physical device that can transmit and receive radio signals. The host is usually a software layer that manages how two or more devices communicate with each other. The application layer is the most superficial layer and is the one that contains the logic and the interface with the user. The architecture of this layer depends very much on the application for which it was developed.

Network topologies The typology of the BLE network is relatively simple. There are 2 scenarios:

1) piconet - In the piconet, one or more slaves are connected to a single master. To identify each slave separately, each slave is assigned a temporary 3 bit address to be used when it is active. This means piconet can have maximum 8 devices. This includes 1 master and 7 slave bluetooth devices.
2) scatternet - Scatternet is a network which at least is made by combining two or more piconets. In this network, the devices which act as master and slaves, for example, when a certain device in the piconet acts as master or a slave by participating with another piconet which also acts as a master or a slave (Fig. 1). Scatternet primarily acts as a bridge between different piconets [25].

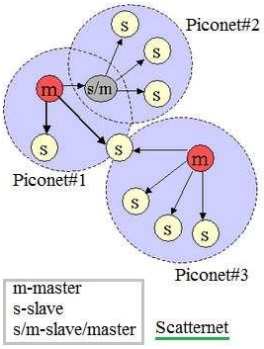


Fig. 1 Scatternet network [25]

IV. SMART BLE SENSOR NETWORK

The main objective of this work is to implement a system to monitor, through the internet, several physical quantities in a vehicle. A local network based on BLE and CAN technologies is implemented, which will gather information from several sensors and control different actuators located in truck deliver. Also for tracking of goods. The solution incorporates sensors in a vehicle to collect data about the trip such as: inertial sensors, temperature sensors, light sensor, limit switches and others. The system also incorporates a CAN bus and a GPS to get more data about the trip. From the CAN bus it is possible to measure driving styles and monitor consumption [26]. The system architecture is very flexible since it allows to add or remove sensors easily. It is a low-power solution that allows to wirelessly transmit the information to the cloud. All this information will be stored in the cloud for monitoring purposes.

A. Architecture and System Design

The proposed system, Fig. 2, is a system of up to 7 nodes, each one with a sensor to measure a physical quantity and/or actuator. After this information is captured, it is sent via wireless to the master node, which is responsible for receiving information from its slaves. The central module has a GPS to determine its location and an interface with the cloud, where it is possible to view data from any network sensor. The block called On-Board Diagnostics (OBD-II) is the CAN bus existing in cars. This network can provide a lot of information such as: instant speed information, engine speeds, etc [27]. This information is sent and stored in the cloud.

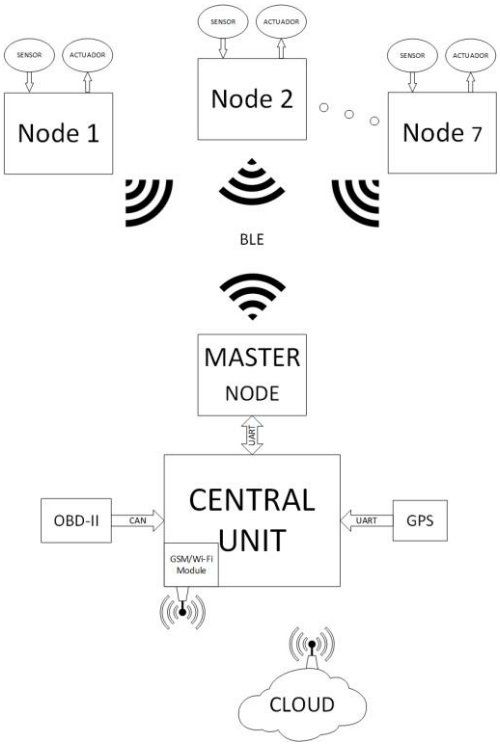


Fig. 2 System Architecture

Note that, currently the system is limited to 7 devices because it is a piconet topology, however, it can be transformed into a scatternet [7].

Slave Nodes - The proposed system has various slave nodes that have the capacity to measure various physical quantities such as: luminosity, temperature, atmospheric pressure, acceleration, etc. Each slave element in the network contains a SoC (System-on-a-Chip) nrf51822. This SoC is suited for Bluetooth low energy and 2.4GHz ultra low-power wireless applications. The SoC nrf51822 contains a 32-bit ARM Cortex M0 CPU with 256kB/128kB flash + 32kB/16kB RAM for improved application performance. It has several communication peripherals, namely Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C) and Universal Asynchronous Receiver/Transmitter (UART). It also has 31 pins of General Purpose Input/Output (GPIO) and 10-bit

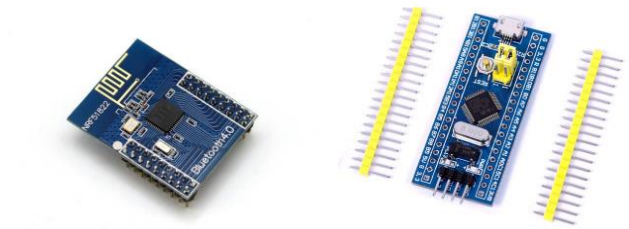
Analog Digital Converter (ADC). All these characteristics made this SoC the right choice considering performance, low energy and low cost [28]. The DS18B20 will be used to measure the temperature. This sensor operates with the OneWire protocol, allowing the connection of several sensors in a network. With this it is possible to acquire temperature data of several points in the physical space through only one pin. To quantify the brightness a Light Dependent Resistor (LDR) will be used. This is a resistive sensor, so a variation in brightness results in a variation in its resistance.

For this project it was chosen the MPU-6050 which is a chip that contains an accelerometer and a gyroscope together. It is displayed as a 6-axis low power MEMS sensor, 3-axis for the accelerometer and the remaining for the gyroscope. With a scale of up to 16g and 2000º/Sec [29].This sensor has I2C interface that allows data acquisition up to 400kHz in fast mode.

Master Node - The master node receives sensors information via wireless from the slave nodes and is always ready to respond to requests from the central unit. The task of this node is to gather all the information of its slaves, to send later to the central unit. Centralizing all the sensor information in the central unit. The hardware is pretty simple, this node only has a SoC nrf51822.

Central Unit - The central unit is the brain of the entire network. The function of this device is to gather all information from the sensors, GPS, information available on CAN-BUS (vehicle) and send it in a coordinated way to the cloud. Here we have a GPS sensor whose function is to determine the location periodically. The microcontroller that processes all this information is the STM32F103 [30]. The GPS module used is the uBlox NEO-6M. The choice of this module is due to the fact that it contains the UART protocol for communication [31] and also has the standard ISO 16750 [32].

A serial port is used to connect the microcontroller to the GSM or Wi-Fi module. This module is responsible for sending all information via wireless to the cloud. All the information related to the vehicle trip will be stored in the cloud and will be available in any place in the world.



a) b)

Fig. 3- a) slave node nrf51822 b) Master node STM32F103 [32]

Cloud - The Information of all elements in the network is sent to the cloud. This enables remote monitoring, and the configuration of the entire system. It is not the responsibility of this system to host the cloud.

B. Software

To explain the operation of the whole system it is presented a brief description of the software architecture.

Slave Nodes - The software for slave nodes always follows the same principle. In the Fig. 4 is presented a diagram where it is possible to see all main steps. For each node, there are different types of sensors, they use different communication protocols and therefore the microcontroller needs a peripheral that allows it to communicate with the sensor. This is the first step, initialize the necessary peripheral. In this step, the microcontroller pins are also configured for the outputs/inputs. Still here, the BLE stack, the services and other parameters necessary for the perfect operation of the BLE are initialized.

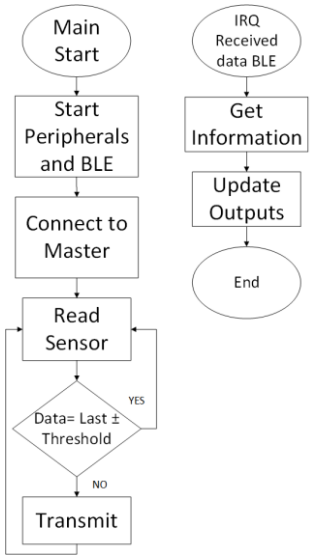


Fig. 4 Diagram of Software (Slave)

The next step represents the connection of the slave to the master, starting with the advertising mode and wait for the connection to be established.

Once gathered the sensor data, it is compared with the previous reading plus a threshold value, if the actual reading is bigger than the previous reading plus a threshold value, the slave transmits the data to the master otherwise it stars a new reading. it is also possible to define a sampling rate. Whenever the slave receives some information from the master node, it comes via BLE and this causes an interrupt service routine to be executed. At that time, the message is proceed and then the outputs updated.

Master Node - The software for the master is relatively simple (Fig. 5). In the first step the BLE stack and the peripheral UART are initialized. UART is the communication module used to exchange data between the master and the central unit. After all slaves are connected, the master waits for messages from the slaves. After receiving some information this will save the value and prepare the sending via UART. The reception of information is made by interruption. In this interruption is given an "ok" to send this same information by UART in the main cycle. This information goes to the central unit to be sent to the cloud.

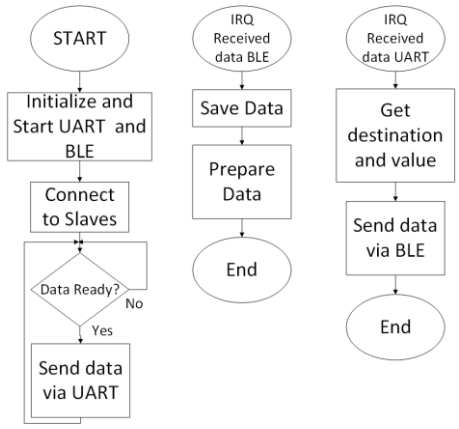


Fig. 5 Diagram of software (master)

If the master receives information from the UART it means that some slave node output must be updated. After receiving the message, it is necessary to identify the target slave and the logical level, and then transmit the information to the slave.

Central Unit -In the Fig. 6 it is possible to see the central unit diagram where it is described all the steps. Like the master and the slave nodes, the peripherals are also initialized, in this case the UART and CAN. UART is the module used to communicate with the GPS module and the CAN module is used to gathered information from the vehicle.

After initializing the peripherals, the central unit receives data from master node, the GPS and finally the CAN bus can provide detailed vehicle data or used to transmitted data using existing cable system in the vehicle or truck. The program main cycle will poll all data ready flags and then sends the data to the cloud.

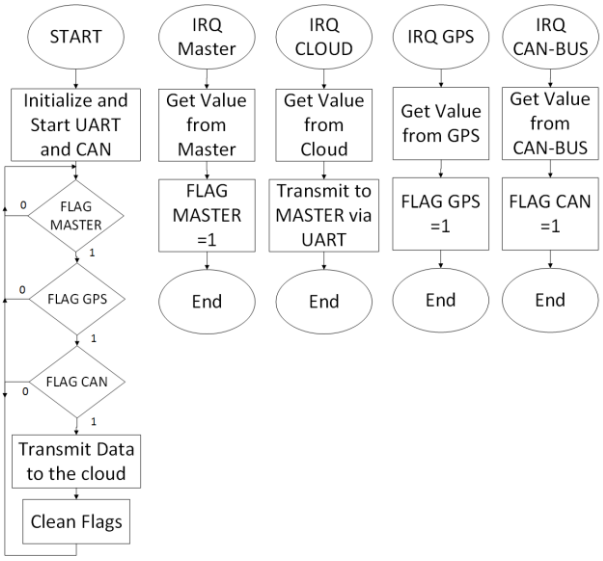


Fig. 6 Diagram of software (Central)

V. URBAN DISTRIBUTION OF GOODS IN A CITY ENVIROMMENT

IoT is applied to track delivery process and to control drivers behavior based on the analyses of fuel consumption, acceleration, breaking process, engine rotation. This proposed network allows the implementation of a low cost sensing because there is no need of cabling and sensors are available at low prices. It is possible to monitor temperature with special delivers, follow the goods and operator logistics can monitor their driver behavior at low cost solution. This delivers truck are becoming a part of the broader IoT logistics ecosystem that could transform trucking and the logistics industry. These trucks are becoming a mobile node in the IoT, real time connection between the delivers stakeholders (driver, sender, logistics operator and receiver). Optimization process can be performed in real time. Connected with high-speed cellular Internet data from sensors on trucks, dispatchers can measure real-time fuel efficiency, and how drivers operate their vehicles — do they brake too late, or waste diesel on lead-foot starts? Fleet management systems can recommend the optimal speed for a route [26].

Telematics sensors in trucks and multisensory tags on items transmit data on location, condition and whether a package has been opened. Sensors that measure the capacity of each load can provide additional insight into spare capacity in vehicles. The IoT data could then populate a central dashboard that focuses on identifying spare capacity on particular routes or destination pairs and analytics could recommend suggestions for consolidating and optimizing the route.

MoDe (Maintenance on Demand) is a EU-backed research project between Volvo, DHL and other partners sought to create a commercially viable truck that decides autonomously when and how it requires maintenance. The latest sensor technology has been incorporated at several points, such as oil systems and shock absorbers, to identify degradation or property damage. The data is then transmitted first to a central unit in the truck via a wireless network, then to a maintenance platform for analysis. The driver or maintenance teams are alerted to possible problems. [33]

VI. CONCLUSION

This paper presented an IoT system to collect data from vehicle that can be apply of the goods distribution process in a city, where there is always network connectivity at low prices. Allows the easy data collection from several sensors that can be implemented in a distribution truck to monitor in real time this process. IoT allows important payoffs for logistics operators and their business customers and end consumers by a real time deliver monitor process without big investments associated.

Also this network allows easy installation process because there is no need of cable and a diversity of data can be collected based on the type of sensor. This is part of academic research but there is a big potential of this solution of different commercial environments.

REFERENCES

[1] M. Leopold, “Sensor Network Motes :Portability & Performance,” no. December, pp. 1–123, 2007.

[2] K. Chelli, “Security Issues in Wireless Sensor Networks : Attacks

and Countermeasures," vol. I, 2015.

[3] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," *Computer (Long. Beach. Calif).*, vol. 37, no. August, pp. 41–49, 2004.

[4] W. Bluetooth and S. Sub, "Wireless connectivity for IoT applications," 2015.

[5] Ciara O'Brien, "Wearables: Samsung chases fitness fans with Gear Fit 2," 2016. [Online]. Available: http://www.irishtimes.com/business/technology/wearables-samsung-chases-fitness-fans-with-gear-fit-2-1.2763512. [Accessed: 04-May-2017].

[6] TIM HAWKINS, "An edge-to-cloud continuum: Inside GE Digital's Predix Industrial Internet of Things platform | #theCUBE - SiliconANGLE," *Silicon Angle*, 2016. [Online]. Available: https://siliconangle.com/blog/2016/08/22/an-edge-to-cloud-continuum-inside-ge-digitals-predix-industrial-internet-of-things-platform-thecube/. [Accessed: 04-May-2017].

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[8] D. Communication, *Wireless Sensor and Actuator Networks Algorithms and Protocols for Scalable Coordination*. Wiley, 2010.

[9] C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham, and F. Sommers, "Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers," *Handb. Nature-Inspired Innov. Comput. Integr. Class. Model. with Emerg. Technol.*, pp. 521–551, 2006.

[10] T. Schoellhammer, B. Greenstein, and D. Estrin, "Hyper: A Routing Protocol To Support Mobile Users of Sensor Networks," *Cent. Embed. Netw. Sens.*, Jan. 2006.

[11] M. Staroswiecki, "On fault tolerant estimation in sensor networks," *... Eur. Control Conf. Cambridge, UK*, no. 5, pp. 1–6, 2003.

[12] G. Hoblos, M. Staroswiecki, and A. Aitouche, "Optimal design of fault tolerant sensor networks," *IEEE Int. Conf. Control Appl.*, pp. 467–472, 2000.

[13] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan, "Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks," *Proc. 7th Annu. Int. Conf. Mob. Comput. Netw. - MobiCom '01*, pp. 272–287, 2001.

[14] A. Woo and D. E. Culler, "A transmission control scheme for media access in sensor networks," *Proc. ACM 9th Int. Conf. Mob. Comput. Netw.*, pp. 211–235, 2001.

[15] J. Chilo, C. Karlsson, P. Ängskog, and P. Stenumgaard, "EMI disruptive effect on wireless industrial communication systems in a paper plant," *IEEE Int. Symp. Electromagn. Compat.*, pp. 221–224, 2009.

[16] Shubhi Mittal, "IoT and Home Automation: How Beacons are Changing the Game | Beaconstac," 2015. [Online]. Available: https://blog.beaconstac.com/2015/11/iot-and-home-automation-how-beacons-are-changing-the-game/. [Accessed: 02-May-2017].

[17] H. R. Bogena, M. Herbst, J. a. Huisman, U. Rosenbaum, a. Weuthen, and H. Vereecken, "Potential of Wireless Sensor Networks for Measuring Soil Water Content Variability," *Vadose Zo. J.*, vol. 9, no. 4, p. 1002, 2010.

[18] R. Beckwith, D. Teibel, and P. Bowen, "Report from the field: results from an agricultural wireless sensor network," *29th Annu. IEEE Int. Conf. Local Comput. Networks*, pp. 471–478, 2004.

[19] Q. Shan, Y. Liu, G. Prosser, and D. Brown, "Wireless intelligent sensor networks for refrigerated vehicle," *Proc. IEEE 6th Circuits Syst. Symp. Emerg. Technol. Front. Mob. Wirel. Commun. (IEEE Cat. No.04EX710)*, vol. 2, pp. 525–528, 2004.

[20] T. O'Brien, "iPhone 4S claims title of first Bluetooth 4.0 smartphone, ready to stream data from your cat," *engadget*, 2011. [Online]. Available: http://www.engadget.com/2011/10/12/iphone-4s-claims-title-of-first-bluetooth-4-0-smartphone-ready/. [Accessed: 02-Mar-2016].

[21] Bluetooth Special Interest Group, "Specification of the Bluetooth System Covered Core Package Version 4.0," vol. 0, no. June, p. 2302, 2010.

[22] M. Ryan, "Outsmarting Bluetooth Smart," *CanSecWest*, 2014.

[23] A. . Fallis, "Final Report on the Inter-Agency Workshop on Research Issues for Smart Environments," *J. Chem. Inf. Model.*, vol. 53, no. 9, pp. 1689–1699, 2013.

[24] R. Heydon and N. Hunn, "Bluetooth Low Energy," *... , Bluetooth SIG https//www. bluetooth. org/ ...*, 2012.

[25] RF Wireless World, "Piconet vs Scatternet-Difference between Piconet,Scatternet," 2015. [Online]. Available: http://www.rfwireless-world.com/Terminology/difference-between-piconet-and-scatternet-in-bluetooth.html. [Accessed: 06-May-2017].

[26] J. Ferreira, J. Almeida, A. Silva. The Impact of Driving Styles on Fuel Consumption: A Data Warehouse and Data Mining based Discovery Process, IEEE Transactions on Intelligent Transportation Systems Magazine on Intelligent Transportation Systems in March 2015. DOI (identifier) 10.1109/TITS.2015.2414663

[27] SAE Standard, "SAE J1979: E/E Diagnostic Test Modes," *SAE Stand.*, vol. 552, no. 1, 2002.

[28] Waveshare, "Core51822 - Waveshare Wiki," 2016. [Online]. Available: http://www.waveshare.com/wiki/Core51822. [Accessed: 06-May-2017].

[29] Dynastream Innovations Inc., "ANT Message Protocol and Usage," pp. 1–134, 2013.

[30] O. Rtc, A. Autonomous, H. Gps, A. Online, A. Offline, O. M. A. Supl, G. P. S. Solution, T. Neo-, T. Ddc, A. Neo-, M. Type, and S. Interfaces, "NEO-6 series," pp. 6–7.

[31] P. Code, "International Standard ISO 16750," *Shock*, vol. 2003, pp. 1–8, 2002.

[32] Roger Clark, "STM32F103 and Maple / Maple Mini with Arduino 1.5.x IDE | Roger Clark," 2016. [Online]. Available: http://www.rogerclark.net/stm32f103-and-maple-maple-mini-with-arduino-1-5-x-ide/. [Accessed: 06-May-2017].

[33] "FP7 - Maintenance on Demand - MoDe." [Online]. Available: http://fp7-mode.eu/. [Accessed: 08-Jul-2017]