Mestrado em Engenharia Eletrotécnica

GRETDAI – *Gateway* de recolha e tratamento de dados em ambiente industrial

Relatório de Estágio apresentado para a obtenção do grau de Mestre em Engenharia Eletrotécnica – Área de Especialização em Automação e Comunicações em Sistemas Industriais

Autor

David Varanda Caniceiro

Orientadores

Doutor Fernando José Pimentel Lopes

Doutor Inácio de Sousa Adelino da Fonseca

Instituto Superior de Engenharia de Coimbra

Supervisor na Empresa

Engenheiro Carlos Martinho Simões Oliveira

Enging – Make Solutions, Lda

Coimbra, novembro, 2018

AGRADECIMENTOS

A realização deste relatório de estágio contou com a colaboração de várias pessoas, que me ajudaram a cumprir os meus objetivos e ainda com importantes apoios e incentivos, sem os quais não se teria tornado uma realidade e aos quais estarei eternamente grato.

À Enging – Make Solutions, Lda., por me ter dado a oportunidade do estágio curricular, bem como todas as condições técnicas e laborais durante o estágio, aos seus colaboradores, pelas suas opiniões e sugestões técnicas. Agradeço especialmente aos Engenheiros Martinho Simões e Carlos Martins pela supervisão, incentivo, motivação e pelo apoio técnico bem como conhecimentos transmitidos durante a realização do estágio.

Aos Professores Fernando Lopes e Inácio Fonseca, pela clareza, rigor e total disponibilidade no apoio prestado durante o estágio e na laboração do presente relatório, assim como na disponibilização de recursos para realização de testes.

Ao Instituto Superior de Engenharia de Coimbra, em especial o Departamento de Engenharia Eletrotécnica pela disponibilidade dos seus laboratórios e equipamentos.

Por último, tendo consciência que sozinho nada disto teria sido possível, dirijo um agradecimento especial aos meus pais, por serem modelos de coragem, pelo seu apoio incondicional, incentivo, motivação e paciência demonstrados e total ajuda na superação dos obstáculos que ao longo desta caminhada foram surgindo. Dedico ainda esta obra, aos meus avôs Abílio Varanda e Manuel Caniceiro.

A todas estas pessoas, o meu sincero obrigado.

David Varanda Caniceiro

RESUMO

Nos dias de hoje, os sistemas IoT estão cada vez mais presentes no nosso dia-a-dia, tanto em sistemas domésticos como em sistemas industriais, tal como se tem vindo a comprovar através da crescente popularidade da Indústria 4.0. O presente Relatório de Estágio pretende descrever o desenvolvimento de uma *Gateway* de Recolha e Tratamento de Dados em Ambiente Industrial (GRETDAI) para motores e transformadores elétricos durante o período de estágio curricular no âmbito do Mestrado em Engenharia Eletrotécnica – Área de Especialização em Automação e Comunicações em Sistemas Industriais, na empesa Enging – Make Solutions, Lda. A Enging é uma empresa criadora de produtos inovadores para a monitorização de equipamentos industriais, incluindo motores e transformadores elétricos.

São introduzidos os conceitos de IoT e Indústria 4.0, assim como uma apresentação da empresa Enging. São ainda introduzidos, os microcomputadores mais conceituados no mercado, bem como um breve estudo comparativo entre eles, e é efetuada uma pequena análise dos equipamentos para aquisição, recolha e tratamento de dados existentes no mercado.

Este relatório prossegue apresentando os elementos que constituem a proposta detalhada do sistema que se pretendeu desenvolver. O desenvolvimento do sistema GRETDAI foi divido em duas fases principais. A primeira fase refere-se aos diversos testes com *hardware*, através do desenvolvimento e teste de pequenos módulos de *software*. Nestes testes com o *hardware*, foram incorporados os vários protocolos de comunicação a usar no sistema. Na segunda fase é abordado o desenvolvimento do *software* principal onde são apresentados e descritos, através de fluxogramas, todos os algoritmos que implementam as funcionalidades do sistema. Esta segunda fase foi divida em três subfases: Versão Inicial, Versão Final e Versão Específica. O sistema da Versão Final foi testado em ambiente laboratorial para recolha e tratamento de dados de um transformador ligado à rede elétrica, e o mesmo sistema foi ainda testado em ambiente industrial para recolha e tratamento de dados de um transformador trifásico.

É ainda apresentada uma abordagem direcionada para a implementação industrial do sistema, que se foca na industrialização, certificação e comercialização. Por fim são apresentadas algumas considerações finais onde se inserem as conclusões deste trabalho, bem como possíveis melhorias e trabalhos futuros.

Palavras-chave: *Gateway*; Recolha de dados; Tratamento de dados; Motor Elétrico; Transformador.

ABSTRACT

Nowadays, IoT systems are increasing their presence in our daily lives, from domestic to industrial systems, what can be verified with the increasing popularity of Industry 4.0. This Internship Report intends to describe the development of a Gateway for Collection and Processing of Data in Industrial Environment (GRETDAI) for electric motors and transformers.

This work was performed during the internship period of the Master in Electrical Engineering – Automation and Communications in Industrial Systems Specialization Area, at Enging – Make Solutions. Enging is a company that creates innovative products for the monitoring of industrial equipment, including electric motors and transformers.

The concepts of IoT and Industry 4.0 are introduced first, as well as a short presentation of Enging. The best-known microcomputers in the market are described, together with a brief comparative study between them. We then present a short analysis of equipment that can be found in the market, for acquiring, collecting and processing data, including the elements that constitute them.

This report proceeds with the detailed proposal of the system that was intended to be developed. The development of the GRETDAI system was divided into two main phases. The first phase refers to several tests with hardware, carried out through the development of small software modules that incorporated the tests for the many communication protocols to be used in the system. The second phase consisted on the development of the main software where all the algorithms that implement the system's functionality are presented and described through flowcharts. This second phase was further divided into three sub-phases: Initial Version, Final Version and Specific Version. The Final Version system was laboratory tested for collecting and processing data from a transformer connected to the electrical network, and field tested in an industrial environment for collecting and processing data from a three-phase transformer.

An approach oriented to the industrial implementation of the system is also described in this report, focusing on its industrialization, certification and commercialization. Finally, some final considerations are made that include the conclusions of this work, as well as possible improvements and future work.

Keywords: Gateway; Data collection; Data Processing; Electric Motor; Transformer.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE QUADROS

SIMBOLOGIA E ABREVIATURAS

A – ampere

ADC – *Analog-to-Digital Converter*

ARM – *Acorn RISC Machine or Advanced RISC Machine*

ASCII – *American Standard Code for Information Interchange*

bps – *bits* por segundo

CAN – *Control Area Network*

CS – *Chip Select*

DAC – *Digital-to-Analog Converter*

DAQ – *Data Acquisition*

DDR – *Double Data Rate*

DHCP – *Dynamic Host Configuration Protocol*

DNS – *Domain Name System*

DTBO – *Device Tree Blob Object*

DTS – *Device Tree Source*

ETH –*Ethernet*

FAT – *File Allocation Table*

FPGA – *Field-Programmable Gate Array*

FTP – *File Transfer Protocol*

GND – *Ground*

GPIO – *General Purpose Input/Output*

GPS – *Global Positioning System*

HDMI – *High-Definition Multimedia Interface*

HTTP – *Hypertext Transfer Protocol*

Hz – *Hertz*

I/Os – *Inputs/Outputs*

I2C – *Inter-Integrated Circuit*

ICSP – *In Circuit Serial Programming*

IoT – *Internet of Things*

IP – *Internet Protocol*

JTAG – *Joint Test Action Group*

kS/s – *kiloSamples/second*

LCD – *Liquid Crystal Display*

LED – *Light Emitting Diode*

LSB – *Least Significant Bit*

MB – *Megabytes*

MISO – *Master In Slave Out*

mm – milímetro

MOSI – *Master Out Slave In*

MSB – *Most Significant Bit*

ns – nano segundo

PPS – *Pulses Per Second*

PROFIBUS – *Process Field Bus*

PRU – *Programmable Real-time Unit*

PWM – *Pulse-Width Modulation*

RAM – *Random Access Memory*

RISC – *Reduced Instruction Set Computer*

ROM – *Read-Only Memory*

RX – Receptor

SCLK – *Serial Clock*

SCSI – *Small Computer System Interface*

SFTP – *SSH File Transfer Protocol*

SO – Sistema Operativo

SoC – *System on Chip*

SPI – *Serial Peripheral Interface*

SRAM – *Static Random Access Memory*

SSH – *Secure Shell*

TCP – *Transmission Control Protocol*

TX – Transmissor

UART – *Universal Asynchronous Receiver/Transmitter*

USB – *Universal Serial Bus*

uSD – *micro Secure Digital*

V – *volt*

VGA – *Video Graphics Array*

W - *Watts*

1 Introdução

1.1 Motivação Pessoal e Enquadramento

Durante a formação académica, nomeadamente no projeto final da Licenciatura em Engenharia Eletrotécnica, o autor participou num projeto com o título "Sistema de Alta Disponibilidade para monitorização de PLCs", usando apenas *Raspberry's Pi* para a monotorização, tendo também elaborado pequenos projetos pessoais de cariz tecnológico. No entanto, havia a necessidade de contacto direto com o mercado de trabalho, para uma perceção exata da realidade laboral na sua área de formação.

Assim surgiu a escolha do estágio para obtenção do grau de Mestre em Engenharia Eletrotécnica, com a intenção de obter contacto e adquirir alguma experiência no mercado de trabalho. Depois de alguma pesquisa por empresas na área de automação e preferencialmente na região de Coimbra, foi contactada a Enging – Make Solutions, Lda., em que numa primeira entrevista foi proposto o desenvolvimento de um produto para a Indústria. A proposta suscitou interesse de imediato, por ser um produto na área dos sistemas embebidos e IoT (*Internet of Things*), onde já tinha alguma experiência laboral, dado o projeto realizado no âmbito do projeto final de Licenciatura em Engenharia Eletrotécnica, e também por permitir a participação no desenvolvimento de algo útil e com aplicação direta em contexto industrial. Todos estes aspetos contribuíram, em grande parte, para a confirmação do estágio na empresa em questão. A Enging já disponha de produtos de monitorização de correntes e tensões de motores e transformadores elétricos, mas havia a necessidade de um novo produto para a recolha e processamento dos dados adquiridos por placas de aquisição analógicas e digitais, e envio posterior dos dados processados para a *cloud*. Desta forma, a proposta de estágio consistiu no desenvolvimento de um sistema de recolha e processamento de dados.

1.2 Apresentação da Empresa

Fundada em 2011, a Enging define-se como uma empresa de engenharia especializada em desenvolver soluções industriais, tendo como principal objetivo ajudar os seus clientes nas diversas áreas de atuação, nas quais se inserem: Manutenção Preditiva, Soluções de Energia, Soluções de Segurança, Automação e Instrumentação, Redes e Infraestruturas de Comunicações, Consultadoria e Projetos de Engenharia e Formação.

A Enging, para além de ser criadora de um produto inovador para a monitorização de motores e transformadores, está constantemente focada na inovação por forma a entrar no topo este mercado.

1.3 Objetivos do Estágio

Para além das atividades operacionais da empresa, o principal objetivo deste estágio curricular consistiu no desenvolvimento de um sistema de recolha e processamento de dados, provenientes de placas de aquisição analógicas e digitais.

Este sistema resume-se no desenvolvimento de *software* para a recolha e processamento de dados e *interface* com o *hardware* de comunicação para a recolha de dados de sinais analógicos de tensões e correntes nos motores e transformadores elétricos, em meio industrial, com a finalidade de monitorizar estes sinais que se podem traduzir em comportamentos específicos e importantes dos motores e transformadores.

Deste modo, pretende-se recolher e processar os dados dos sinais analógicos de entrada e saída de correntes e tensões dos motores e transformadores, adquiridos pelas placas analógicas e outras variáveis adquiridas pelas placas digitais. Assim, pretendeu-se inicialmente ter contacto com a plataforma de desenvolvimento utilizada: BeagleBone Black, a linguagem de programação *Python* 2.7 e com os protocolos de comunicação RS-232, RS-485, SPI.

Desta forma, pretende-se simplificar ao máximo a arquitetura de *software*, de forma a permitir a sua utilização em diversas aplicações académicas e industriais. O sistema a desenvolver deve ainda permitir fomentar a experiência dos proponentes nesta área, para o lançamento deste tipo de aplicações.

1.4 Organização do Relatório

Este relatório de Estágio está dividido em 7 capítulos, tal como se sintetiza seguidamente:

- O Capítulo 1 contém a introdução ao projeto associado ao estágio, a sua contextualização, os objetivos, as metas a atingir e a organização do documento;
- O Capítulo 2 apresenta o Estado da Arte, que tem uma introdução à Indústria 4.0, bem como aos sistemas IoT. Descreve os sistemas de aquisição de dados utilizados no apoio à manutenção de motores elétricos, bem como as plataformas de desenvolvimento com potencial para desempenhar o papel de *gateway*, com ou sem sistema operativo incluído, e os protocolos de comunicação RS-232, RS-485 e SPI;
- O Capítulo 3 apresenta uma introdução à plataforma de desenvolvimento selecionada, bem como testes realizados com *hardware* e *software* para um primeiro contacto com o sistema;
- O Capítulo 4 apresenta o desenvolvimento do *software* para a aplicação final, descrevendo através de excertos de código e fluxogramas a sua lógica de funcionamento. É ainda feita uma descrição de *hardware* que foi desenvolvido pela empresa, para integração com o sistema final;
- O Capítulo 5 descreve os resultados obtidos em testes realizados na empresa, recolhendo sinais da rede elétrica para validação do sistema, utilizando o *software* desenvolvido, bem como testes realizados em ambiente industrial;
- O Capítulo 6 apresenta as conclusões deste trabalho e algumas sugestões para desenvolvimentos futuros;
- O final deste Relatório de Estágio inclui as referências bibliográficas e 3 anexos.

2 Estado da arte

2.1 Enquadramento

Neste capítulo pretende-se apresentar um resumo relativo aos sistemas de aquisição de dados utilizados no suporte à manutenção de motores elétricos. Como motivação para a monitorização de motores elétricos apresenta-se a estratégia de manutenção preditiva utilizada pela Enging. Introduz-se o conceito de DAQ (*Data Acquisition*) incluindo os sistemas existentes no mercado, para uma comparação com o sistema desenvolvido, assim como uma descrição sucinta da arquitetura e da interligação entre os dispositivos e elementos que constituem um sistema de aquisição de dados.

É feita uma pequena introdução à Indústria 4.0 e sistemas IoT, bem como ao conceito de uma *gateway* de IoT e também aos vários microcomputadores existentes no mercado, e a explicação da escolha do microcomputador *BeagleBone Black* para desempenhar o papel de *gateway* no sistema.

São ainda descritos os vários protocolos de comunicação, que são integrados no sistema desenvolvido, bem como a estrutura do mesmo.

2.2 Estratégia de Manutenção Preditiva

A manutenção preditiva está relacionada com as condições reais de funcionamento dos equipamentos com base em dados que informam o seu desgaste ou processo de degradação. Trata-se de um processo que prediz o tempo de vida útil dos componentes dos equipamentos e as condições para que esse tempo de vida seja aproveitado. Este período de vida útil tende a diminuir gradualmente com a idade e o tempo de utilização do próprio equipamento. A manutenção preditiva pode ser comparada a uma inspeção sistemática para o acompanhamento das condições dos equipamentos. No meio industrial ao submetermos um equipamento à sua manutenção, estamos a prolongar a sua vida útil o que permite ao seu proprietário garantir a sua confiabilidade e segurança, mas também aumentar a qualidade e eficiência dos seus processos o que se traduz numa inerente redução de custos. Existem três tipos de manutenção: manutenção corretiva, manutenção preventiva e manutenção preditiva. No sentido do objetivo do trabalho descrito neste relatório, abordaremos apenas a manutenção preditiva. Esta é considerada uma manutenção periódica cuja finalidade é prever avarias ou alterações do estado físico do equipamento até que se justifique a aplicação da devida ação de manutenção. Deste modo, a estratégia de manutenção da Enging consiste no registo dos valores de determinados parâmetros fundamentais no funcionamento do equipamento com o objetivo de prever o acontecimento de uma avaria nesse mesmo equipamento. Assim, uma grande parte das avarias que podem prejudicar o funcionamento normal do equipamento pode ser evitada aplicando este tipo de manutenção. A Enging criou um produto para monitorização de transformadores e motores elétricos, para as variáveis de correntes e tensões. No entanto, surgiu a necessidade de desenvolver um produto para adquirir e tratar os dados adquiridos

pelos produtos de monitorização, e posteriormente enviá-los para uma base de dados, para que um utilizador se mantenha informado sobre o estado dos seus equipamentos através dos dados recolhidos.

2.3 Sistemas de Aquisição de Dados

A aquisição de dados (DAQ) é o processo de medição de um fenómeno elétrico ou físico, como a tensão, a corrente, a temperatura, a vibração, a pressão ou som. Um sistema DAQ é formado por sensores, *hardware* de aquisição e medição de dados e um computador com *software* programável. Em comparação com sistemas tradicionais de medição, os sistemas DAQ baseados em computador exploram a capacidade de processamento, produtividade, sistemas de visualização e recursos de conectividades dos computadores padrão da indústria. O *hardware* DAQ atua como a *interface* entre um computador e sinais do mundo exterior. Ele funciona basicamente como um dispositivo que digitaliza sinais analógicos de entrada para que um computador possa interpretá-los. Os três componentes principais de um dispositivo DAQ usados para medir um sinal são os circuitos eletrónicos de condicionamento de sinais, conversor analógico-digital (ADC – *Analog to Digital Converter*) e o barramento do computador (Figura 2.1). Muitos dispositivos DAQ contêm outras funções, para a automação de sistemas e processos de medição. Por exemplo, conversores digital-analógico (DAC – *Digital to Analog Converter*) fornecem sinais analógicos, linhas de E/S digital que fornecem sinais digitais nas suas entradas e saídas, e os contadores/temporizadores contam e geram pulsos digitais [1].

Figura 2.1 – Esquema típico de um sistema de aquisição de dados [2].

Este tipo de sistemas proporciona uma grande vantagem na indústria, porque dão ao utilizar uma maior qualidade dos sinais e sistemas, redução de custos, maior desempenho de produção e de certo modo facilita a excelência operacional.

Nos dias de hoje, existe uma variedade de sistemas bastante robustos no mercado da área industrial, que permitem a aquisição e processamento de dados de diversas variáveis, como por exemplo os sistemas da *National Instruments* e *DATAQ Instruments*.

Na área da aquisição de dados a *National Instruments* possui plataformas de *hardware* modulares, vários tipos de barramentos de comunicações e ambientes de programação

(*software)*. As plataformas de *hardware* modulares são: PXI, CompactRIO e CompactDAQ. Estes dispositivos suportam vários tipos de barramentos de comunicação industrial, como por exemplo: Ethernet, PROFIBUS, LIN, EtherCAT, CAN e USB. Para permitir a comunicação com o *hardware*, controlo, análise, distribuição e publicação de dados, a *National Instruments* fornece ambientes de programação para desenvolver *software* para os diversos tipos de aplicações, como: LabVIEW, LabWindows™/CVI e Measurement Studio. A plataforma PXI (Figura 2.2) é utilizada na indústria para validação e teste de produção devido à sua capacidade de instrumentação modular de alto desempenho, temporização e sincronização [3].

Figura 2.2 – Plataforma de *hardware* PXI [3].

A plataforma CompactDAQ (Figura 2.3) pode ser utilizada tanto em laboratório como em campo, pois constitui um método simples de aquisição de dados de sensores para diversos tipos de sinais.

Figura 2.3 – Plataforma de *hardware* CompactDAQ [3].

Para uma aplicação de controlo e monitorização que requer um maior grau de complexidade, temos a plataforma CompactRIO (Figura 2.4). Esta possui diversas placas de controlo embebidas, processamentos de tempo real e módulos de entradas/saídas analógicas e digitais e ainda uma FPGA com entradas/saídas programáveis.

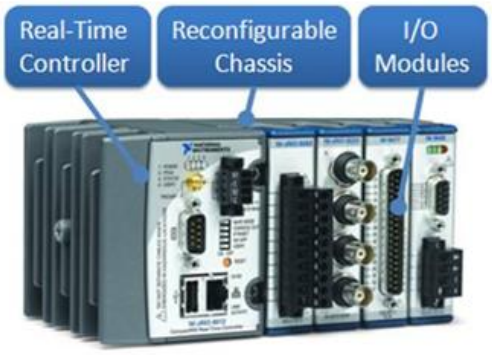


Figura 2.4 – Plataforma de *hardware* CompactRIO [3].

O CompactRIO também é uma plataforma modular, pelo que dispõe de módulos para aquisição de sinais de alta qualidade com condicionamento de sinais para medições específicas, isolamento e entradas/saídas com especificações industriais. Tem a capacidade de fazer vários tipos de medições, desde corrente, tensão e muitas outras grandezas. *O software* de *interface* com o *hardware* pode ser desenvolvido em LabVIEW ou C/C++. No entanto, devido à sua robustez e qualidade este tipo de sistemas são muito dispendiosos, tal como podemos verificar a título de exemplo (Figura 2.5), através de uma simulação de um sistema constituído por um cRIO-9022, 8 entradas analógicas de +/- 10 V com uma taxa de amostragem de 500 kS/s (*kSamples*/segundo) e resolução de 8 *bits*, 8 entradas analógicas de +/- 20 mA com uma taxa de amostragem de 200 kS/s e resolução de 16 bits e 2 módulos de 16 entradas/saídas digitais com uma taxa de atualização de 100 ns (nano segundos). Este sistema apresenta um custo estimado de 8.200 € [3].

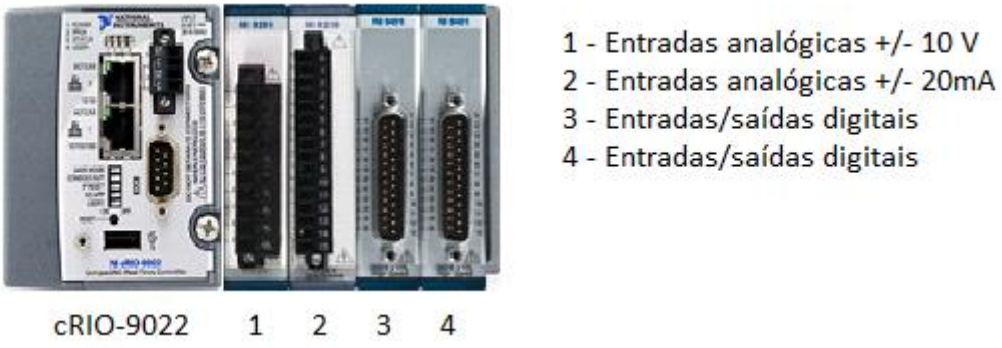


Figura 2.5 – Equipamento gerado através da simulação [3].

A *DATAQ Instruments* também desenvolve equipamentos para aquisição de dados com diversos tipos de medições, como por exemplo: tensão, corrente, entre outras (Figura 2.7). À semelhança da *National Instruments*, a *DATAQ Instruments* também utiliza uma topologia de *hardware* modular para o mais diverso tipo de aplicações e tem como *interface* um *software* específico para aquisição de dados, o *Windaq*. O *Windaq* é um *software* para sistemas operativos Windows que não exige programação, permite mostrar dados de vários canais simultaneamente, como também rever e analisar arquivos de dados

guardados, ajustar escala de cada canal e exportação para Excel. Este *software* é específico para cada tipo de equipamento existindo assim várias versões mais robustas e para altas taxas de amostragem possuem um custo elevado. No entanto realçam-se algumas diferenças comparativamente aos sistemas da *National Instruments*, nomeadamente: a *interface* de comunicação, uma mais baixa taxa de amostragem e resolução. Como *interface* de comunicação a *DATAQ Instruments* dispõe apenas de dois tipos de comunicação: USB e Ethernet [4].

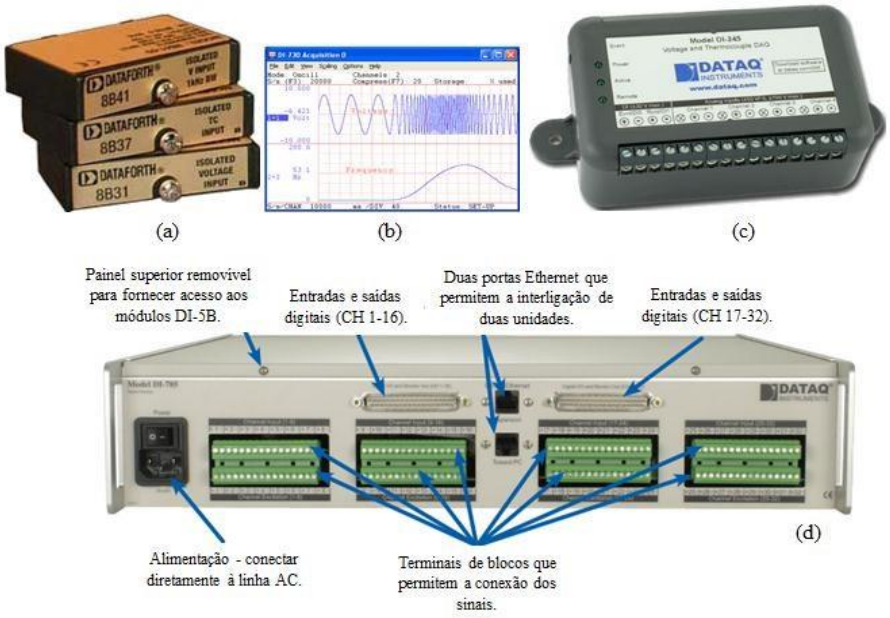


Figura 2.6 – Conjunto de equipamentos DATAQ: (a) módulos *DI-8B* amplificadores, (b) *software Windaq*, (c) módulos *DI-245* de aquisição de dados de tensão e termopares, (d) módulo *DI-785* de aquisição de dados de diversas variáveis [4].

Quando se pretende ter um sistema com várias funcionalidades, a cada funcionalidade corresponde um módulo específico. Assim, consoante o sistema a implementar, um determinado número de funcionalidades implica a introdução do mesmo número de módulos, o que faz aumentar o preço do sistema, e do ponto de vista financeiro estes sistemas tornam-se menos apelativos, uma vez que são muitos dispendiosos.

2.4 Indústria 4.0 e IoT

Num mundo cada vez mais digital, vivemos hoje uma verdadeira quarta revolução industrial, a Indústria 4.0, cujos principais objetivos, são a criação de um mercado único digital e a digitalização da indústria, através da maior automação dos processos e do uso corrente da IoT.

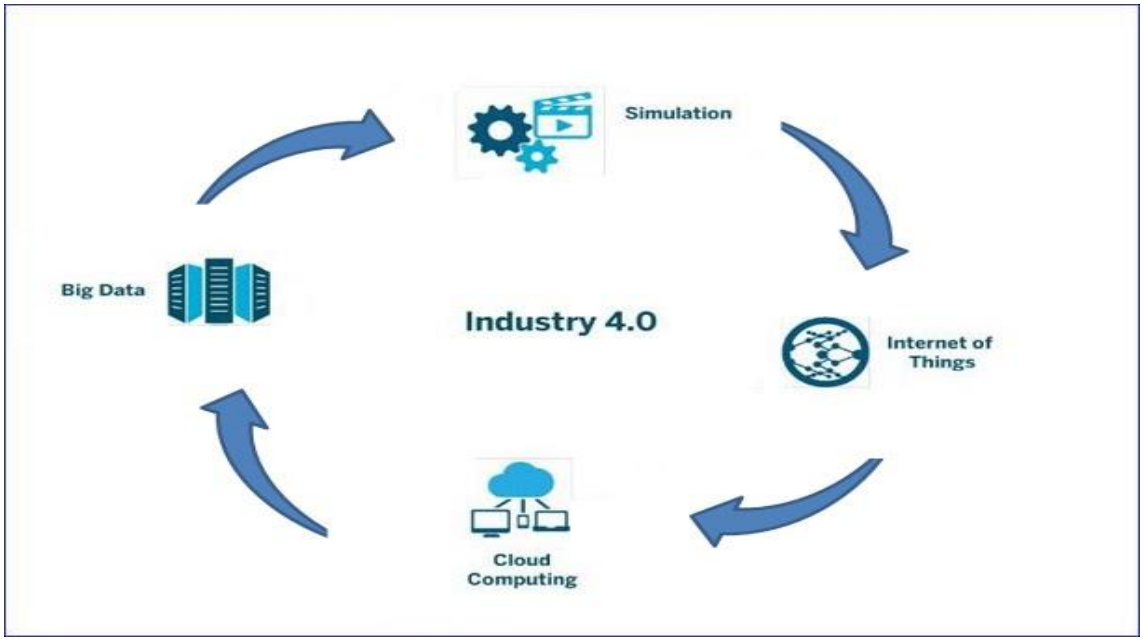


Figura 2.7 – Exemplo de estrutura da Indústria 4.0

As principais tecnologias que a Indústria 4.0 agrega são (Figura 2.7): *Big Data*, *Advanced Analytics (Simulation)*, *Cloud Computing*, IoT.

Big Data engloba os computadores de elevada capacidade e as redes de comunicação abrangentes e de baixo custo, que armazenam com rapidez uma grande quantidade de informação, que depois de tratada e analisada em tempo real, facilita tomar decisões com base nessa informação de valor com mais precisão e confiança.

Advanced Analytics (Simulation) trata-se de um conjunto de métodos e técnicas avançadas para ajudar na elaboração de previsões com base na informação (*Big Data*) e efetuar simulações e análise de cenários que permitam antecipar riscos, tomar decisões e otimizar processos.

Cloud Computing é um sistema informático em que o armazenamento de dados é efetuado em servidores especializados e cujo acesso à informação, serviços e programas é efetuado remotamente via *Internet*.

IoT é um conceito que significa interligar os diversos equipamentos do dia-a-dia, máquinas, equipamentos de transporte, eletrodomésticos, e mesmo pequenos objetos de uso diário à *Internet*, interagindo entre si e "lendo" o ambiente à sua volta através de dispositivos e sensores, transformando objetos estáticos em elementos dinâmicos de uma rede integrada, cujas centrais utilizarão essa informação de forma inteligente [5].

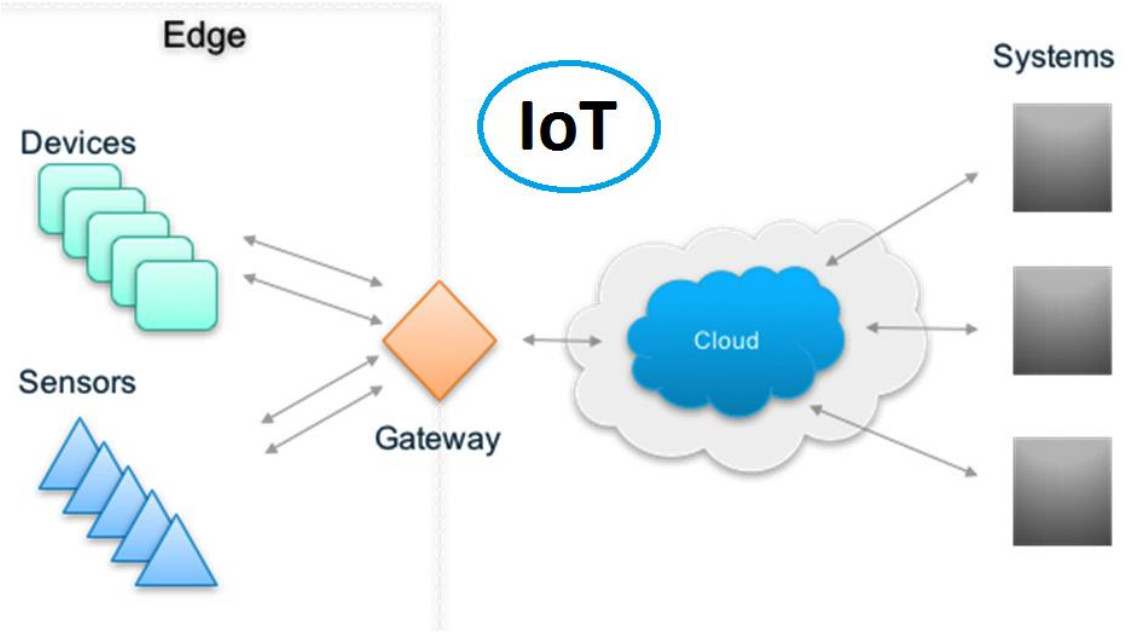


Figura 2.8 – Arquitetura de IoT [6].

À medida que a IoT cresce, muitos dispositivos precisam de se conectar à *Cloud*. Um dos componentes mais críticos dos sistemas da IoT é um dispositivo conhecido como *gateway* de IoT, que tem como principal função agregar dados dos sensores e dispositivos, processar e tratar os dados antes de enviá-los (Figura 2.8) [6].

2.5 Gateway de IoT

Gateway é uma máquina intermediária, por norma destinada a interligar redes, separar domínios de colisão, ou mesmo traduzir protocolos de comunicação. São componentes indispensáveis para alcançar comunicações entre terminais ligados a redes heterogêneas que usam protocolos diferentes (Figura 2.9).

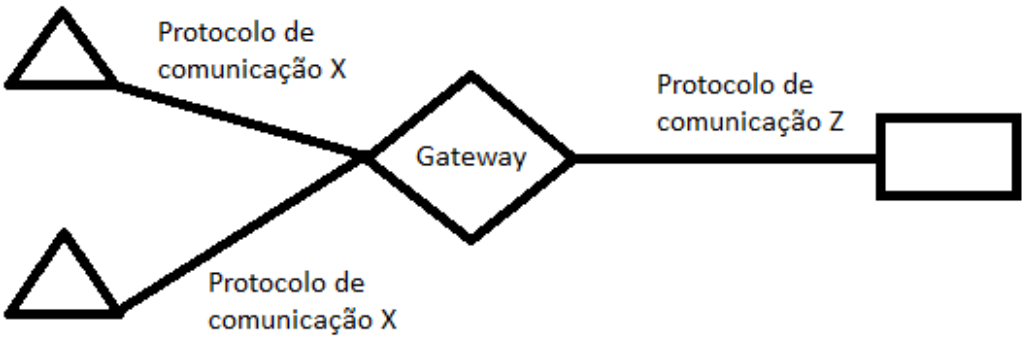


Figura 2.9 – Exemplo de definição de *gateway*.

Uma *Gateway* de IoT, é basicamente um dispositivo que serve como ponto de conexão entre os sensores e a *cloud*.

Com dezenas de protocolos, modelos de conectividade e perfis de energia e a natureza altamente dispersa dos sistemas IoT, as *gateways* são necessárias para gerir e controlar estes ambientes complexos. As *gateways* de IoT executam várias funções críticas, como conectividade do dispositivo, tradução de protocolos, filtragem e processamento de dados, segurança e atualização e outras. Os *gateways* de IoT mais recentes também operam como plataformas para código de aplicativo que processam dados e se tornam parte inteligente de um sistema habilitado para dispositivos [7].

O funcionamento da *gateway* de IoT traduz-se pela recolha de dados de sensores ou dispositivos no meio físico, onde são adquiridas informações do meio onde estão inseridos, pelo processamento e tratamento desses dados e pelo envio desses dados para a *cloud*. A *gateway* também pode receber informações da *cloud*, para ajudar o próprio dispositivo que interpreta o papel de *gateway* a ajudá-lo a executar funções necessárias, como a regulação das mudanças ambientais e a deteção de possíveis problemas com o funcionamento do sistema. Todas as informações que são transmitidas de um dispositivo IoT para a *cloud*, ou vice-versa, passam pela *gateway* de IoT. Ao gerir essa conexão, a *gateway* pode executar tarefas de segurança, ajudar a gerir dispositivos e traduzir diversos protocolos de comunicação [6].

2.6 Potenciais plataformas de desenvolvimento

Para assumir a papel principal no sistema a desenvolver, necessitamos de um microcomputador, que deve ter como características principais, ser de baixo custo, excelente relação qualidade-preço, ser de pequenas dimensões e conter como periféricos, vários entradas/saídas digitais e os periféricos de comunicação UART (*Universal Asynchronous Receiver/Transmitter*) e SPI (*Serial Peripheral Interface*).

As três placas de sistemas embebidos que satisfazem as condições acima mencionadas mais populares são: Arduino, Raspberry Pi e BeagleBone (Figura 2.10).

Figura 2.10 – Arduino Leonardo ETH [8], Raspberry Pi 2 Model B [9] e BeagleBone Black [10], respetivamente.

Dentro de cada uma destas marcas de placas de sistemas embebidos, existe uma basta gama de modelos, mas apenas selecionamos um modelo de cada, tendo em conta a relação de qualidade-preço e que satisfaça algumas necessidades essenciais, como por exemplo, possuir uma porta Ethernet, saídas e entradas Digitais, SPI e UART.

2.6.1 Arduino Leonardo ETH

O Arduino Leonardo ETH é uma placa de desenvolvimento baseada no microcontrolador ATmega32u4 e no novo W5500 TCP / IP Embedded Ethernet Controller. Possuí 20 pinos de entradas/saídas digitais, dos quais 7 podem ser utilizados como saídas PWM (*Pulse-Width Modulation*) e 12 como entradas analógicas, e 2 portas I²C (*Inter-Integrated Circuit*), 1 porta SPI e 1 UART, um oscilador de cristal de 16MHz, uma conexão RJ45, um conector micro USB (*Universal Serial Bus*), um ICSP (*In-Circuit Serial Programming*) *header* e um botão *reset* [8].

O Arduino é bastante básico comparando com as outras placas, e altamente extensível para aplicações IoT, que requerem dados a serem coletados por sensores e acionar outras ações. É uma placa simples de trabalhar, perfeita para pequenos projetos eletrónicos, como ligar um LED, sensores ou motores na placa diretamente. O Arduino possui um IDE (*Integrated Development Environment*) próprio (*software* Arduino), ou seja não possuí sistema operativo, e é através deste *software* que se envia o código fonte diretamente para o Arduino através de USB.

A maior vantagem do Arduino é a sua flexibilidade para se expandir, e mais importante, uma enorme base de suporte disponível na *Internet*, bem como, uma enorme diversidade de projetos já realizados.

2.6.2 Raspberry Pi 2 Model B

O Raspberry Pi 2 Model B, tem cerca de 6 vezes mais capacidade de processamento que os modelos anteriores. Esta segunda geração de Raspberry Pi vem com um processador BCM2836, que contém um ARM Cortex-A7 baseado num *quad-core* que "corre" a uma velocidade de 900MHz, e ainda possui uma RAM de 1GB.

O arranque do sistema operativo é feito a partir do cartão de memória Micro SD, que tem de conter uma versão do sistema operativo Linux, neste caso, mais concretamente, Raspbian ou NOOBS, este modelo também é compatível com o Windows 10.

Possui conectores de Audio e Video (HMDI – *High-Definition Multimedia Interface*), para as suas respetivas *interfaces*. Contando com 4 portas USB 2.0, uma porta Ethernet, e 40 pinos GPIO (*General Purpose Input/Output*), 27 dos quais de saídas e entradas digitais, dos quais podem funcionar ainda como *interface* para 1 UART, 1 porta I²C e 1 porta SPI. Não contém as funcionalidades de saídas PWM e entradas analógicas [11].

O Raspberry Pi 2 Model B pode ser considerado um mini computador *desktop*, onde o cartão de memória Micro SD possui o sistema operativo Linux e a memória externa. O Raspberry Pi é ideal para requisitos que precisam de um alto nível de conectividade direta

com a *Internet*, e para sistemas embebidos ou projetos que requerem mais interatividade e poder de processamento.

É relativamente barato, muito fácil de utilizar, e possui uma grande comunidade na *Internet* de apoio a projetos, e informação sobre a placa, bem como imensos projetos já realizados.

2.6.3 Beaglebone Black

A BeagleBone Black difere ligeiramente da versão regular, fornecendo-lhe uma porta micro HMDI, 512 MB de RAM, 4GB de memória *flash* integrada, um processador TI Sitara AM3358 ARM Cortex-A8 a 1GHz e ainda dois co-processadores (PRU – *Programmable Real-Time Unit*) para processamento em tempo real de 200 MHz, e JTAG (*Joint Test Action Group*) opcional.

A BeagleBone Black contém 65 pinos GPIO e 7 entradas analógicas, que podem funcionar em vários modos. Mais concretamente é possível ter os 65 pinos a funcionar como pinos de entradas e saídas digitais, 8 saídas PWMs, 4 *Timers*, 5 UARTs e 1 *serial header* dedicado para *debug*, 2 portas I²C, 2 portas SPI, 25 *low-latency* PRU I/Os e 20 para interação com a *interface* HMDI. Através das placas de expansão (*capes/shields*) micro HDMI ou VGA (*Video Graphics Array*) e LCD (*Liquid Crystal Display*), a BeagleBone Black é capaz de decodificar e exibir vários formatos de vídeo [10].

A BeagleBone Black é talvez a placa menos conhecida entres as três plataformas, é semelhante a um Raspberry Pi, mas com uma maior velocidade de processamento, com base no microprocessador TI Sitara AM3358, um processador de aplicativos SoC contendo um núcleo ARM Cortex-A8.

Esta plataforma vem integrada com o sistema operativo Linux e uma memória *flash*. Tem um ciclo de processamento muito rápido e vem com bastantes entradas e saídas digitais para conectar sensores e atuadores externos, e com estas características é bastante utilizado em IoT industrial, onde grandes quantidades de dados são rapidamente processados. A sua maior limitação é a sua única porta USB. A BeagleBone conta com uma grande comunidade de *developers* na *Internet*, para apoio a diversas questões, bem como projetos já realizados.

2.6.4 Conclusões

De acordo com as secções anteriores, pode-se concluir que temos 3 plataformas distintas, cada uma com vantagens e desvantagens, diferentes velocidades de processamento, diferentes modos de programação, mas com *interfaces* de comunicação bastante semelhantes (Tabela 2.1).

Tabela 2.1 – Tabela de comparação entre Arduino Leonardo [8], Raspberry Pi 2 [12] e BeagleBone Black [13].

Nome	**Arduino Leonardo**	**Raspberry Pi 2**	**BeagleBone Black**
Modelo	ETH R3	Model B	Rev C
Preço	43.89 €	31.89 €	43.07 €
Dimensões	53.34 x 68.58 mm	85 x 56 mm	86.36 x 54.61 mm
Processador	ATmega32u4	ARM Cortex-A7	ARM Cortex-A8
Clock Speed	16 MHz	900 MHz	1000 MHz
RAM	2.5 Kb	1 GB	512 MB
Flash	32 KB / MicroSD	MicroSD	4 GB / MicroSD
Input Voltage	7-12 V	5 V	5 V
Power Consumption	82mA (0.6W)	450mA (2.3W)	130mA (0.65W)
Digital GPIO	20	27	65
Analog Input	12	-	7
PWM	7	-	8
I²C	2	1	2
SPI	1	1	2
UART	1	1	5
Dev IDE	Arduino Tool	Raspbian, Arch Linux, …	Cloud9/Linux
Ethernet	10/100	10/100	10/100
USB Master	-	4 USB 2.0	1 USB 2.0

Em termos de velocidade de processamento e memória RAM o Arduino está em grande desvantagem comparando com as outras duas plataformas, e o Raspberry não possui qualquer memória interna, ao contrário da BeagleBone que tem 4GB de memória *flash*, e o Arduino uma pequena memória de 32 K*Bytes*. As três placas são de tamanhos idênticos, sendo o Arduino relativamente mais pequeno, mas possuindo menos pinos de entradas e saídas digitais e o Raspberry não tem pinos de entradas analógicas e saídas PWM.

Para iniciantes o Arduino é o mais recomendado, possui a maior comunidade de utilizadores, o maior número de tutoriais e projetos de amostra e é mais simples para *interface* com *hardware* externo.

Para aplicações que se conectam à *Internet*, o mais recomendado seria o RaspBerry Pi e a BeagleBone, apesar de esta versão do Arduino já conter uma porta Ethernet (ao contrário da maioria das versões do Arduino).

Tanto o Raspberry Pi como a BeagleBone são mini computadores Linux, ambos incluem *interfaces* Ethernet e USB, para que seja mais fácil conectá-los à rede, e é possível através da *interface* USB conectar módulos de rede sem fios, que se conectam à *Internet* sem fios.

Além disso, o sistema operacional Linux possui muitos componentes integrados que oferecem capacidades de rede bastante avançadas.

Para aplicações que interagem com sensores externos, os mais recomendados são o Arduino e o BeagleBone. O Arduino torna-se mais fácil de conectar a sensores externo porque opera em diferentes tensões, 3.3 V e 5 V, o BeagleBone só funciona com dispositivos de 3.3 V. Ambos possuem *interfaces* analógicas e digitais o que permite conectar facilmente componentes que produzem tensões variáveis. A BeagleBone possui conversores analógicos e digitais (ADC – *Analog-Digital Converter*) de resolução ligeiramente superior que podem ser úteis para aplicações mais exigentes. Com isto dito, é importante notar que muitos componentes utilizam as *interfaces* digitais SPI, I^2C e UART para se comunicar, e todas as placas possuem estes *interfaces* digitais.

Para aplicações que possam usar baterias como fonte de alimentação, o mais recomendado seria o Arduino, pelo seu baixo consumo em termos de potência, e o Arduino tem mais uma vantagem aqui, uma vez que pode funcionar com uma ampla gama de tensões de entrada, isto permite que ele funcione a partir de uma variedade de diferentes tipos de bateria.

Para aplicativos que usam *interface* gráfica, o Raspberry Pi e a BeagleBone são os mais recomendados, porque ambos têm uma saída HDMI, isto permite conectá-los diretamente a um monitor, e ainda conectar um rato e um teclado nas entradas USB, o que os torna em computadores totalmente funcionais com *interface* gráfica. Isto torna ambas as plataformas ideais para uso como um dispositivo de navegação Web de baixo custo, e para projetos onde utilizadores podem interagir com um *display* (LCD ou *Touch Screen*).

Em resumo, o Arduino é uma plataforma flexível com grande capacidade de *interface*. É uma ótima plataforma para aprender primeiro e perfeita para projetos menores. O Raspberry é bom para projetos que exigem *displays* ou conectividade à rede, e tem uma incrível relação de recursos/preço. A BeagleBone é uma ótima evolução relativamente ao Arduino mas com um processador mais rápido e o ambiente Linux completo do Raspberry Pi.

2.7 Seleção da plataforma de desenvolvimento

Depois de várias pesquisas e comparações, para exercer o papel de *gateway* foi escolhida a placa de desenvolvimento BeagleBone Black, os principais motivos foram: capacidade de processamento, velocidade de *clock*, armazenamento interno e bons recursos de entradas/saídas digitais.

O armazenamento interno de 4 GB da BeagleBone Black foi um fator importante na escolha, porque assim é possível ter o sistema operativo armazenado internamente, e depois usar um cartão de memória para armazenamento externo de dados, ao contrário do Raspberry Pi, onde tínhamos de criar partições num cartão de memória para o mesmo funcionar como sistema operativo e armazenamento externo.

O processador ARM Cortex-A8 com 1000 MHz de velocidade de *clock* da BeagleBone Black é um pouco superior ao do Raspberry Pi de 900 MHz, mas muito superior ao do Arduino de 16 MHz, tendo em conta que se estima que o *software* da *gateway* venha a ter vários processos a correr ao mesmo tempo, torna-se vantajoso ter um bom processador.

Em relação aos recursos de entradas/saídas digitais e *interfaces* de comunicação, a BeagleBone Black é bastante superior em comparação com as restantes placas de desenvolvimento, com 5 UART's, 2 SPI e 65 entradas/saídas digitais (Figura 2.11).

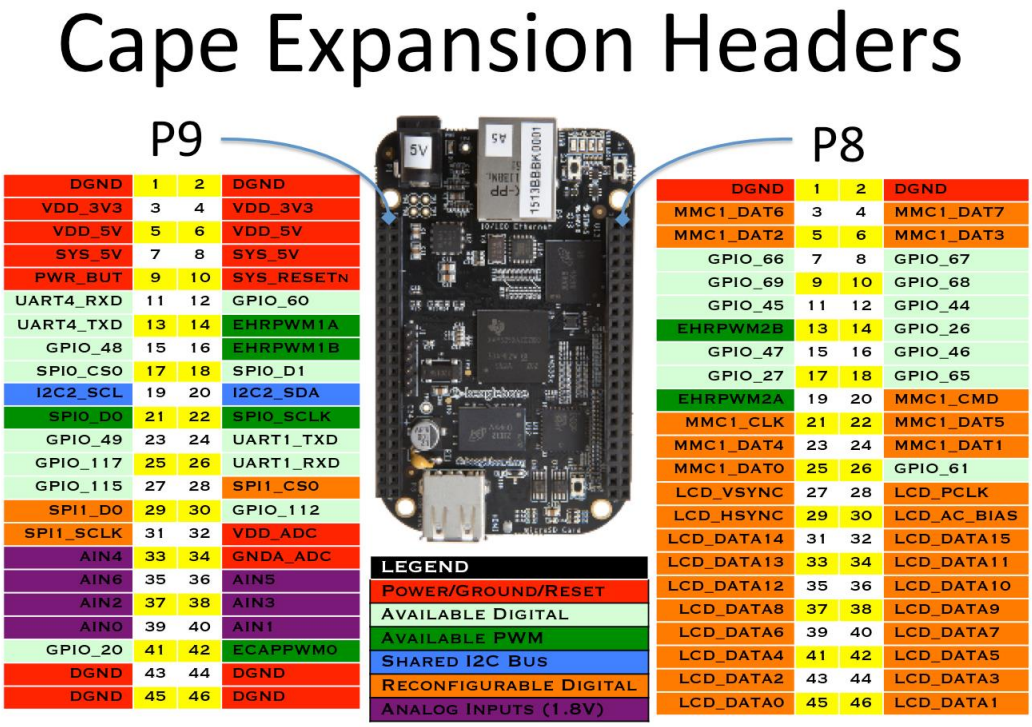


Figura 2.11 – *Pinout* da BeagleBone Black [13].

Como podemos, ver na Figura 2.11 do *pinout* da BeagleBone Black, só vemos 2 UART's, 1 SPI e 16 entradas/saídas digitais, isto porque todos os pinos funcionam como *multiplexers*, isto é, cada pino pode ter mais de uma função (Figura 2.12), à exceção dos pinos de alimentação, *ground*, *reset* (pinos a vermelho) e entradas analógicas (pinos roxo).

Table 11. Expansion Header P9 Pinout

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2						GND				
3,4						DC_3.3V				
5,6						VDD_5V				
7,8						SYS_5V				
9						PWR_BUT				
10	A10	SYS_RESETn	RESET_OUT							
11	T17	UART4_RXD	gpmc_wait0	mii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd		uart4_rxd_mux2	gpio0[30]
12	U18	GPIO1_28	gpmc_be1n	mii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir		mcasp0_aclkr_mux3	gpio1[28]
13	U17	UART4_TXD	gpmc_wpn	mii2_rxerr	gpmc_csn5	rmii2_rxerr	mmc2_sdcd		uart4_txd_mux2	gpio0[31]
14	U14	EHRPWM1A	gpmc_a2	mii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18		ehrpwm1A_mux1	gpio1[18]
15	R13	GPIO1_16	gpmc_a0	gmii2_txen	rmii2_tctl	mii2_txen	gpmc_a16		ehrpwm1_tripzone_input	gpio1[16]
16	T14	EHRPWM1B	gpmc_a3	mii2_txd2	rgmii2_td2	mmc2_dat2	gpmc_a19		ehrpwm1B_mux1	gpio1[19]
17	A16	I2C1_SCL	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci				gpio0[5]
18	B16	I2C1_SDA	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone				gpio0[4]
19	D17	I2C2_SCL	uart1_rtsn	timer5	dcan0_rx	I2C2_SCL	spi1_cs1			gpio0[13]
20	D18	I2C2_SDA	uart1_ctsn	timer6	dcan0_tx	I2C2_SDA	spi1_cs0			gpio0[12]
21	B17	UART2_TXD	spi0_d0	uart2_txd	I2C2_SCL	ehrpwm0B			EMU3_mux1	gpio0[3]
22	A17	UART2_RXD	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A			EMU2_mux1	gpio0[2]
23	V14	GPIO1_17	gpmc_a1	gmii2_rxdv	rgmii2_rxdv	mmc2_dat0	gpmc_a17		ehrpwm0_synco	gpio1[17]
24	D15	UART1_TXD	uart1_txd	mmc2_sdwp	dcan1_rx	I2C1_SCL				gpio0[15]
25	A14	GPIO3_21*	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4_mux2			gpio3[21]
26	D16	UART1_RXD	uart1_rxd	mmc1_sdwp	dcan1_tx	I2C1_SDA				gpio0[14]
27	C13	GPIO3_19	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU2_mux2			gpio3[19]
28	C12	SPI1_CS0	mcasp0_ahclkr	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out			gpio3[17]
29	B13	SPI1_D0	mcasp0_fsx	ehrpwm0B		spi1_d0	mmc1_sdcd_mux1			gpio3[15]
30	D12	SPI1_D1	mcasp0_axr0	ehrpwm0_tripzone		spi1_d1	mmc2_sdcd_mux1			gpio3[16]
31	A13	SPI1_SCLK	mcasp0_aclkx	ehrpwm0A		spi1_sclk	mmc0_sdcd_mux1			gpio3[14]
32						VADC				
33	C8					AIN4				
34						AGND				
35	A8					AIN6				
36	B8					AIN5				
37	B7					AIN2				
38	A7					AIN3				
39	B6					AIN0				
40	C7					AIN1				
41#	D14	CLKOUT2	xdma_event_intr1		tclkin	clkout2	timer7_mux1		EMU3_mux0	gpio0[20]
	D13	GPIO3_20	mcasp0_axr1	eQEP0_index		Mcasp1_axr0	emu3			gpio3[20]
42@	C18	GPIO0_7	eCAP0_in_PWM0_out	uart3_txd	spi1_cs1	pr1_ecap0_ecap_capin_apwm_o	spi1_sclk	mmc0_sdwp	xdma_event_intr2	gpio0[7]
	B12	GPIO3_18	Mcasp0_aclkr	eQEP0A_in	Mcaspo_axr2	Mcasp1_aclkx				gpio3[18]
43-46						GND				

Figura 2.12 – *Pinout* do *Header* P9 da BeagleBone Black [14].

2.8 Interfaces de Comunicação

Interfaces de comunicação são padrões lógicos e físicos em relação a como são ligados e transmitidos os sinais entre equipamentos e meios de comunicação, ou seja, se dois equipamentos partilharem a mesma *interface* de comunicação, do mesmo modo conseguem estabelecer uma comunicação entre si. São constituídos por conectores que têm um *pinout* específico onde são transmitidas as informações. Existem dois tipos principais de *interface* de comunicação: série e paralelo. São alguns exemplos de *interface* série: RS-232, RS-422 ou RS-485 (*Recommended Standard*), 1-Wire (*Onewire*), SPI, I2C e USB. Em relação à *interface* paralela são exemplos: ISA (*Industry Standard Architeture*), ATA (*Advanced Technology Attachment*), SCSI (*Small Computer System Interface*) e PCI (*Peripheral Component Interconnect*). No âmbito deste projeto serão abordadas apenas as *interfaces* de comunicação série: RS-232, RS-485 e SPI.

2.8.1 RS-232

O RS-232 for introduzido em 1962, e apesar dos rumores da sua queda precoce, permaneceu amplamente utilizado em toda a indústria. É um protocolo assíncrono ponto-a-ponto, os canais independentes são estabelecidos para comunicações bidirecionais (*full-duplex*). Na sua implementação base utiliza três terminais (transmissor, recetor e GND (*Ground*)) com níveis de sinal de +/- 3V a +/- 15 V onde o nível lógico "1" tem o sinal negativo, e o nível lógico "0" tem o nível positivo (Figura 2.13). A distância máxima para taxas de transmissão baixas é cerca de 15 metros, mas para distâncias mais pequenas, de mais ou menos 1,5 metros (50') podemos atingir taxas de transmissão de 256000 bps (*bits per second*) [15].

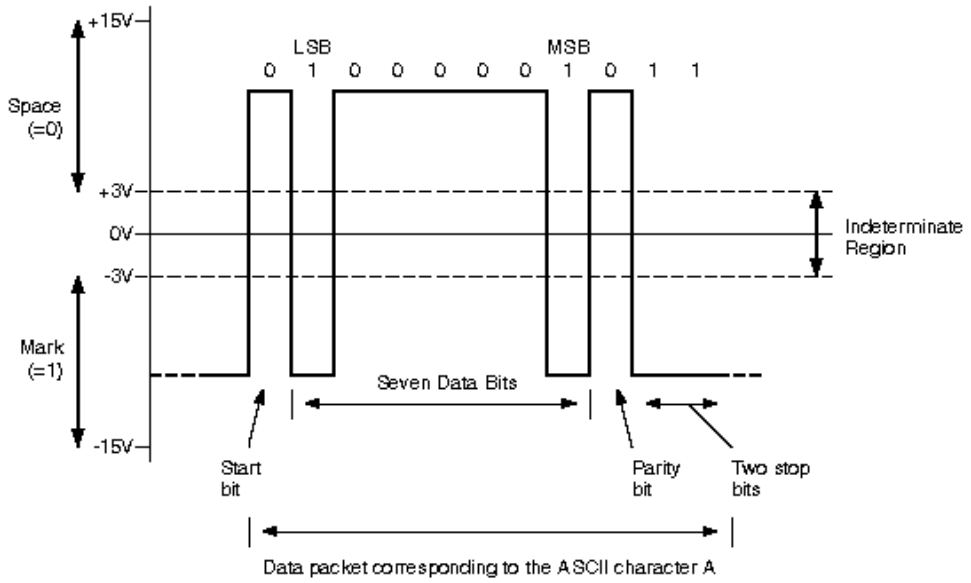


Figura 2.13 – Exemplo de comunicação RS-232 do caracter "A" [15]

2.8.2 RS-485

O RS-485 também é um protocolo assíncrono, suporta redes de comunicação multiponto, pode ter até 32 dispositivos no mesmo barramento, usando a mesma linha diferencial equilibrada em par entraçado, pode ser usado com taxas de transmissão de até 10 Mbps e distâncias até 1200 metros, tendo em conta que para esta distância, só é possível atingir uma taxa de transmissão de até 100 kbps. É constituído por dois terminais diferenciais (Figura 2.14), A (+) e B (-), com nível de sinal entre -7 V e +12 V e geralmente necessita de uma resistência de terminação com um valor de impedância correspondente à impedância característica do barramento, normalmente de 120 Ω (ohm), cuja finalidade é atenuar reflexões que distorcem os dados transmitidos, permitindo assim aumentar a distância e a velocidade de transmissão [16].

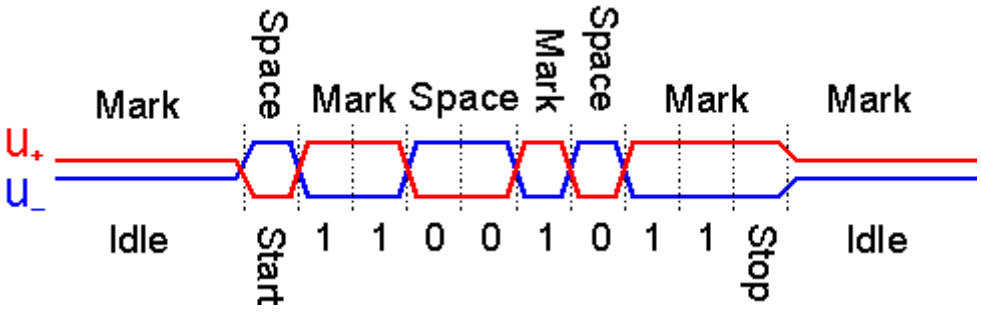


Figura 2.14 – Exemplo de comunicação RS-485 onde o sinal U+ representa o terminal A e o U- representa o terminal B [16].

2.8.3 SPI

O SPI é um protocolo síncrono, o que significa que usa linhas separadas de dados e um *clock*, que mantém as linhas em perfeita sincronia. Este protocolo segue a topologia mestre-escravo onde o mestre é geralmente um microcontrolador e os escravos os restantes periféricos. Para implementar a comunicação SPI, são necessários os seguintes pinos: Serial Clock (SCLK), Master Output Slave Input (MOSI), Master Input Slave Output (MISO) e Slave/Chip Select (SS ou CS) (Figura 2.15).

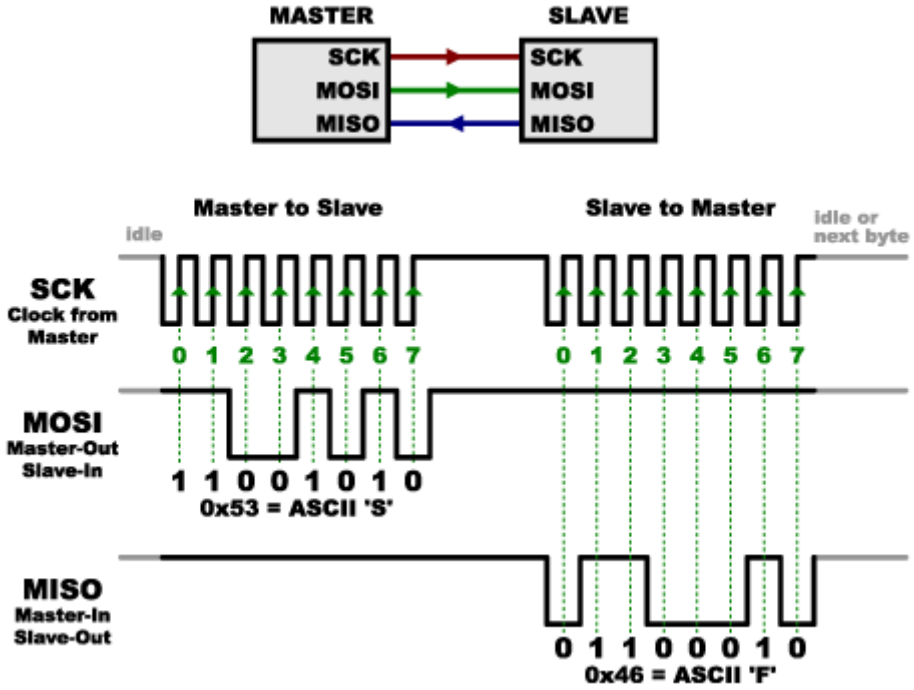


Figura 2.15 – Exemplo de comunicação SPI do caracter "S" e "F" [17].

No SPI, o mestre gera o sinal *clock* (SCLK), quando os dados são enviados do mestre para o escravo, são enviados pela linha de dados MOSI, se o escravo enviar uma resposta de volta para o mestre, é utilizada a linha de dados MISO, e o mestre continuará a enviar um número de ciclos *clock* pré-estabelecidos. O pino SS também é gerado pelo mestre, o que permite ao mestre selecionar o escravo com que vai comunicar. O protocolo SPI permite altas velocidades de transmissão, comunicação *full-duplex* ou *half-duplex*, e destaca-se por ser muito utilizado em módulos como Arduíno, Raspberry Pi e BeagleBone [17].

2.9 Estrutura do sistema desenvolvido

Tendo em conta os aspetos abordados neste capítulo, pode ser enunciado de forma mais concreta, sob a forma de arquitetura e estrutura do sistema desenvolvido, o objetivo do projeto descrito neste Relatório de Estágio. Assim, pretendeu-se desenvolver um sistema de recolha e tratamento de dados adquiridos por placas analógicas e digitais, seguindo a estrutura do diagrama apresentado na Figura 2.16.

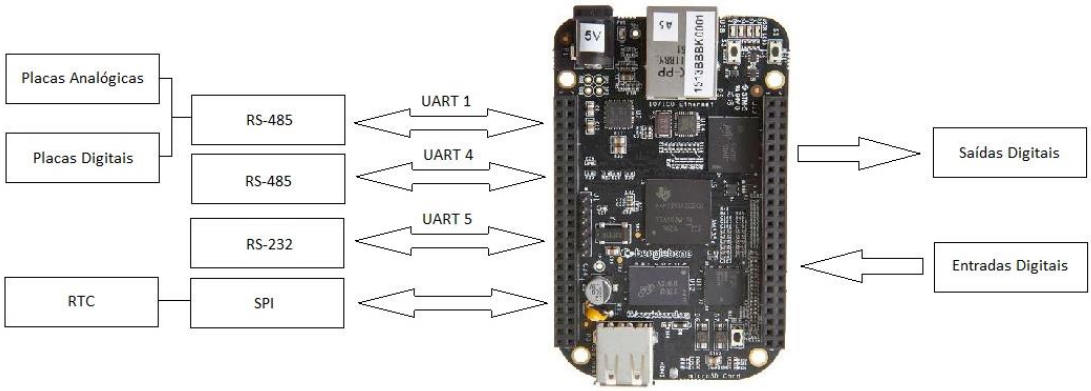


Figura 2.16 – Estrutura geral do sistema de recolha e tratamento de dados que se pretende desenvolver.

De acordo com a Figura 2.16, este sistema é constituído por 2 *interfaces* de comunicação RS-485 para recolha de dados das placas analógicas e digitais, um barramento SPI para comunicação com um RTC (*Real-Time Clock*), RS-232, 4 saídas digitais e 4 entradas digitais.

3 Testes iniciais e configuração da *Gateway*

Neste capítulo são realizados alguns testes com *hardware* externo, onde foi desenvolvido para cada teste um *software* próprio para perceber melhor o funcionamento da BeagleBone Black e as *interfaces* de comunicação que vão ser posteriormente utilizadas no sistema final.

Numa primeira interação, constatamos que a BeagleBone Black já traz consigo um sistema operativo Linux previamente instalado no armazenamento interno, o Debian, assim não foi necessário a instalação de um sistema operativo Linux no cartão de memória, tal como se faz no procedimento de uma primeira interação com um Raspberry Pi.

Depois de conectar a BeagleBone Black por USB a um computador, é necessário instalar os *drivers* de acordo com o sistema operativo utilizado no computador. Depois de instalados os *drivers*, é possível conectar-se à BeagleBone através de acesso remoto, utilizando por exemplo, o Putty (*open-source terminal emulator*), o IP de acesso remoto à BeagleBone por USB é o 192.168.7.2, utilizando o protocolo de rede SSH, onde habitualmente é utilizada a porta 22 (Figura 3.1).

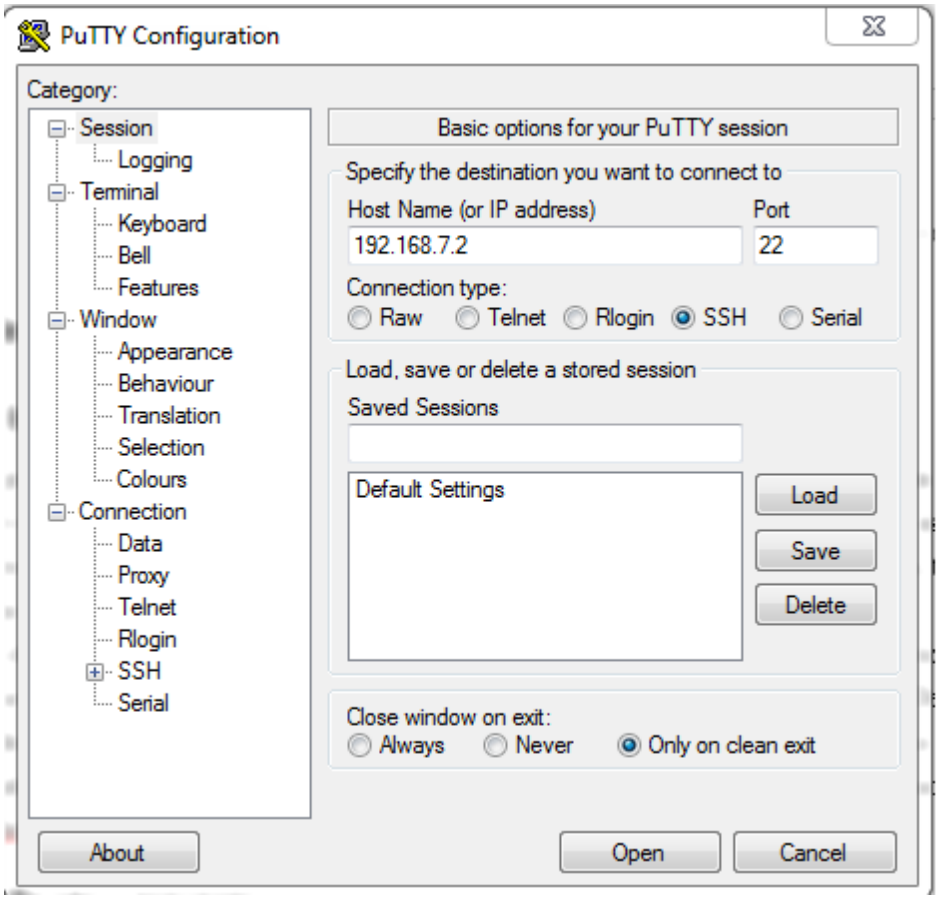


Figura 3.1 – Terminal *Putty* onde se insere o IP do equipamento a que se pretende conectar.

Depois basta iniciar sessão com o utilizador: root (não contém *password*), e a BeagleBone Black está pronta a ser utilizada (Figura 3.2).

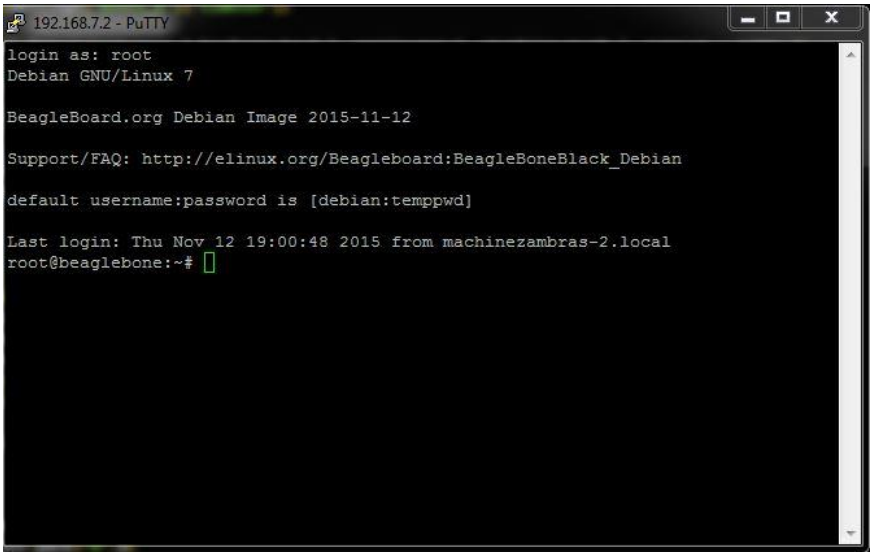


Figura 3.2 – Terminal *Putty* já com sessão iniciada na BeagleBone Black.

3.1 Linguagem de Programação

Para linguagem de programação, foi escolhida a linguagem *Python*, dada a sua facilidade de aprendizagem e conhecimento base da mesma. *Python* é uma linguagem de *script*, isto é, uma linguagem que suporta *scripts*, programas escritos para um sistema de tempo de execução especial que automatiza a execução de tarefas. As linguagens de *script* são frequentemente interpretadas ao invés de compiladas.

Python é bastante utilizado em aplicações web, ou que interagem com aplicações web e administração de sistemas, isto porque, as bibliotecas padrão incluem imensos módulos de protocolos de rede (HTTP, FTP, etc.), processamento de texto e acessos aos serviços do sistema operativo.

Além de ser rápido de desenvolver, e uma linguagem de uso geral, sendo fácil de ler e compreender, torna-se uma linguagem com uma manutenção fácil, o que se torna uma vantagem futura, para desenvolver trabalhos futuros a partir do sistema a desenvolver.

O sistema operativo original da BeagleBone Black, o Debian já contem as bibliotecas padrão *Python* 2.7 e versões anteriores. Para iniciar um *script* de *Python*, basta usar a instrução "python" seguido do nome do *script* com a extensão ".py" na linha de comandos:

python scriptname.py

3.2 Testes iniciais

Esta secção apresenta os vários testes iniciais, para interação com os pinos de entradas e saídas digitais e com as *interfaces* de comunicação digital: SPI e UART. A *interface* de comunicação UART é utilizada para implementar os protocolos de comunicação RS-232

e RS-485 que permitem a interação entre a plataforma de desenvolvimento BeagleBone Black e componentes eletrónicos, bem como circuitos integrados. Para cada teste, foi desenvolvido um *software* próprio para execução do mesmo, e para obtenção dos sinais digitais foi utilizado um BitScope DSO, que possui um *software* para computador, que permite visualizar os sinais.

3.2.1 GPIO (Output e Input)

Como primeiro teste, foi desenvolvido um *software* com o objetivo de piscar um LED, para isso foi necessário a interação com a biblioteca GPIO da BeagleBone Black:

import Adafruit_BBIO.GPIO as GPIO

Depois configurar o pino escolhido como saída digital:

LED = "P9_12"

GPIO.setup(LED, GPIO.OUT)

Seguido de um ciclo *for* onde é mudado o estado no pino de saída entre 1 e 0 (HIGH / LOW) de segundo em segundo, durante 5 vezes:

for i in range(0,5):

GPIO.output(LED, GPIO.HIGH) # LED acende

time.sleep(1) #espera 1 segundo

GPIO.output(LED, GPIO.LOW) # LED apaga

time.sleep(1) #espera 1 segundo

Na Figura 3.3, podemos observar a montagem usada para testar o *software* acima descrito.

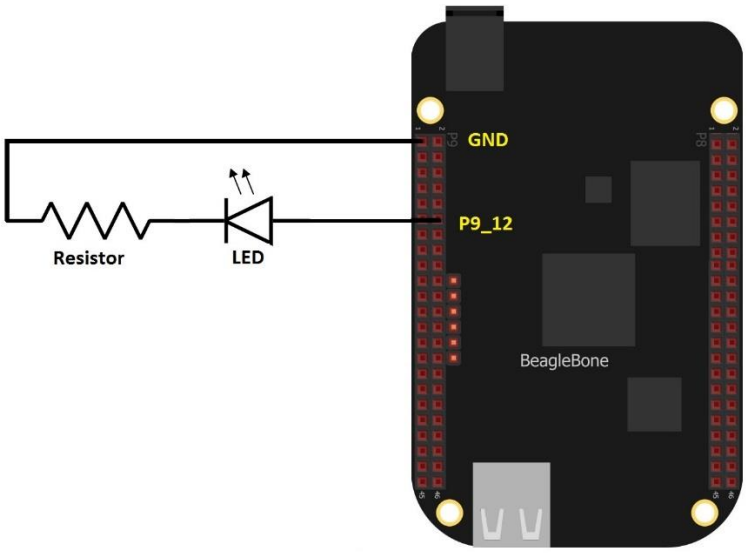


Figura 3.3 – Esquemático de ligação de um LED.

E na Figura 3.4, o LED desligado e ligado, respetivamente:

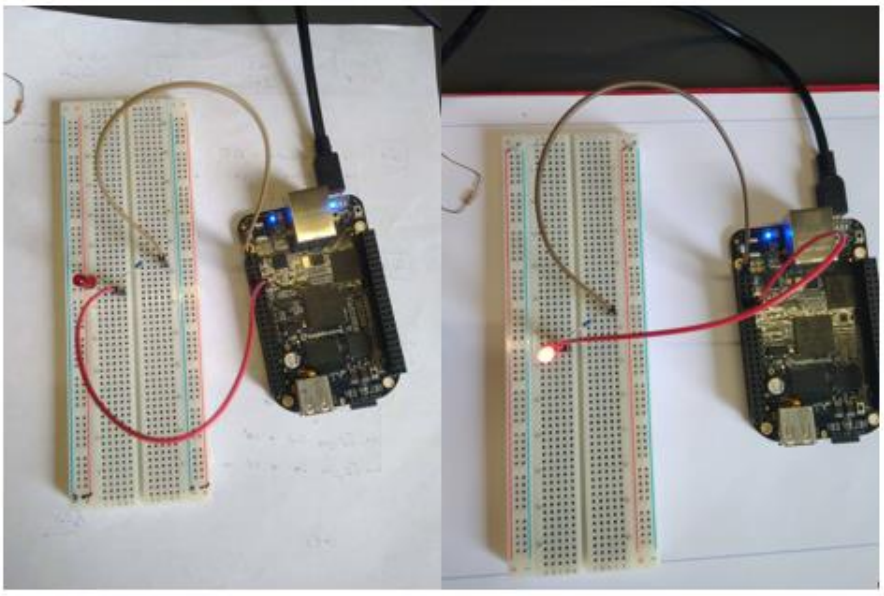


Figura 3.4 – Demonstração de LED apagado e aceso, respetivamente.

No segundo teste, dentro da mesma categoria e utilizando a mesma biblioteca, utilizou-se um interruptor e um LED para verificar a existência de tensão no circuito apresentado (Figura 3.5).

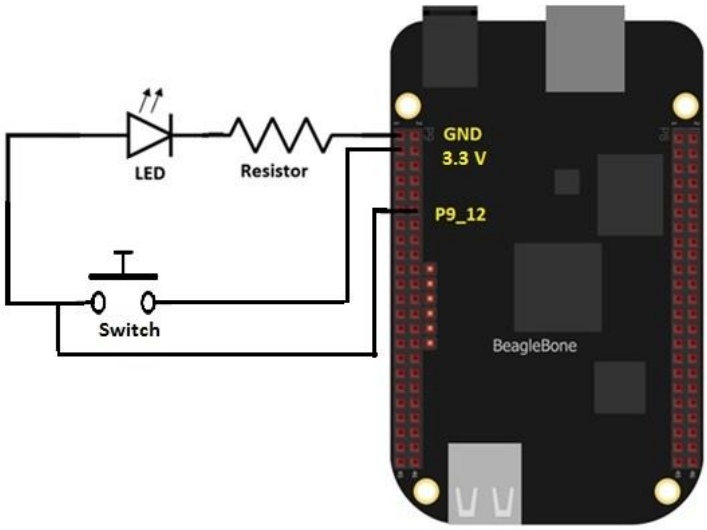


Figura 3.5 – Esquemático de ligação de um LED com interruptor.

Neste teste, ao contrário do exemplo anterior onde o pino digital P9_12 era configurado agora como saída digital, agora é configurado como entrada digital.

interrupt = "P9_12"

GPIO.setup(interrupt, GPIO.IN)

E de seguida, foi feito um ciclo infinito *while* onde lê-mos o estado digital do pino com uma periodicidade de 0.5s (segundos). Se o estado digital do pino mudar de 0 para 1, obtemos uma mensagem a informar (Figura 3.6) que o botão foi pressionado:

while True:

time.sleep(0.5) # delay adicionado, para não ler constantemente o estado da entrada digital

if GPIO.input(interrupt) == 1:

print ' >>> Botao Pressionado <<< '

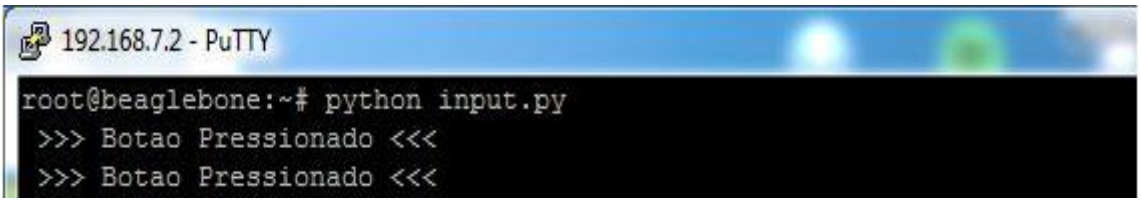


Figura 3.6 – *Output* do programa teste (input.py) de pino de entrada na BeagleBone Black.

E como podemos constatar na Figura 3.7, quando o botão é pressionado o LED acende, e quando não está pressionado, o LED está apagado, respetivamente.

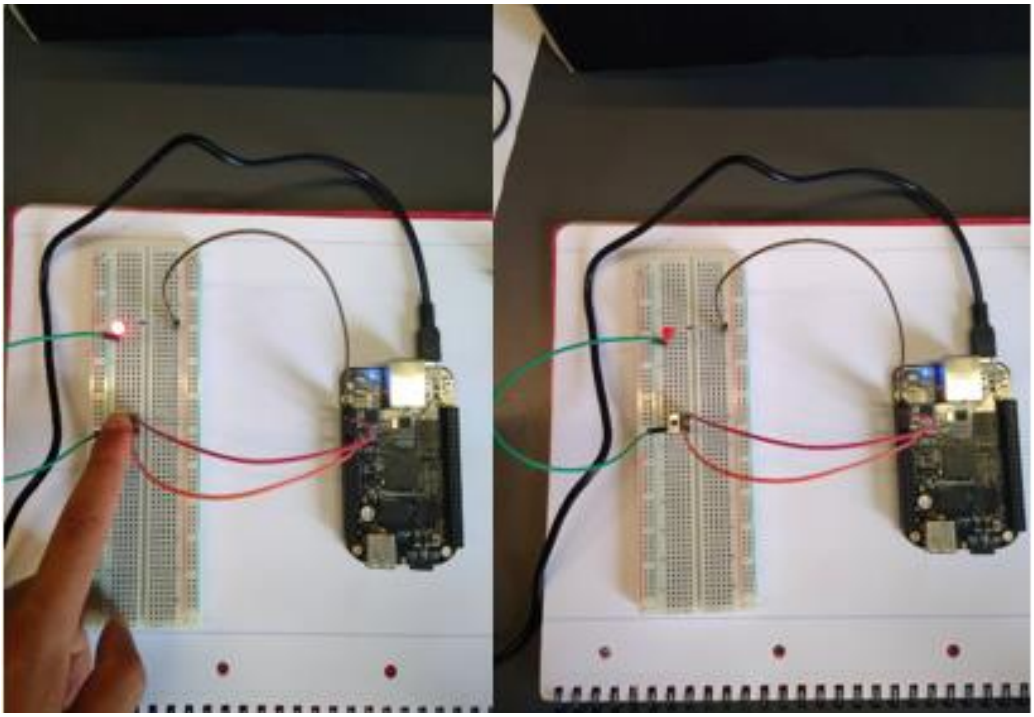


Figura 3.7 – Demonstração do LED apagado e aceso, respetivamente, ao pressionar um botão.

3.2.2 UART

No terceiro teste, foram realizados dois testes bidirecionais entre a UART 1 (TX - P9_24, RX – P9_26) e a UART 4 (TX – P9_13, RX – P9_11). Com este objetivo, ligou-se o TX da UART 1 ao RX da UART 4 (Figura 3.8).

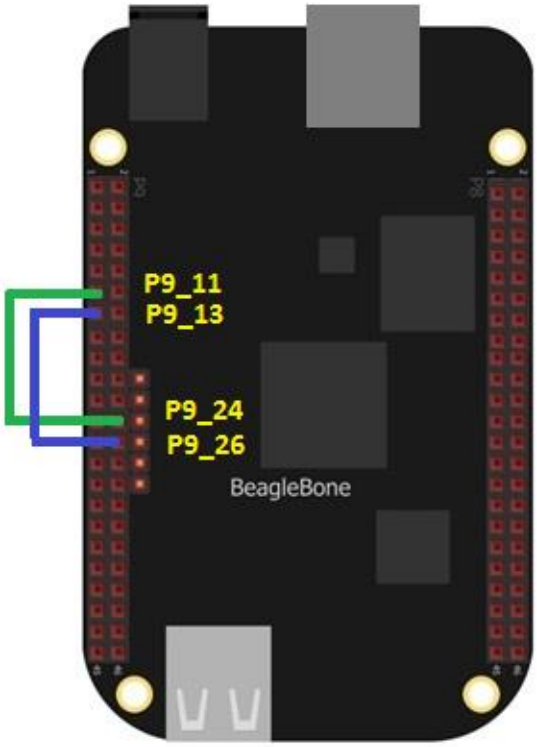


Figura 3.8 – Esquemático de ligação entre a UART 1 (RX – P9_26 e TX – P9_24) e a UART 4 (RX – P9_11 e TX – P9_13).

Para comunicar através das UARTs da BeagleBone Black, basta importar a biblioteca "UART" e "serial":

import Adafruit_BBIO.UART as UART

import serial, time

E de seguida iniciar e configurar a UART, que neste caso foi escolhida a UART 1 ("/dev/ttyO1") e a *baudrate* (115200):

UART.setup("UART1")

uart = serial.Serial('/dev/ttyO1',115200)

Para enviar dados é utilizada a instrução "write":

uart.write(string_enviar)

E para receber dados é utilizada a instrução "read", que utiliza como parâmetro o número de *bytes* a ler (neste caso, a variável "dados_buffer" é um inteiro):

data = uart.read(dados_buffer)

No primeiro exemplo, foi enviada a partir da UART 1 (tx.py) uma *string* de 5 caracteres "abcde", e obtida uma resposta da UART 4 (rx.py) de uma *string* hexadecimal "/xFF", que corresponde ao número decimal 255 (Figura 3.9).

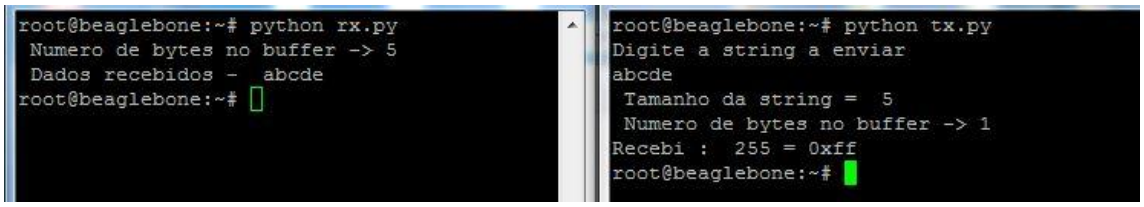


Figura 3.9 – *Output* dos programas rx.py (UART 4) e tx.py (UART 1) do teste 1 na BleagleBone Black.

E como podemos ver na Figura 3.10, cada caracter corresponde a 1 *byte*, fazendo um total de 5 *bytes*, e a *string* hexadecimal "/xFF" corresponde apenas a 1 *byte*.

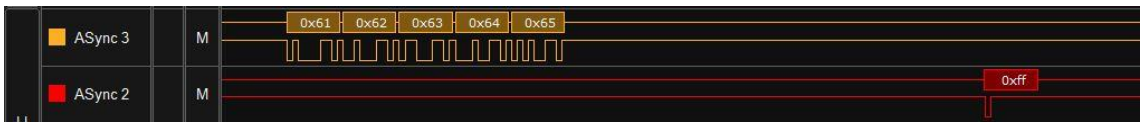


Figura 3.10 – A laranja a transferência da *string* "abcde" da UART 1 para a UART 4, e em resposta a vermelho o *byte* "/xFF".

No segundo exemplo, foi enviada a *string* "255" e depois é recebida, novamente, a resposta de uma *string* hexadecimal "/xFF" (Figura 3.11).

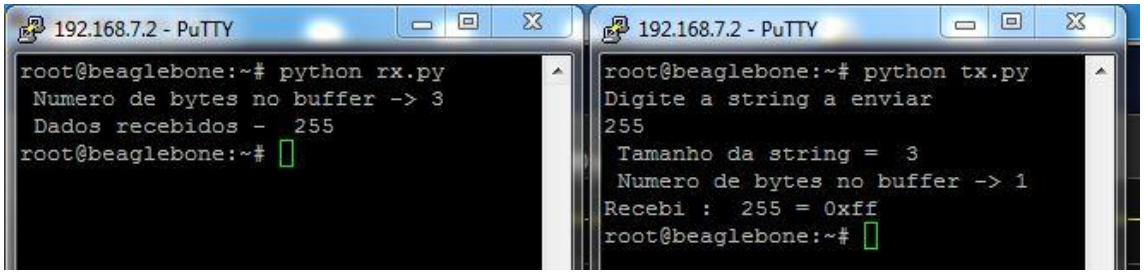


Figura 3.11 – *Output* dos programas rx.py (UART 4) e tx.py (UART 1) do teste 2 na BeagleBone Black.

E como podemos ver na Figura 3.12, na *string* "255", cada número corresponde a 1 *byte*, fazendo um total de 3 *bytes*, é enviado em binário o correspondente caracter na tabela ASCII, ao contrário da *string* hexadecimal, onde o decimal 255 ocupa apenas 1 *byte*.

Figura 3.12 – A laranja a transferência da *string* "255" da UART 1 para a UART 4, e em resposta a vermelho o *byte* "/xFF".

3.2.3 RS-232

No quarto teste, foi utilizado pela primeira vez um circuito integrado, um conversor RS-232, para converter a linha de dados da UART da BeagleBone Black em RS-232 e posteriormente, um conversor RS-232 para USB para visualizar os dados no computador, através de um terminal, neste caso foi utilizado o *Hterm* (Figura 3.13).

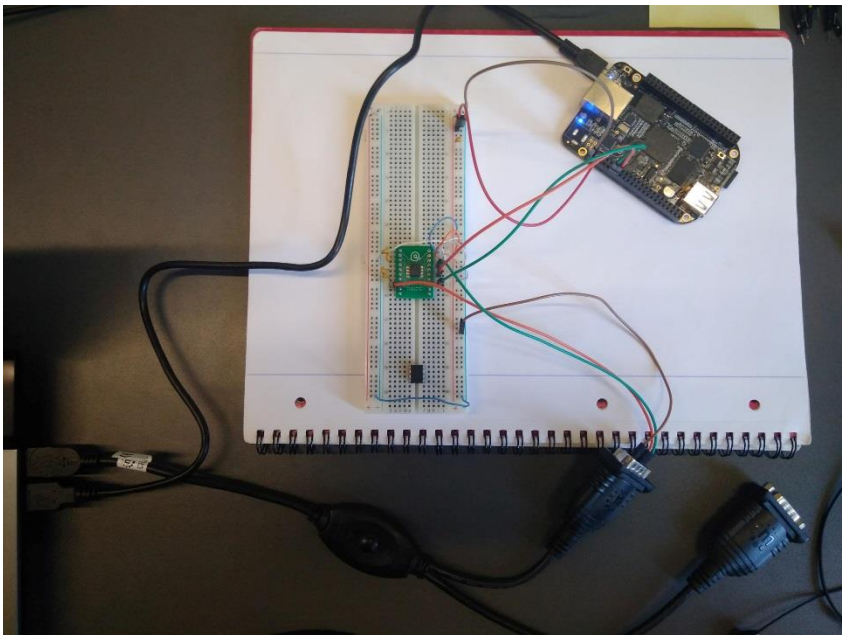


Figura 3.13 – Circuito de ligação da BeagleBone Black (UART) a um computador, com *transceiver* RS-232, conversor RS-232 para USB.

Para a realização do teste foi utilizado um *transceiver* RS-232, cuja sua configuração elétrica dispõe de 3.3V de alimentação e 5 condensadores de 0.1uF (Figura 3.14).

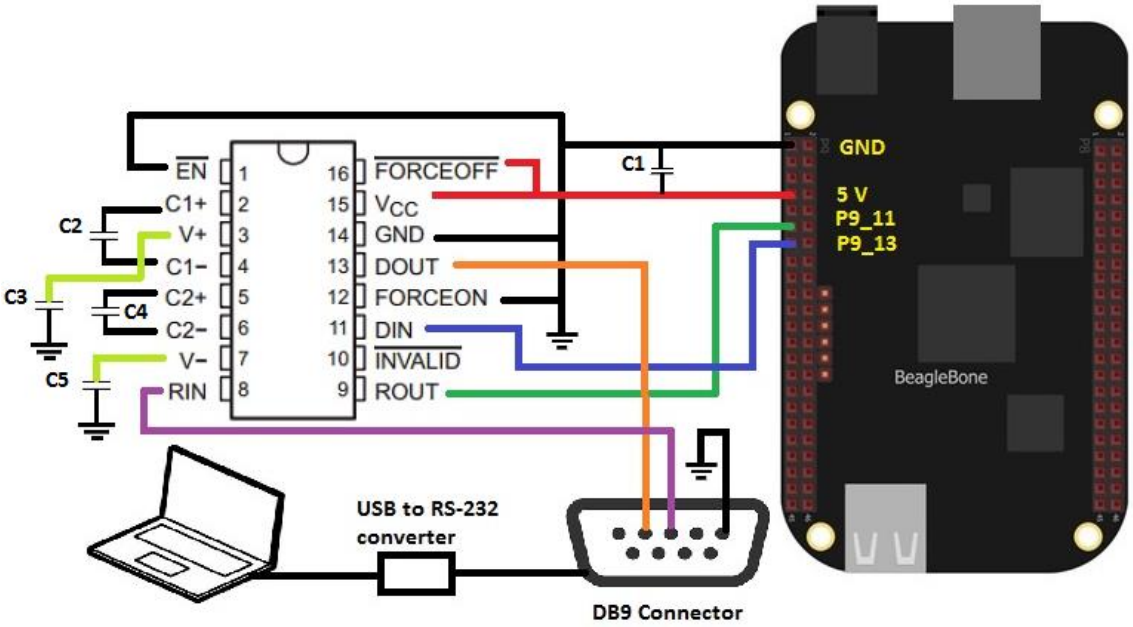


Figura 3.14 – Esquemático de ligação entre BeagleBone Black e um computador através dos conversores RS-232 e RS-232 para USB.

Para este teste, foi desenvolvido um programa que está sempre a ler apenas um *byte*, e depois de ler o valor recebido, envia como resposta uma *string* hexadecimal desde o número 1 até ao número recebido. Se for enviado o número 4 do terminal Hterm, a BeagleBone Black envia como resposta "/x01/x02/x03/x04", que no terminal é lido como valores decimais 1,2,3 e 4 (Figura 3.15).

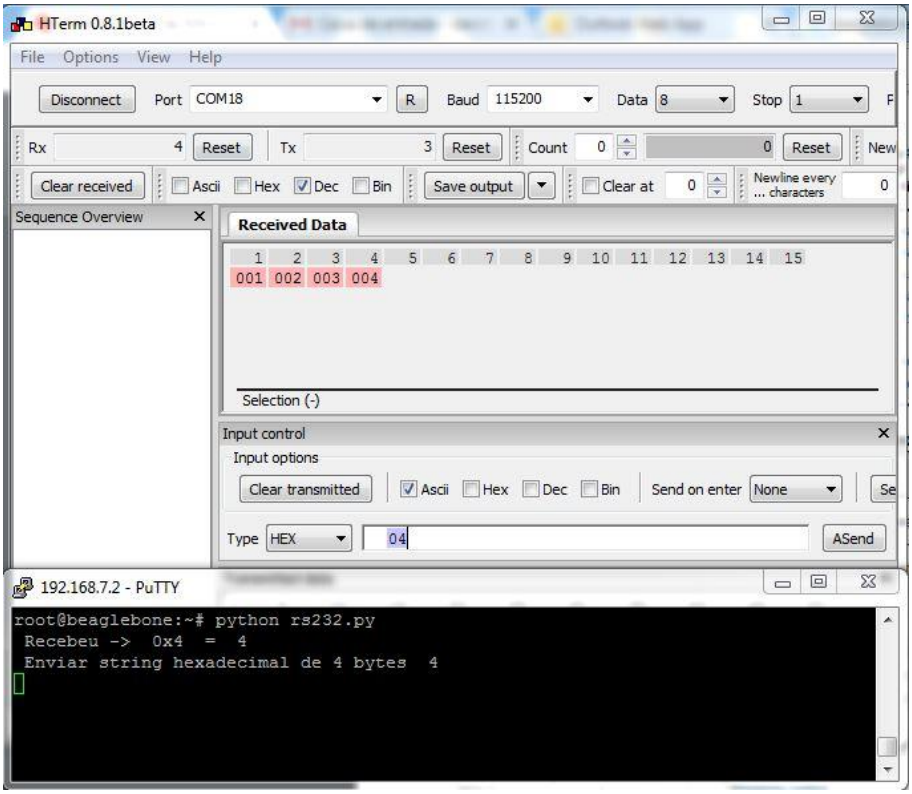


Figura 3.15 – *Output* do terminal *Hterm* (computador) e terminal *Putty* (BeagleBone Black) na comunicação por RS-232.

Na Figura 3.16, podemos ver os dados transmitidos. A laranja e vermelho, o RX e TX da linha de dados UART da BeagleBone, respetivamente, e a castanho e branco, o RX e TX da linha de dados do conversor RS-232, respetivamente, e como podemos constatar, ao comparar os *bits* das linhas de dados UART e RS-232, respetivamente, têm lógica inversa.

Figura 3.16 – Linhas de dados RX (laranja) e TX (vermelho) da UART da BeagleBone Black e RX (castanho) e TX (branco) da linha de dados do conversor RS-232.

3.2.4 RS-485

No quinto teste, para testar o protocolo RS-485, foram utilizados dois conversores RS-485 e duas BeagleBone Black (Figura 3.17).

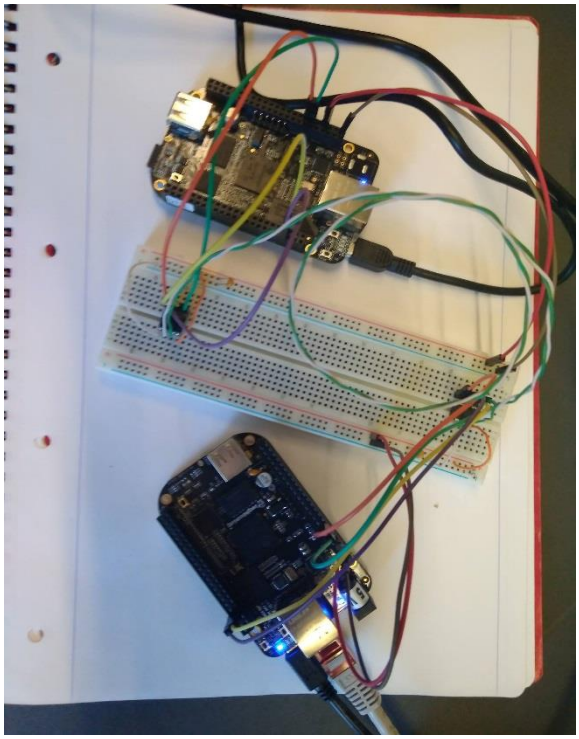


Figura 3.17 – Circuito de ligação entre duas BeagleBone Black através de comunicação RS-485

Para além dos dois *transceivers* RS-485, foram utilizados dois condensadores (C1,C2) de 0.1 µF, para a alimentação (Figura 3.18).

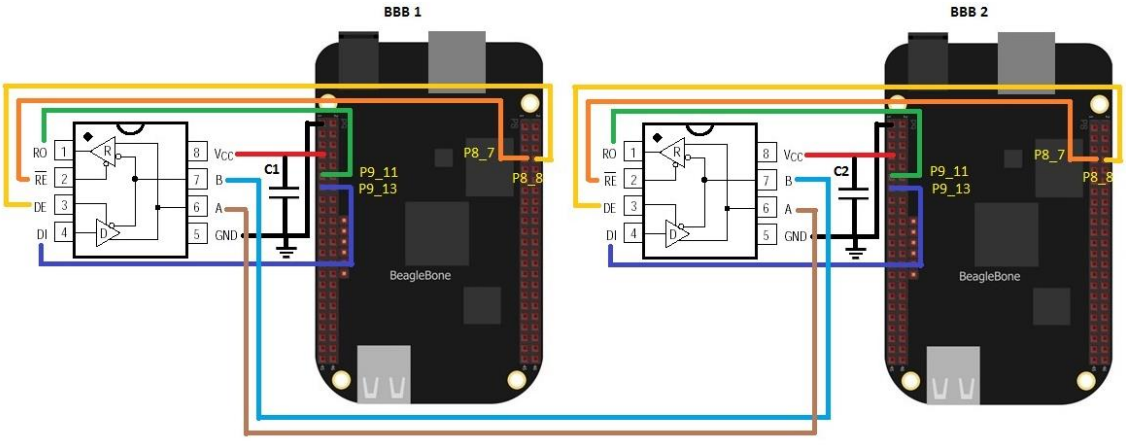


Figura 3.18 – Esquemático de ligações entre a UART 4 da BeagleBone Black e o conversor RS-485.

Ao contrário do conversor RS-232, o conversor RS-485 contém dois pinos de controlo de acesso ao barramento RS-485, o RE e o DE, que se pode ligar a dois pinos GPIO, onde mudamos de estado lógico, de acordo com o modo de operação desejado para o conversor

RS-485. No caso deste conversor, para enviar dados, colocamos o estado lógico dos pinos RE e DE a "1", e para receber dados a "0"

def rs_send():

#RE e DE a 1 para enviar

GPIO.output(RE, GPIO.HIGH)

GPIO.output(DE, GPIO.HIGH)

def rs_receive():

#RE e DE a 0 para receber

GPIO.output(RE, GPIO.LOW)

GPIO.output(DE, GPIO.LOW)

O *software* desenvolvido para testar o protocolo RS-485 é idêntico ao *software* usado no protocolo RS-232, onde a BeagleBone 2 (BBB 2) envia um número decimal entre a gama de 0 e 255, para a BeagleBone 1 (BBB 1), e esta envia como resposta os valores desde 1 até ao valor recebido (Figura 3.19).

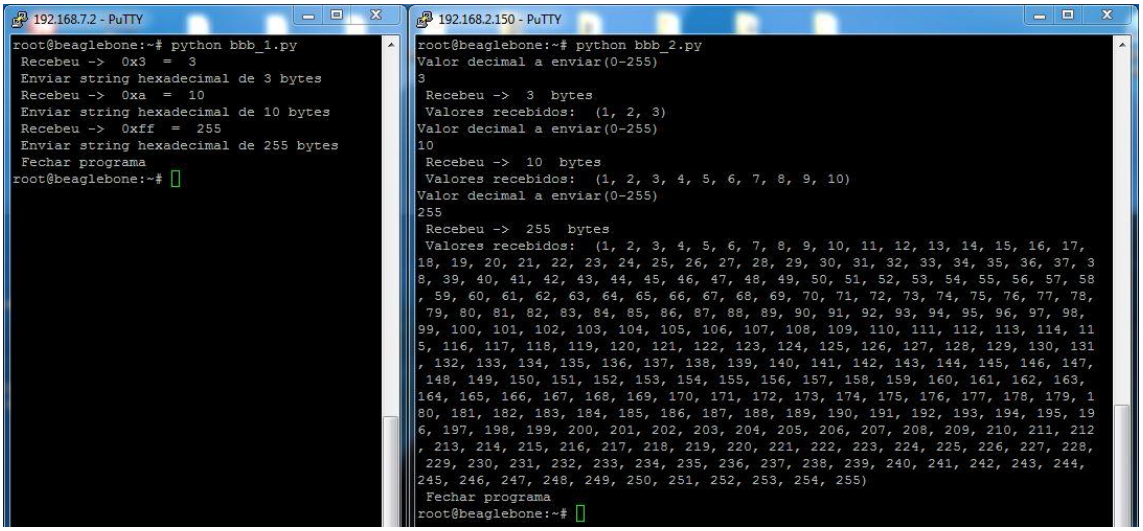


Figura 3.19 – *Output* da comunicação RS-485 entre a BeagleBone Black 1 (bbb_1.py) e BeagleBone Black 2 (bbb_2.py)

Na Figura 3.20 podemos observar as linhas diferenciais do RS-485: A(+) – vermelho e B(-) – Castanho, e as linhas de dados: a laranja o RX da BeagleBone 1, e a branco o TX da BeagleBone 2, durante a transferência do valor 3. E como podemos constatar, quando a linha diferencial A(+) é superior à linha diferencial B(-), é obtido como resultado na linha de dados o valor lógico "1", e quando a linha diferencial A(+) é inferior à linha diferencial B(-), é obtido como resultado o valor lógico "0".

Figura 3.20 – Linhas de dados da UART 4 na BeagleBone Black 1, RX (laranja) e TX (branco) e as linhas diferencias RS-485, A(+) (vermelho) e B(-) (castanho).

3.2.5 SPI

Num último teste, foi utilizada uma memória SRAM para validar o funcionamento do protocolo SPI. Para testar a comunicação com a memória foram realizados vários testes, como ler e escrever em dois modos distintos, modo *byte* (escrever/ler apenas um *byte*) e modo sequencial (escrever/ler vários *bytes* em sequência) (Figura 3.21).

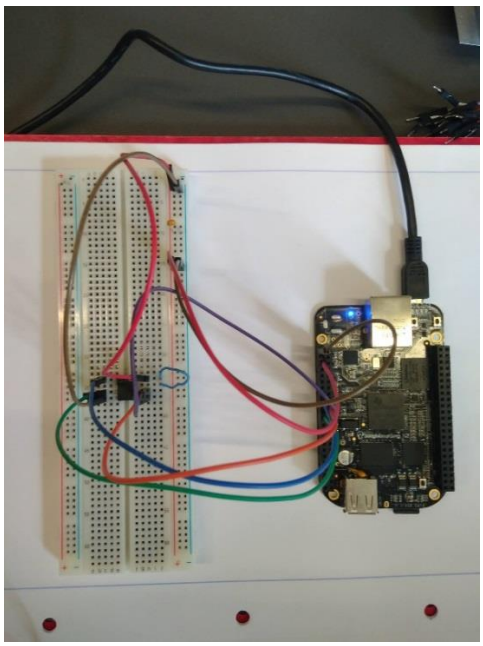


Figura 3.21 – Circuito de ligação entre a memória SRAM e a BeagleBone Black através da *interface* SPI.

Como componentes, para além da memória apenas foi adicionado um condensador (C1) na alimentação. Nas ligações de comunicação do SPI temos o pino CS (Chip Select) que liga ao CS da BeagleBone (P9_17 – Azul), o SO (Slave Out) que liga ao MISO (Master Input Slave Output) (P9_21 – Verde), o SI (Slave In) que liga ao MOSI (Master Output Slave Input) (P9_18 – Laranja) e o SCK (Serial Clock) que liga ao SCLK (P9_22 – Roxo) (Figura 3.22).

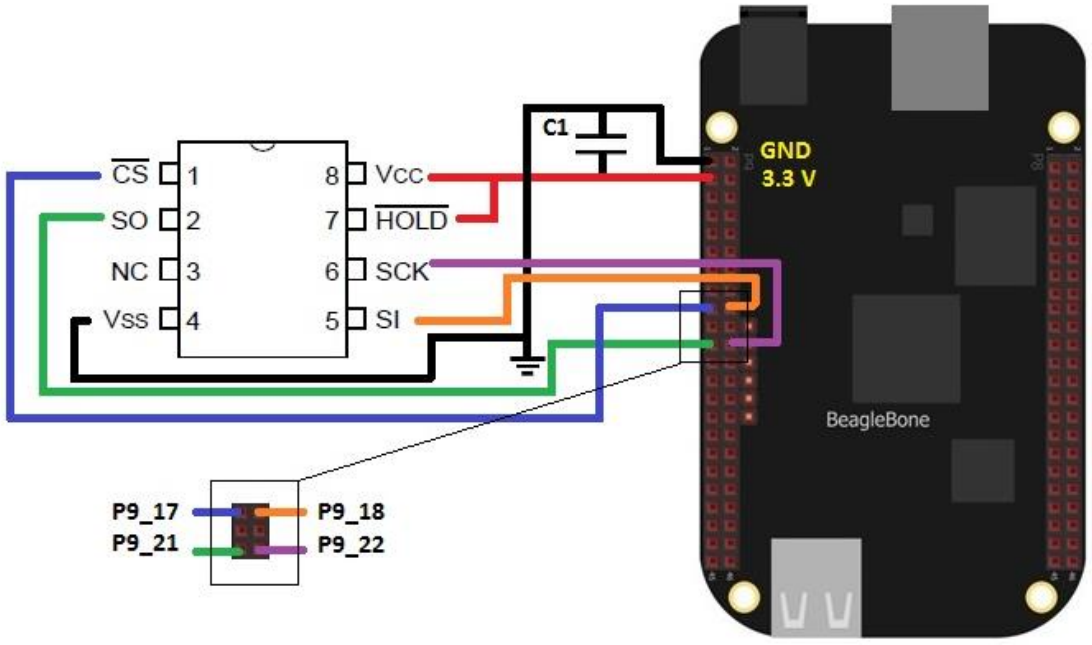


Figura 3.22 – Esquemático de ligação entre a memória SRAM e a BeagleBone Black através do protocolo de comunicação SPI.

Para comunicar por SPI, usámos a biblioteca *spidev*:

import spidev

De seguida, é necessário abrir e configurar o SPI:

spi = spidev.SpiDev()

spi.open(1, 0) # SPI 1 (bus,device)

spi.max_speed_hz = 1000000 # SCLK 1 MHz

spi.mode = 0 #modo 0 - clock 0 e fase 0

spi.lsbfirst = False #msb primeiro

spi.threewire = False #modo 4 fios

spi.bits_per_word = 8 # 8 bits

spi.cshigh = False # CS 0

Cada componente que funcione por comunicação SPI, tem um protocolo de comunicação específico, neste caso, para comunicar com a memória o SCLK tem uma frequência máxima de 4 MHz e o CS tem de estar no estado lógico "0" durante a comunicação. A sequência de *bytes* para ler/escrever é composta inicialmente pela instrução de 8 *bits* (1 *byte*) que indica se é modo de leitura ou escrita, seguido de um endereço de 16 *bits* (2 *bytes*) e por fim um *byte* ou mais de dados, dependendo se está em modo *byte* ou sequencial (Figura 3.23).

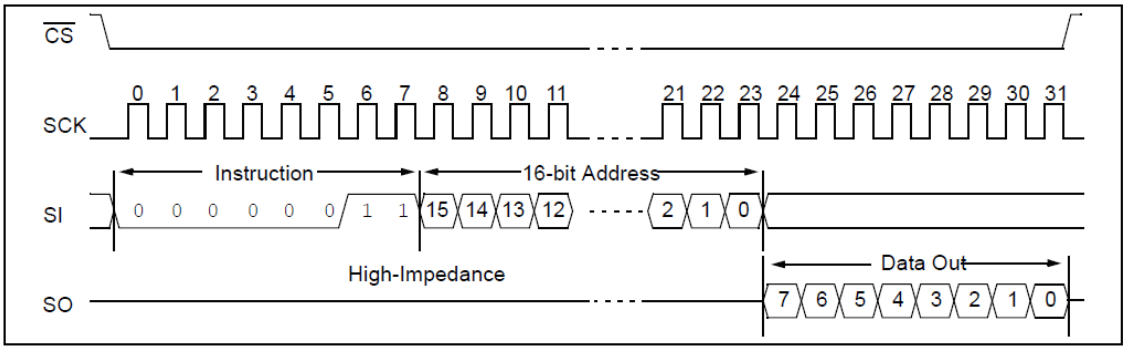


Figura 3.23 – Sequência temporal para a leitura de um *byte* na memória SRAM.

Para este teste, foi feito um programa mais complexo do que os anteriores onde inicialmente lê o *status* da memória, onde se verifica se está em modo *byte* ou sequencial, e se queremos mudar o *status*. Quando se pretende escrever na memória, é necessário indicar o endereço em que se deseja ler e de seguida executar a leitura dessa posição na memória. Quando se pretende escrever na memória, é necessário indicar o endereço em que se deseja escrever e de seguida escrever nessa posição de memória (Figura 3.24).

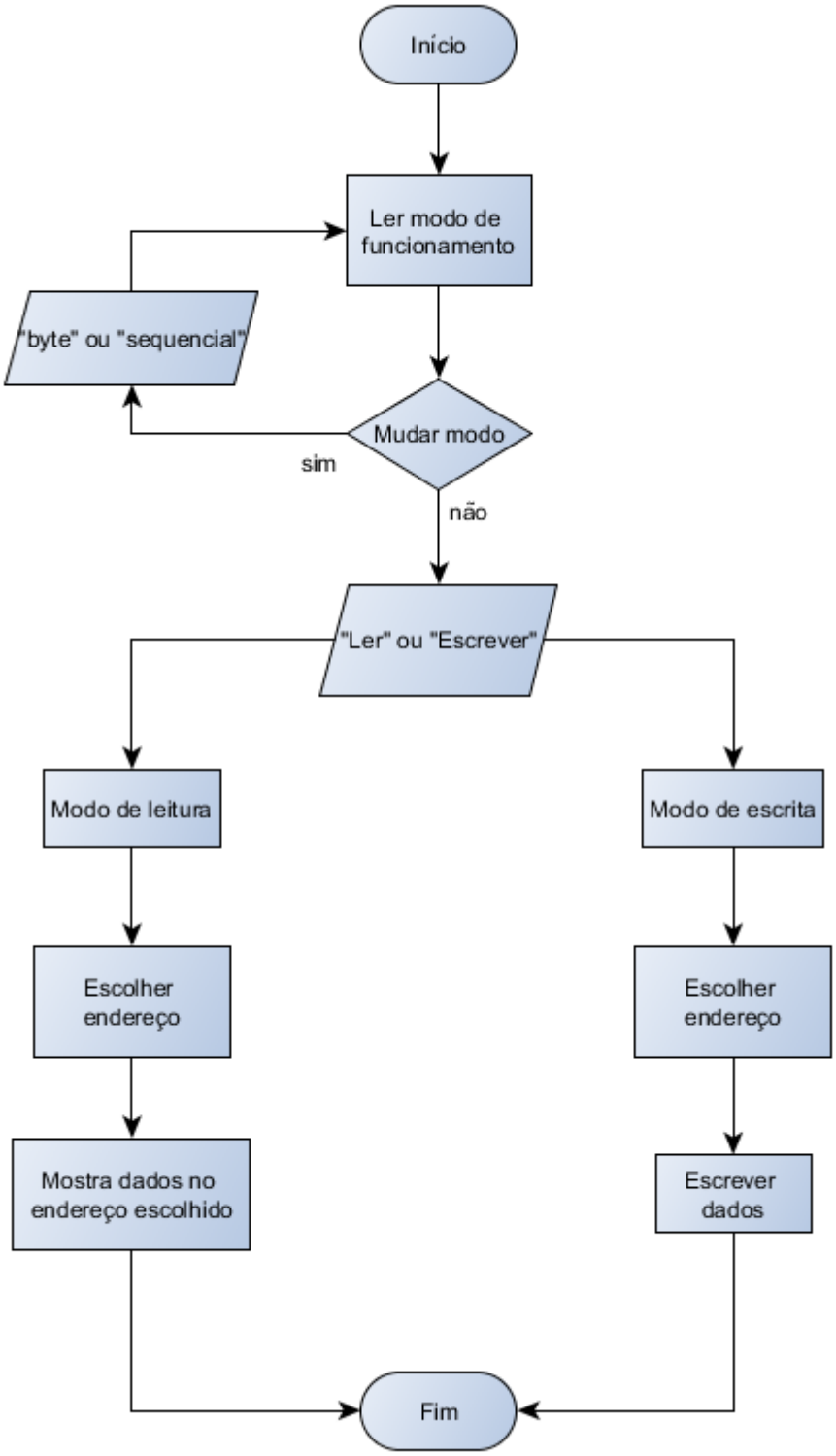


Figura 3.24 – Fluxograma Geral usado no programa memory_debug.py, para a *interface* SPI com a memória SRAM.

Na Figura 3.25 temos um exemplo de escrita e leitura em modo *byte*, do caracter "e" no endereço 1000. Na leitura do endereço 1000, o valor decimal lido é o 101, que na tabela ASCII corresponde ao caracter "e".

```
root@beaglebone:~# python memory_debug.py
 Reading Status
 Status : Modo Byte
 Mudar de modo?  (sim/nao)
nao
Ler ou Escrever? (modo sequencial->5 bytes)
Escrever
Endereco? (16 bits -> 0-65535)
1000
O que escrever?(Em modo sequencial, separar por espacos os caracteres)
e
root@beaglebone:~# python memory_debug.py
 Reading Status
 Status : Modo Byte
 Mudar de modo?  (sim/nao)
nao
Ler ou Escrever? (modo sequencial->5 bytes)
Ler
Endereco? (16 bits -> 0-65535)
1000
 Valor na posicao 1000 -> 101 = e
root@beaglebone:~#
```

Figura 3.25 – *Output* do programa memory_debug.py na escrita e leitura de dados, no modo *byte*.

Na Figura 3.26 podemos observar o protocolo de comunicação SPI, da leitura de um *byte*. São necessários 4 *bytes* de SCLK (1 de instrução, 2 de endereço e 1 de dados) e o CS no estado lógico "0" durante a comunicação. Como podemos ver, a instrução para leitura é 0x03 (0b00000011), depois os *bytes* hexadecimais 0x03 (0b00000011) e 0xE8 (0b11101000), que correspondem aos números decimais 3 e 232, respetivamente, compõem o endereço decimal 1000. Para agrupar 2 *bytes* em uma *word* de 16 *bits*, faz-se a transformação: 256*MSB + LSB, neste caso, 256*3+232 = 1000. No último *byte* do MOSI, podemos enviar qualquer *byte*, pois é ignorado pela memória, e é neste *byte* de SCLK onde obtemos a resposta do MISO no modo leitura, neste caso 0xA5 (0b10100110) porque o *software* BitScope inverte o sentido de leitura do MSB e LSB do MISO, se invertermos o binário 0b10100110 (0xA5), obtemos 0b01100101 (0x65), que corresponde ao número decimal 65, e na tabela ASCII corresponde ao caracter "e".

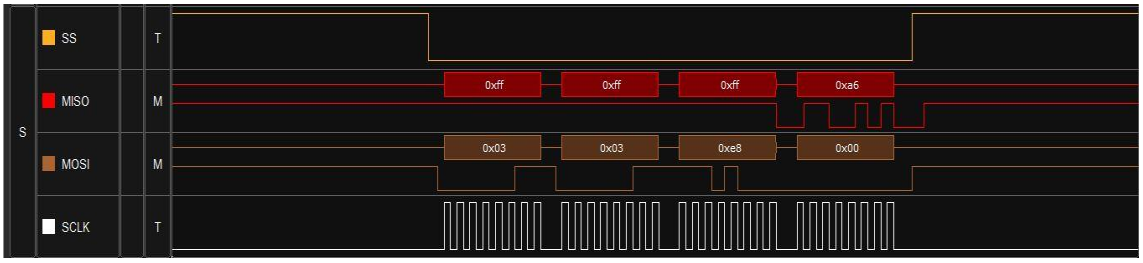


Figura 3.26 – Linhas de dados da *interface* SPI, CS (laranja), MISO (vermelho), MOSI (castanho) e SCLK (branco) durante a leitura de dados em modo *byte*.

Na Figura 3.27 temos um exemplo de escrita e leitura em modo sequencial de 5 *bytes* no endereço 20000, neste caso os carateres escritos foram "a","e","i","o" e "u".

```
root@beaglebone:~# python memory_debug.py
 Reading Status
 Status : Modo Byte
 Mudar de modo?  (sim/nao)
sim
Qual modo?  (byte/sequencial)
sequencial
 Estamos agora em modo sequencial
Ler ou Escrever? (modo sequencial->5 bytes)
Escrever
Endereco? (16 bits -> 0-65535)
20000
O que escrever?(Em modo sequencial, separar por espacos os caracteres)
a e i o u
root@beaglebone:~# python memory_debug.py
 Reading Status
 Status : Modo Sequencial
 Mudar de modo?  (sim/nao)
nao
Ler ou Escrever? (modo sequencial->5 bytes)
Ler
Endereco? (16 bits -> 0-65535)
20000
 Valor na posicao 20000 -> [97, 101, 105, 111, 117] = ['a', 'e', 'i', 'o', 'u']
root@beaglebone:~#
```

Figura 3.27 – *Output* do programa memory_debug.py na escrita e leitura de dados, no modo *byte*.

Na Figura 3.28 podemos observar o protocolo de comunicação SPI, para a escrita na memória em modo sequencial. Onde nos primeiros 8 pulsos do SCLK indicamos a instrução de escrita (0b00000010), depois nos dois próximos *bytes* o endereço de escrita para o primeiro *byte* de dados, neste caso os *bytes* 0x4E e 0x20, que compõem o endereço 20000 (256*78+32), e depois os 5 *bytes* de dados: 0x61, 0x65, 0x69, 0x6F e 0x75, que na tabela ASCII correspondem aos carateres “a”,”e”,”i”,”o” e “u”, respetivamente.

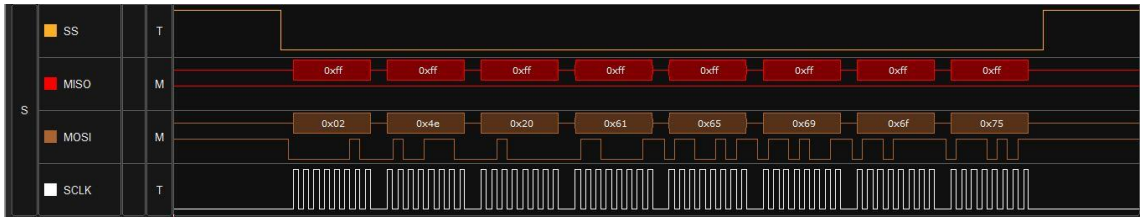


Figura 3.28 – Linhas de dados da *interface* SPI, CS (laranja), MISO (vermelho), MOSI (castanho) e SCLK (branco) durante a escrita de dados em modo sequencial.

3.3 *Shield*

Para ajudar no desenvolvimento do sistema pretendido, a Enging decidiu desenvolver uma placa de circuitos com vários *transceivers* para os diversos *interfaces* de comunicação e conetores. Esta placa mais conhecida por *shield*, encaixa na parte superior da BeagleBone Black através dos *headers* (Figura 3.29).

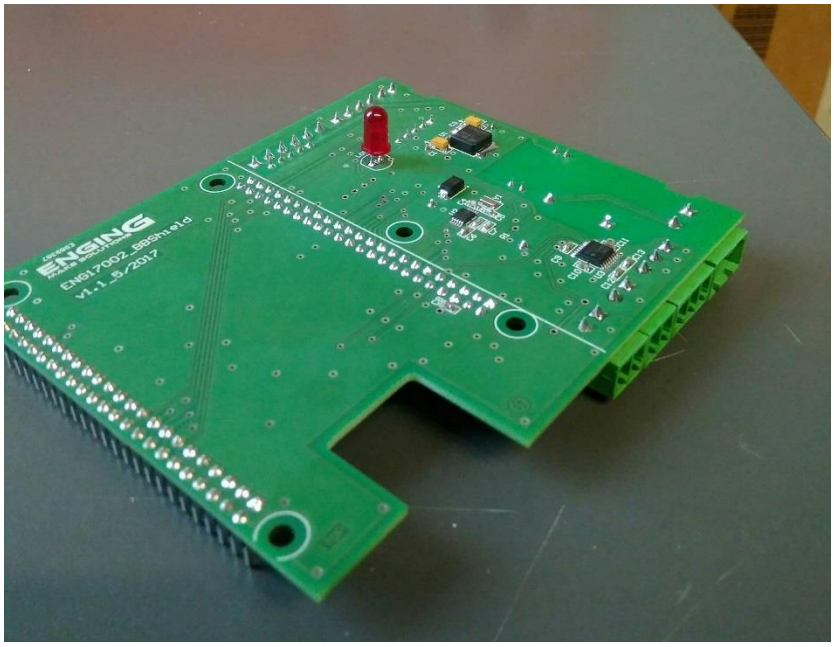


Figura 3.29 - *Shield*

Como podemos ver na Figura 3.30, a *shield* possui dois *transceivers* RS-485, um conversor RS-232, quatro entradas digitais de +24V e quatro saídas digitais de +24V. Como as entradas e saídas digitais da BeagleBone Black são de 3,3V, é feita a conversão de +24V para 3,3V e são utilizados *optocouplers* para o isolamento. Cada destes componentes possui os seus respetivos conetores. Para além dos componentes antes descritos a *shield* possui ainda um RTC, um conetor para a pilha (alimentação do RTC para gravação do *timestamp* em caso de falhas de energia) e um LED vermelho. A alimentação da *shield* é de +24V.

Figura 3.30 – Caixa da *Gateway* (*Shield* + BeagleBone Black)

3.4 Configurar *Gateway*

Para preparar uma BeagleBone Black pela primeira vez para o sistema pretendido, ou seja, para ficar a funcionar com a *gateway* desejada, é necessário algumas alterações no seu *software* e *firmware*, como habilitar e desabilitar algumas funcionalidades, e adicionar o *software* para as funcionalidades desejadas. Tal como mencionado anteriormente, a BeagleBone Black já traz consigo um SO (Sistema Operativo) instalado na sua memória interna, neste caso, a imagem Debian 2015-11-12 (Debian 4.6.3-14) que contém a versão Linux 3.8.13-bone79, e como decidido com a empresa, o SO de fábrica mantém-se na memória interna, e o cartão de memória SD *card* irá ser unicamente para alocação de ficheiros texto com os dados recolhidos.

Ao alimentar a BeagleBone Black a um computador pelo cabo mini-USB, é possível aceder a várias pastas dentro da BeagleBone Black através do computador, uma delas contém os drivers necessários, para aceder via *ssh* sem necessidade de uma conexão Ethernet na BeagleBone Black. Ao instalar os *drivers*, é criada uma nova placa de rede no computador com o IP: 192.168.7.1, e assim já é possível aceder por acesso remoto à BeagleBone Black com um terminal de emulação (Ex: Putty) através do IP: 192.168.7.2 na porta 22 (porta pré-definida de ssh) (Figura 3.1).

Depois de iniciar a ligação através do terminal de emulação Putty, é necessário iniciar sessão da BeagleBone Black com o utilizador "root" e não contem palavra-passe (Figura 3.2).

Depois da BeagleBone Black iniciar sessão, é possível realizar as modificações necessárias para o bom funcionamento da *gateway* desejada. Inicialmente é realizado um *update* e *upgrade* às várias bibliotecas do SO, através dos seguintes comandos:

sudo apt-get update

sudo apt-get upgrade

Com as bibliotecas atualizadas, procedemos à alteração da velocidade *clock* do processador para o seu máximo (1GHz) no ficheiro "cpufrequtils":

nano /etc/init.d/cpufreutils

Dentro do ficheiro "cpufrequtils" o modo de funcionamento do processador é alterado para "*performance*" e a velocidade máxima para "1000" (MHz):

```
ENABLE="true"
GOVERNOR="performance"
MAX_SPEED="1000"
```

Depois de alterada a frequência do processador, é necessário reiniciar a BeagleBone Black, e para verificar a alteração efetuada, usamos o seguinte comando:

cpufreq-info

Tal como demonstrado na Figura 3.31, a velocidade do processador encontra-se nos 1000 MHz.

```
current CPU frequency is 1000 MHz (asserted by call to hardware).
cpufreq stats: 300 MHz:0.00%, 600 MHz:0.00%, 720 MHz:0.00%, 800 MHz:0.00%, 1000 MHz:100.00%
```

Figura 3.31 – Frequência do CPU a 1000 MHz

De seguida é necessária a instalação de uma biblioteca para a *interface* SPI na linguagem Python, a biblioteca Spidev, onde primeiro é criada uma diretoria:

mkdir python-spi

Mudar para a diretoria criada:

cd python-spi

E realizar o *download* dos seguintes módulos:

wget https://raw.github.com/doceme/py-spidev/master/setup.py

wget https://raw.github.com/doceme/py-spidev/master/spidev module.c

E por fim, a instalação da biblioteca:

sudo python setup.py install

Depois de instalada a biblioteca Spidev, é desativada a *interface* HDMI e AUDIO, e habilitadas as seguintes *interfaces*: UART 1, UART 4, UART 5 e SPI 0.

Para visualizar quais os pinos/*interfaces* habilitadas na BeagleBone Black, basta excutar o seguinte comando:

cat /sys/kernel/pinctrl/44e10800.pinmux/pingroups

Para desabilitar as funcionalidades HDMI e AUDIO e habilitar as UART 1,4 e 5 e SPI 0 da BeagleBone Black, é necessário aceder ao ficheiro "uEnv.txt", através do comando:

nano /boot/uboot/uEnv.txt

E adicionar as respetivas linhas:

```
capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
capemgr.enable_partno=BB-UART5, BB-UART4, BB-UART1, BB-SPIDEV0
```

Depois de desabilitar e habilitar as *interfaces* desejadas, é sucede-se a inserção do cartão de memória micro SD a funcionar como memória externa.

Começando por formatar o cartão de memória através do *software* SD Card Formatter (Windows *software*), tal como exemplificado na Figura 3.32, em modo "Quick Format" no tipo FAT32.

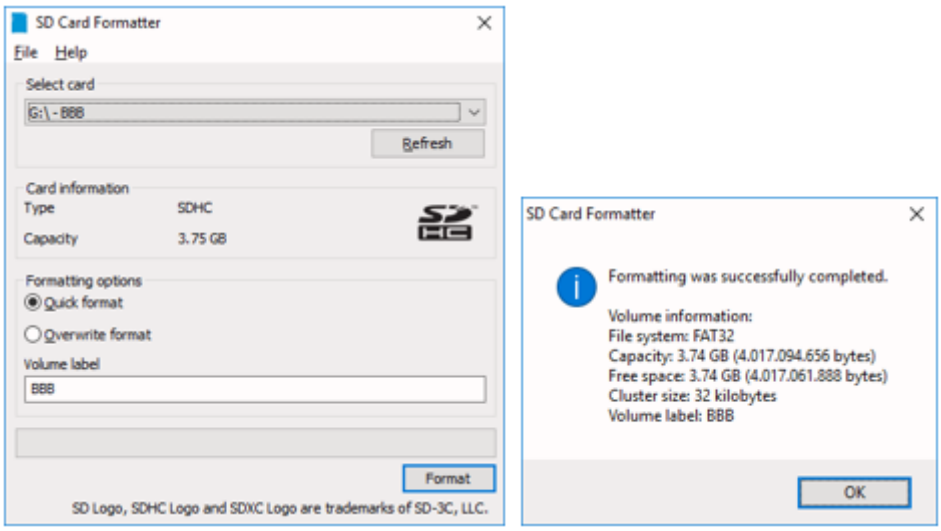


Figura 3.32 – *Software* de formação de cartões de memória

Em seguida, criamos um ficheiro texto no cartão de memória com o nome "uEnv.txt", com as seguintes linhas de texto:

```
mmcdev=1

bootpart=1:2

mmcroot=/dev/mmcblk1p2 ro
```

Estas linhas de comandos são necessárias porque, no arranque da BeagleBone Black é necessário perceber que o cartão de memória não está configurado com o SO, mas sim como uma parte extra de memória.

Com o ficheiro texto "uEnv.txt" criado no cartão de memória, insere-se o cartão na BeagleBone Black e edita-se o ficheiro "fstab" através do seguinte comando:

nano /etc/fstab

Adicionando a seguinte linha ao ficheiro:

```
/dev/mmcblk0p1   /media/card   auto   auto,rw,async,user,nofail  0  0
```

Que permite que seja criada a partição de memória externa no SO da BeagleBone Black. E finalmente é realizada uma reiniciação à BeagleBone Black e o cartão de memória irá aparecer como memória externa adicional na diretoria "/media/card".

Depois de adicionado o cartão de memória como memória externa adicional, é adicionada uma palavra-passe ao utilizador "root", para isso, basta executar o seguinte comando:

passwd

E a BeagleBone Black irá pedir para adicionar uma palavra-passe e reinserir novamente a palavra-passe adicionada, para mudar a palavra-passe posteriormente é usado o mesmo comando.

Depois de adicionada uma palavra-passe à BeagleBone Black, é adicionado um IP estático. Ao ser ligado um cabo Ethernet à BeagleBone Black, esta recebe um IP do *router* através de DHCP. Para verificar qual o IP recebido, basta visualizar o "IP address" na placa de rede "eth0" através do comando:

ifconfig

Para configurar o IP estático na BeagleBone Black é necessário editar o ficheiro "*interfaces*" através do seguinte comando:

nano /etc/network/*interfaces*

E adicionar as seguintes linhas de código com o IP estático desejado ("*address*"), a máscara de rede ("*netmask*"), o IP do *router* ("*gateway*") e servidores DNS ("*dns-nameservers*"). Nos servidores DNS pode-se usar o IP do *router* ou por exemplo um DNS público, como é caso dos IPs: 8.8.8.8 ou 4.4.4.4 (IPs DNS públicos da Google):

```
auto eth0
allow-hotplug eth0
iface eth0 inet static
    address 10.0.0.100
    netmask 255.255.255.0
```

Com o IP estático configurado, finalizamos todas as alterações necessárias no SO da BeagleBone Black e procedemos à configuração da hora e data no RTC. Posteriormente adiciona-se o *software* da *gateway* à BeagleBone Black e configurar este *software* no arranque da BeagleBone Black.

Para ler e configurar a hora e data no RTC, foram desenvolvidos dois *softwares* próprios na linguagem Python: O *software* de leitura, lê a hora e data registadas no RTC, o *software* de escrita, lê a hora atual da BeagleBone Black e escreve essa hora e data no RTC, bem como outras configurações necessárias para o bom funcionamento do RTC.

Para configurar a hora e data no RTC, são realizados os seguintes passos, pela ordem indicada:

1. Escrever a hora e data, mais as configurações necessárias;
2. Adicionar a pilha;
3. Ler a hora e data para verificar se escreveu corretamente e o cristal do RTC está funcional;
4. Desligar a alimentação da *shield* e voltar a ligar;
5. Voltar a ler a data e hora para verificar a funcionalidade em caso de falha de energia.

Depois de configurada a data e hora no RTC, é usado um novo *software* de acesso remoto, o WinSCP. Com este *software* é possível transferir ficheiros para qualquer diretoria da BeagleBone Black, pois este *software* usa os protocolos SFTP e FTP.

Através deste *software*, adicionamos o *software* desenvolvido (*software Gateway*) à BeagleBone Black bem como as bibliotecas de comunicação entre a *gateway* e servidor/base de dados desenvolvidas pela empresa contratada. Para além das bibliotecas é também configurado o IP e porta do servidor, bem como as credenciais de acesso da *gateway*. E por fim, é adicionado o ficheiro de configurações da *gateway* já configurada.

Depois de todos estes ficheiros adicionados e configurados, como último passo, através da ferramenta Linux "crontab", o *software gateway* é configurado para começar no arranque do SO da BeagleBone Black, ou seja, ao ligar a BeagleBone Black o *software gateway* arranca automaticamente.

4 Desenvolvimento de *Software*

Este capítulo apresenta através de fluxogramas toda a estrutura do sistema desenvolvido, onde foram realizadas 3 versões:

- Versão Teste (Inicial)
- Versão Final (Genérico)
- Versão Específica (GPS)

Na primeira versão, a versão teste, é desenvolvido um *software* apenas para recolha e processamento de dados, de apenas uma ou duas placas de sinais analógicos, com uma rotina de recolha de periodicidade de 30 minutos, ou de 60 minutos, e os dados são sempre gravados num ficheiro texto (.txt). Esta versão teste, para além do teste global das funcionalidades do microcomputador, permitiu também detetar possíveis erros, na comunicação com o *hardware* da Enging e possíveis melhoramentos no algoritmo para a versões seguintes.

Na segunda versão, a versão final (genérica) do sistema, o *software* desenvolvido está preparado para se conectar a várias placas de sinais analógicos e digitais, caso sejam necessárias. Nesta versão, as rotinas de recolha de dados, são realizadas na ordem de grandeza de minutos, com um mínimo de 3 minutos. Os dados depois de processados, são enviados para um servidor/base de dados, que foram desenvolvidos por uma empresa contratada, ou gravados em ficheiro texto em caso de falha de comunicação.

Na terceira versão, a versão específica, adaptada através da versão genérica, é composta por duas *gateway*s, onde as duas realizam a recolha de dados em sincronia, através do *timestamp* de um GPS ligado a cada uma das *gateway*s, e os dados são reunidos, na *gateway* "master" e enviados por esta para o servidor/base de dados ou gravados em ficheiro texto.

Nas diversas versões do sistema, todas seguem uma topologia mestre-escravo, onde a *gateway* desempenha o papel de mestre, e as placas de aquisição analógica e digital desempenham o papel/função de escravos.

Tendo em consideração os vários tipos de comunicação, mencionados no Capítulo 3, foi desenvolvido *software* para garantir a *interface* por:

- RS-232
- RS-485
- SPI
- TCP/IP

Para o desenvolvimento da *gateway*, a Enging desenvolveu uma *shield* dedicada à BeagleBone Black, que possui portas dedicadas aos vários tipos de comunicação referidos, entradas e saídas digitais de 24 V e um RTC.

4.1 Versão Teste – Inicial

Na Versão Teste (Inicial), onde se pretende executar uma recolha de dados com uma rotina de hora a hora, ou meia em meia hora, em horas certas, como por exemplo 12:00, 12:30 e 13:00, desenvolveu-se um *software* com apenas um ciclo, com início e fim, sem ciclo infinito, em que o agendamento deste ciclo é feito através de uma ferramenta Linux, o *crontab* e onde é possível fazer um agendamento de um ou mais *scripts Python*, de hora a hora ou meia em meia hora, em horas "certas", tal como é pretendido.

Esta versão está destinada apenas a recolher dados de uma ou duas placas de sinais analógicos, a processá-los e gravar os mesmos num ficheiro de texto.

Tal como apresentado no fluxograma geral (Figura 4.1), na rotina principal, no início são feitas as inicializações para *interface* com *hardware* externo (equipamentos da Enging), neste caso uma porta UART, uma SPI e três saídas digitais, de seguida são iniciadas as varáveis de aquisição. Depois de todas as inicializações concluídas é realizada a recolha e processamento de dados, e acaba a rotina principal.

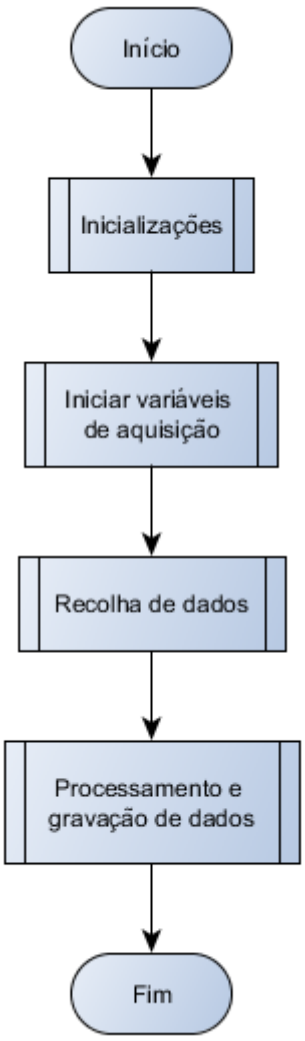


Figura 4.1 – Fluxograma Geral da Versão Teste (Inicial)

4.1.1 Inicializações

Como primeiro passo da rotina principal, temos as inicializações onde é configurada inicialmente a saída digital "TRIGGER", que irá ser utilizada durante a rotina de recolha de dados e que tem como função informar as placas de sinais analógicos quando devem adquirir os sinais. Depois são configuradas mais duas saídas digitais, dadas pelos nomes de "RE" e "DE", que têm como objetivo informar o conversor RS-485, se deve entrar em modo transmissão ou receção, tal como exemplificado no Secção 3.2.

Depois de configuradas todas as saídas digitais necessárias para o bom funcionamento do sistema, é iniciada e configurada a UART 1 da BeagleBone Black que é designada pela porta "/dev/ttyO1" para uma *baudrate* de 115200 bps, e todos os outros parâmetros são *standard* (*stopbit* = 1, *parity* = *None* e *timeout* = 1).

E por fim é iniciada e configurada a porta SPI 0, tal como exemplificado no Secção 3.2.5, com o modo a 0, ou seja, *clock* a 0 e fase a 0, a velocidade de clock (SCLK) é configurada a 4 MHz e o CS a "0", esta porta SPI tem como funcionalidade fazer *interface* com o RTC, para recolher o *timestamp* da aquisição de dados (Figura 4.2).

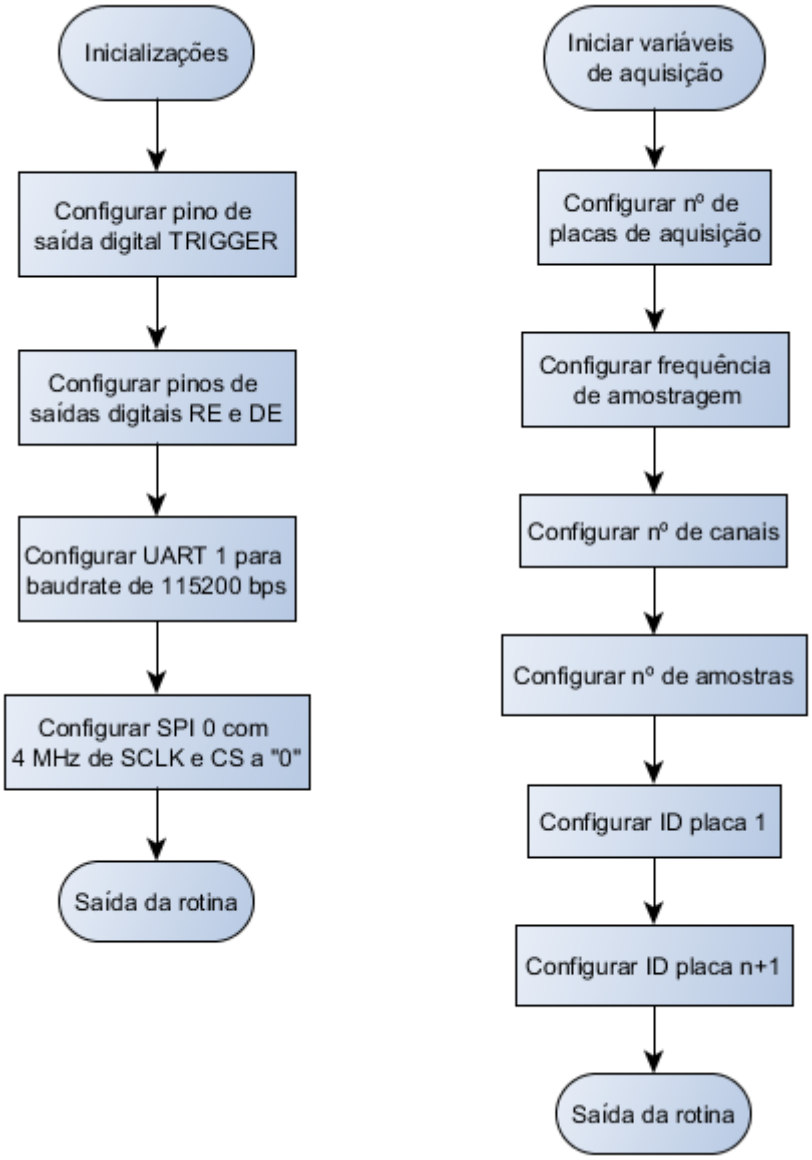


Figura 4.2 – Fluxograma de inicializações e iniciações de variáveis de aquisição.

Depois de realizadas todas as inicializações de *interface* com o *hardware*, são iniciadas e configuradas as denominadas variáveis de aquisição, estas que são usadas durante a comunicação com as placas de sinais analógicos, para as informar dos parâmetros da aquisição. Estas variáveis de aquisição são: frequência de amostragem, número de canais (a utilizar da placa de sinais analógicos) e o número total de amostras que desejamos adquirir. Todas estas variáveis têm um valor máximo, a frequência máxima de amostragem é 100 kHz, um total de 6 canais e 25 mil amostras.

Além de iniciadas e configuradas as variáveis de aquisição, são também configuradas as variáveis de sistema, que informam o *software* do que está presente no sistema configurado, estas são: o número de placas de sinais analógicos e os seus respetivos IDs (estes servem para identificar com que placa analógica a *gateway* deseja comunicar no

barramento RS-485). Depois de todas as variáveis iniciadas e configuradas, o *software* sai desta rotina.

4.1.2 Recolha de dados

Depois de concluída a rotina de inicializações, o *software* entra na rotina de recolha de dados, que tal como o nome indica, é a rotina que recolhe os dados adquiridos pelas placas de sinais analógicos. Esta rotina incluí o protocolo proprietário de comunicação entre a *gateway* e as placas de aquisição de sinais analógicos (Figura 4.3).

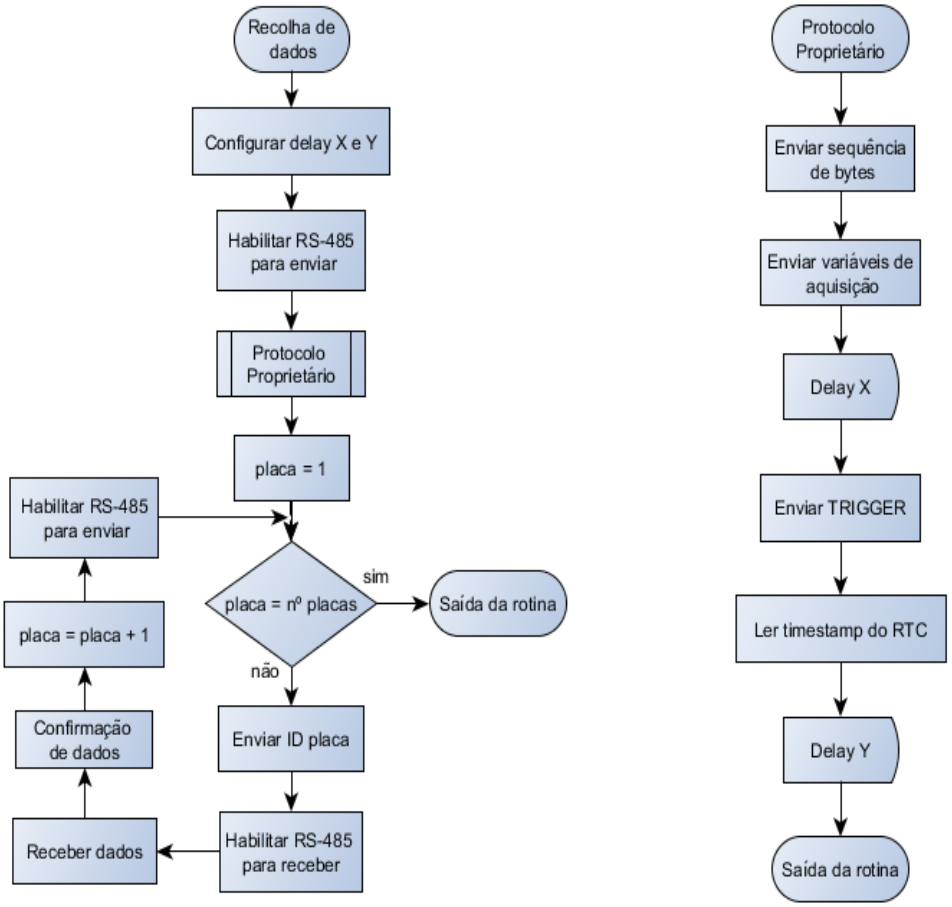


Figura 4.3 – Fluxograma de recolha de dados e protocolo proprietário.

Tal como mostrado na Figura 4.3, inicialmente na rotina de recolha de dados são configurado os *delays* X e Y, que são usados no protocolo proprietário, depois o RS-485 é configurado para modo de transmissão, seguido do protocolo proprietário.

O protocolo proprietário é usado para sincronizar e possibilitar a comunicação entre a *gateway* e as placas de aquisição de sinais analógicos, com o objetivo de controlar a transferência de dados entre os dispositivos no barramento RS-485.

Neste protocolo proprietário, inicialmente é enviado uma sequência de *bytes* para todos os dispositivos no barramento RS-485. As placas analógicas depois de receberem esta sequência ficam preparadas para receber as variáveis de aquisição, e como explicado na

secção anterior onde são inicializadas estas variáveis. Depois de enviadas as variáveis de aquisição, é esperado o *delay* X. Este *delay* foi determinado em laboratório, através da utilização de um osciloscópio, que corresponde ao tempo que o microcontrolador mestre das placas analógicas demora a enviar aos seus escravos as variáveis de aquisição, previamente enviadas pela *gateway*.

Depois de esperado o *delay* X, é enviado o sinal digital "TRIGGER" para as placas analógicas, que as informa para adquirir os sinais analógicos e de seguida, ler o *timestamp* do RTC, que é designado por *timestamp* da aquisição.

Após lido o *timestamp* da aquisição, é esperado novamente um *delay*, agora o *delay* Y, calculado através da seguinte equação:

$$\text{Delay Y} = \frac{número\ de\ amostras}{frequência\ de\ amostragem} + 0.5 \qquad (1)$$

O *delay* Y corresponde ao tempo que as placas analógicas demoram a adquirir os sinais, que corresponde à fração entre o número de amostras e a frequência de amostragem, somado do tempo que o microcontrolador da placa analógica demora a processar os dados para posteriormente enviar à *gateway*. Após o *delay* Y, é fechado o ciclo do protocolo proprietário.

Com o protocolo proprietário finalizado, vem a recolha de dados, através de um ciclo *for* que começa na placa número 1, e acaba no número total de placas analógicas no sistema. Apesar de nesta versão teste, haver um máximo de duas placas analógicas, o *software* está preparado para mais. Assim sendo, no início do ciclo, é enviado o ID da placa número 1, para pedir os dados por ela adquiridos, e logo de seguida o RS-485 é mudado para modo receção e são recolhidos os dados da placa cujo número total de *bytes* a receber, que pode ser calculado pela seguinte expressão:

$$Número\ total\ de\ bytes\ a\ receber = Número\ de\ canais * (Número\ de\ amostras * 1000 * 2 + 7) + 6 \qquad (2)$$

O número total de *bytes* a receber é composto pela multiplicação entre o número total de canais e o número de amostras multiplicado por 1000, porque esta multiplicação é realizada na placa analógica, ou seja, quando queremos 10000 amostras, configuramos o número 10 na inicialização de variáveis de aquisição, depois é multiplicado por 2 porque a resolução de cada amostra é de 16 bits, ou seja, cada amostra são 2 *bytes*, depois é somado 7 porque no inicio de cada canal de amostras é recebido o ID da placa (6 *bytes*) mais o número do canal (1 *byte*) e no final é somado 6, porque o ID da placa é recebido novamente no final dos dados.

Depois de receber o número total de *bytes*, é realizada a confirmação de dados, que consiste em comparar os *bytes* recebidos no início de cada canal, ou seja, o ID e o número de canal e o ID no final dos dados. Se a confirmação falhar, apenas é avisado o utilizador, que os dados são inválidos. Na versão teste, não há qualquer proteção contra dados inválidos.

Após a confirmação dos dados ser bem-sucedida, é incrementada a variável "placa", para realizar o mesmo processo para a placa analógica seguinte, e se o número da variável placa for igual ao número total de placas, é fechada a rotina de recolha de dados, seguindo para o processamento dos mesmos.

4.1.3 Processamento e gravação de dados

Com a rotina de recolha de dados finalizada, inicia-se a última rotina, a rotina de processamento e gravação de dados, onde são separados os dados recebidos por placa, por canal e por amostra através de ciclos *for*, e depois os dados processados são gravados num ficheiro texto de acordo com uma estrutura pré-definida (Figura 4.4).

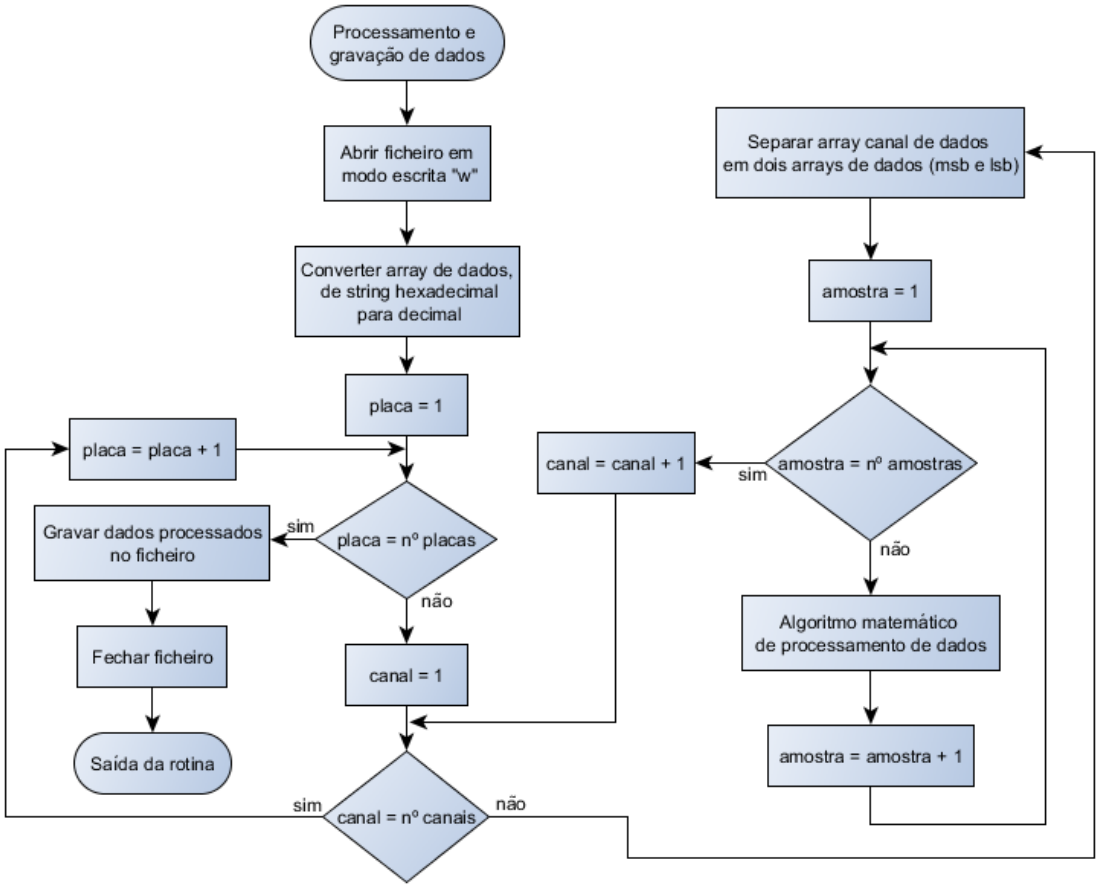


Figura 4.4 – Fluxograma de processamento de dados

Tal como é exemplificado no fluxograma de processamento de dados, na Figura 4.4, inicialmente é aberto o ficheiro texto em modo escrita ("w"), com o título composto pela diretoria onde vai ser gravado o ficheiro, o nome "Teste", mais o *timestamp* da aquisição (Ex: "/testes/Teste_2017-04-02 13:00:00.txt").

Após o ficheiro texto ser aberto, passamos ao processamento de dados, começando por converter todos os dados recebidos de *string* hexadecimal, para valores decimais, e dá-se início ao ciclo *for* das placas, estando cada *array* de dados recebidos separado por placa, os dados são novamente separados, agora por canal.

Estando agora os dados separados por canal, é feito novamente uma separação, o *array* de dados por canal é separado em dois, o *array msb* e o *array lsb*, porque cada amostra dentro do *array* é composta por 2 *bytes*, o *msb* e o *lsb*. Sendo feita esta separação, é executada a seguinte equação:

$$Amostra = 256 * MSB + LSB \qquad (3)$$

Depois de calculada a equação acima, é novamente realizada uma operação matemática, de acordo com o *datasheet* do ADC que integra a placa de sinais analógicos.

Finalizados todos os ciclos *for*: placa, canal e amostra, os dados processados e ficam gravados numa lista de *arrays* separada por canal (Ex: {"placa 1": [canal 1, …, canal n], "placa 2": [canal 1, …, canal n]}). Os dados são gravados no ficheiro de texto previamente aberto, de acordo com a estrutura da Figura 4.5, onde é exemplificado pelos dados de uma só placa, com 6 canais e 1000 amostras, sendo os dados dos canais separados pelo espaçador "TAB".

Canal 1	Canal 2	Canal 3	Canal 4	Canal 5	Canal 6
amostra 1	amostra 1	amostra 1	amostra 1	amostra 1	amostra 1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
amostra 1000	amostra 1000	amostra 1000	amostra 1000	amostra 1000	amostra 1000

Figura 4.5 – Estrutura dos dados gravados no ficheiro texto

Depois de gravados os dados processados no ficheiro, de acordo com a estrutura exemplificada, é fechado o ficheiro, e finalizada a rotina de processamento de dados, a última rotina da versão teste (Inicial).

4.2 Versão Final – Genérico

Com a versão teste finalizada, os testes de funcionamento das *interfaces* com *hardware* externo realizadas com sucesso, e desenhado um algoritmo melhorado e otimizado para a versão final, passou-se à codificação da mesma. A Versão final, ou Versão genérica, tem como objetivo recolher e processar dados de dois tipos diferentes de placas, placas de sinais analógicos e placas de sinais digitais, sem limite de número de placas de cada tipo.

A grande diferença em comparação com a Versão Teste é a recolha de dados. Na Versão Teste, a recolha era realizada por agendamento, através da ferramenta *crontab*. Na Versão Final é utilizado o relógio interno do processador da BeagleBone Black para contabilizar o tempo da rotina, o script *Python* é iniciado apenas uma vez, através de ciclos infinitos fica constantemente a correr. A ferramenta *crontab* é novamente utilizada para, em caso de falha de energia, o *script Python* correr automaticamente no arranque do sistema.

Nas inicializações são acrescentadas mais duas UARTs e uma entrada digital. A recolha de dados tem a mesma lógica que a Versão Teste, mas é acrescentada a recolha de dados digitais após a recolha de dados analógicos. O processamento de dados já tem uma lógica diferente, pois os dados processados são enviados para um servidor/base de dados apenas em caso de falha de comunicação com o servidor. Os dados processados são gravados em ficheiro texto de acordo com a mesma lógica da Versão Teste.

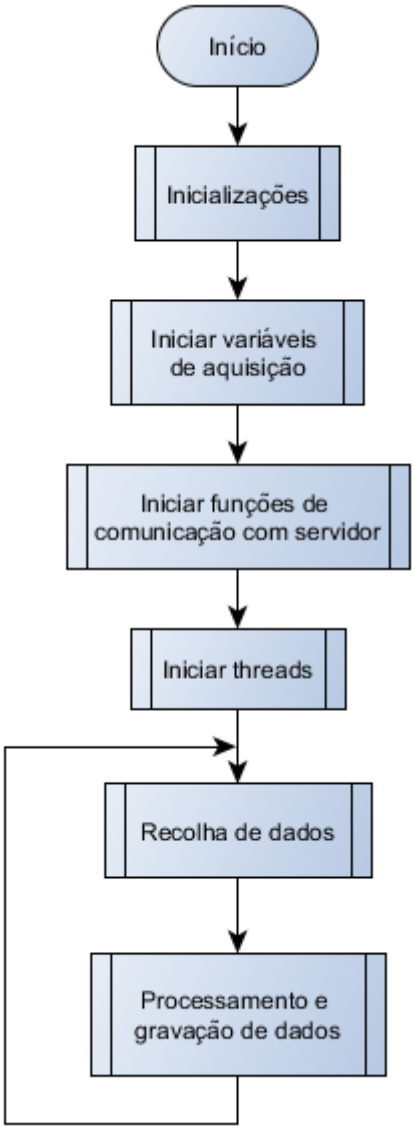


Figura 4.6 – Fluxograma Geral da Versão Final

Tal como demonstrado no fluxograma geral (Figura 4.6), inicialmente são realizadas as inicializações e as variáveis de aquisição, depois são iniciadas as funções de comunicação com o servidor/base de dados, e logo de seguida, são iniciadas as *threads*, estas que incluem os ciclos infinitos, onde a principal rotina infinita contém a recolha, processamento e gravação de dados.

4.2.1 Inicializações

Nas inicializações da Versão Final, há um grande *update* em comparação com a Versão Teste, para além das inicializações das *interfaces* com o *hardware* externo, foi acrescentado um ficheiro de configurações e um ficheiro de *logs*, onde o ficheiro de configurações inclui todas as variáveis iniciadas nas inicializações e nas variáveis de aquisição, e o ficheiro de *logs* inclui todas as informações, desde o estado de variáveis e informações sobre o decorrer do programa. Estes dois ficheiros vão ser descritos na secção seguinte.

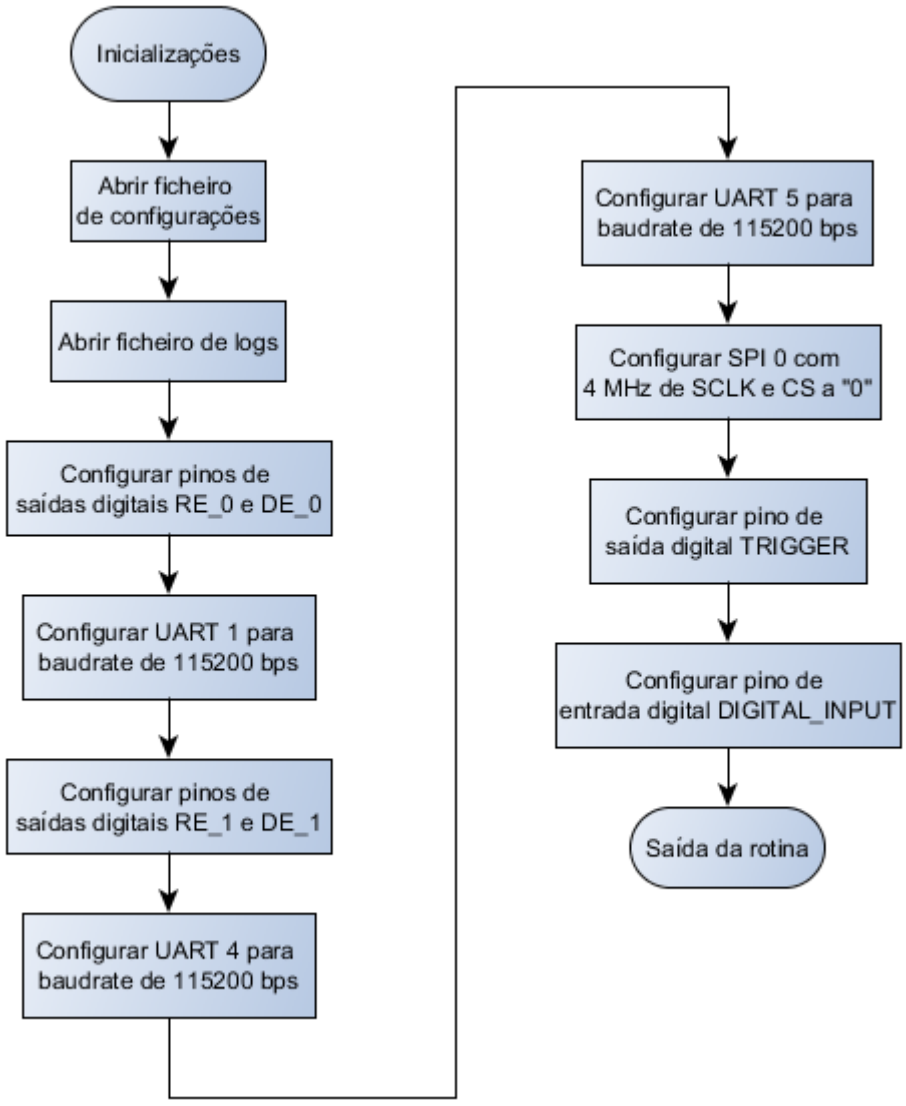


Figura 4.7 – Fluxograma de inicializações da versão final

Tal como demonstrado no fluxograma de inicializações da Versão Final, inicialmente é aberto o ficheiro de configurações, são lidas todas as variáveis, seguido da abertura do ficheiro de *logs*.

De seguida, são configuradas as duas saídas digitais: "RE_0" e "DE_0" para controlo do conversor RS-485 que vai ser usado no barramento de comunicação RS-485 do sistema,

onde é configurada a UART 1 com uma *baudrate* de 115200 *bps*. Nesta versão é usado um segundo conversor RS-485, para caso de falha ou avaria do primeiro, que é configurado com a UART 4 também com uma *baudrate* de 115200 bps e com as saídas digitais: "RE_1" e "DE_1" para controlo do mesmo. Para além da UART 1 e da UART 4, é configurada a UART 5 com uma *baudrate* de 115200 *bps* que vai ser usada posteriormente na versão específica para o barramento RS-232.

Depois de iniciadas e configuradas as UARTs, é configurada a *interface* SPI 0, com uma velocidade de *clock* de 4 MHz e o CS a "0".

Finalizando a rotina das inicializações, temos a configuração da saída digital "TRIGGER" e a entrada digital "DIGITAL_INPUT", que apenas é usada para *interface* com as placas de sinais digitais.

4.2.2 Ficheiros de configurações e de *logs*

Tal como referido na secção anterior, houve um importante *update* da Versão Teste para a Versão Final na rotina das inicializações, pelo acréscimo do ficheiro de configurações e do ficheiro de *logs*. Numa versão intermédia, o ficheiro de configurações era de extensão ".cfg", que era a extensão que a empresa usava para desenvolver os seus ficheiros de configurações. Como a empresa contratada que desenvolveu a base de dados e servidor que recebe os dados processados pela *gateway* trabalha apenas com ficheiros de configurações de extensão ".json", o ficheiro de configurações da *gateway* na versão final passou a ser de extensão ".json", para facilitar a agregação desta parte do *software* com o trabalho desenvolvido pela empresa contratada.

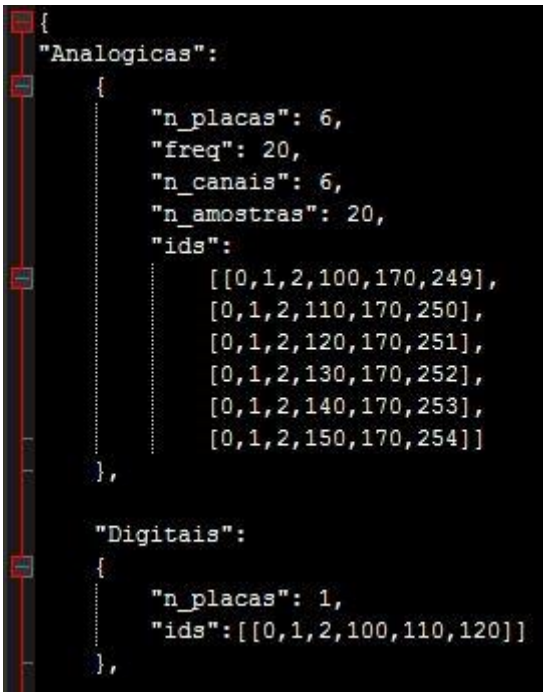


Figura 4.8 – Excerto do ficheiro de configurações

Tal como demonstrado neste excerto do ficheiro de configurações (Figura 4.8), são configurados os números de placas analógicas e digitais, os seus respetivos IDs, bem como todas as variáveis de aquisição:

- Frequência de amostragem (kHz)
- Número de canais
- Número de amostras (*1000)

Ainda neste ficheiro de configurações, são configuradas todas as variáveis relativas à configuração de todas as UARTs, os pinos de entradas e saídas digitais, o tempo de rotina de aquisições e as informações do cliente.

O ficheiro de *logs* é onde são gravadas todas as informações de tempo-real dos processos que a *gateway* está a desempenhar, se a *gateway* está a executar uma recolha de dados, se está a processar ou a gravar dados.

```
DEBUG:Gateway:Logs Created
INFO:root:Attempting to connect websocket.
INFO:root:Connection established
INFO:Gateway:Start
INFO:Gateway:1100:First time schedule trigger at 2017-05-29 16:55:00
DEBUG:Gateway:1120:>>>>>>>>>>     Main Interrupts active, aquisition will start at 2017-05-29 16:55:00
DEBUG:Gateway:1200:Broadcast sent at 2017-05-29 16:55:00(acquitision timestamp)
INFO:Gateway:1201:Collecting data from Analog Boards(AI) will start at 2017-05-29 16:55:00
DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete - (1,2,3,110,170,153)
DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete - (1,2,3,110,170,154)
DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete - (1,2,3,110,170,155)
DEBUG:Gateway:1204:Collecting data from all Analog Boards(AI) in the system Complete
DEBUG:Gateway:1250:Collecting from Digital Board(AI) Complete - (100,110,120,130,140,150)
DEBUG:Gateway:1256:Collecting data from all Digital Boards(AI) in the system Complete
INFO:Gateway:1257:Enter in Process Mode-Digital
INFO:Gateway:1350:Processing Digital Data
INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS connection (1): 10.0.0.2
DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings HTTP/1.1" 200 82
DEBUG:Gateway:1020:Analog Reading successfully sent
INFO:Gateway:1207:Enter in Process Mode-Analog
INFO:Gateway:1310:Processing Analog Data
INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS connection (1): 10.0.0.2
DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings HTTP/1.1" 200 82
DEBUG:Gateway:1020:Analog Reading successfully sent
DEBUG:Gateway:1208:Process and Analog Data Record Complete at - 2017-05-29 16:56:23
DEBUG:Gateway:1122:>>>>>>>>>>     Main Interrupts finish at 2017-05-29 16:56:23
DEBUG:SDcard:1500:Analyzing sdCard, 0 files detected
```

Figura 4.9 – Excerto inicial de ficheiro de *logs*

Como demonstrado no excerto inicial de um ficheiro de *logs* (Figura 4.9), depois de criado, é estabelecida a comunicação com o servidor/base de dados e a *gateway* começa as suas *threads*. Quando é detetado um evento, neste caso foi um evento de *timer*, onde é iniciada a recolha de dados analógicos, são recolhidos os dados das placas digitais e os mesmos são processados. Estes dados são posteriormente enviados, é recebida a confirmação que foram recebidos. É feito o mesmo processamento e envio dos dados analógicos, recebida a confirmação do servidor, e é finalizada esta rotina até ser detetado um novo evento.

Tal como demonstrado, o *log* é composto por 4 partes, pela ordem respetiva:

- Tipo
- Origem
- Código
- Informação

Existe 5 tipos de *logs*:

- *Debug*
- *Info*
- *Warning*
- *Error*
- *Critical*

Estes tipos de *logs* facilitam a separação dos mesmos por categorias, desde *logs* de informação a *logs* de erros.

Os *logs* podem ter 3 origens diferentes:

- *Gateway*
- *SDcard* (Funções de monitorização do cartão de memória)
- Servidor (Comunicação com o servidor)

Como podemos ver pela composição dos *logs*, e no exemplo demonstrado (Figura 4.9), cada *log* tem um código associado, facilitando ao utilizador procurar o *log* que deseja encontrar. Os *logs* foram escritos na língua inglesa, de acordo com o pedido da empresa, para posteriormente facilitar a internacionalização do produto.

4.2.3 Funções de comunicação com servidor/base de dados

Tal como descrito anteriormente, o servidor e base de dados onde são armazenados os dados recolhidos pela *gateway*, foram desenvolvidos por uma empresa contratada devido à sua complexidade. Assim, a mesma empresa desenvolveu várias bibliotecas para instalar na *gateway*. As funções de comunicação entre *gateway* e servidor, que estão incluídas no *software* desenvolvido para a versão final, foram desenvolvidas em conjunto com um colaborador da empresa contratada.

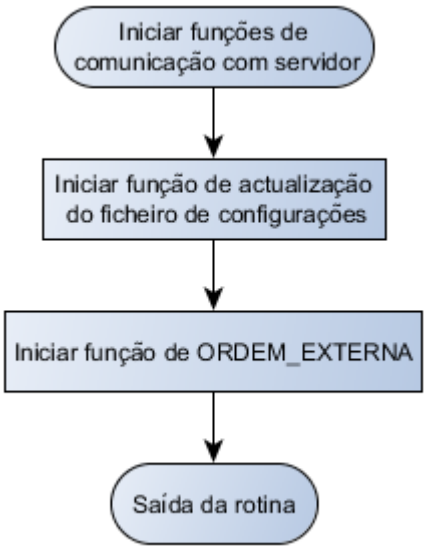


Figura 4.10 – Fluxograma de funções de comunicação com servidor

Tal como demonstrado no fluxograma de funções de comunicação entre *gateway* e servidor (Figura 4.10), são iniciadas duas funções que têm como objetivo receber instruções do servidor: a função de atualização do ficheiro de configurações e a função de ordem externa. Estas funções são iniciadas logo após a configuração das variáveis de aquisição, tal como demonstrado no fluxograma geral (Figura 4.6).

Após estas funções serem iniciadas, o *software* sai desta rotina, porque existem ciclos infinitos que mantêm as funções sempre à "escuta" nas bibliotecas desenvolvidas pela empresa contratada.

A função de atualização do ficheiro de configurações, tal como o próprio nome indica, tem como objetivo receber alterações no ficheiro de configurações, sendo que esta função grava as alterações recebidas a partir do servidor, e na próxima rotina, a rotina das *threads*, recebe a informação desta função, que vai reiniciar o processo principal do *software* para ler o novo ficheiro de configurações.

A função de ordem externa tem como objetivo receber uma ordem de recolha de dados a partir do servidor, e tal como referido anteriormente, a rotina das *threads* recebe esta informação e é realizada uma recolha de dados.

4.2.4 *Threads*

Depois de realizar todas as inicializações gerais, de variáveis de aquisição e de funções de comunicação com o servidor, são iniciadas as *threads*, onde cada *thread* corresponde a um processo diferente, que são todos iniciados ao mesmo tempo, e dentro de cada *thread* está associado um ciclo infinito.

Este ciclo corresponde ao chamado ciclo "*Main*", onde inicialmente as funções respetivas são configuradas como *threads*, e posteriormente todas iniciadas simultaneamente, através de um "*join*".

Existem 3 *threads*: "TIMER", "INPUT_DI" e "MAIN" (Figura 4.11), onde cada *thread* corresponde unicamente ao seu processo específico, mas existe partilha de informação entre elas. Quando existe um evento no processo "TIMER" ou no processo "INPUT_DI", estes enviam a informação do mesmo para o processo "MAIN". Estes processos funcionam com a mesma filosofia de um microcontrolador e as suas interrupções, mas neste caso em programação de alto-nível, que neste caso podemos associar o processo "MAIN" ao microcontrolador, e os processos "TIMER" e "INPUT_DI" a interrupções.

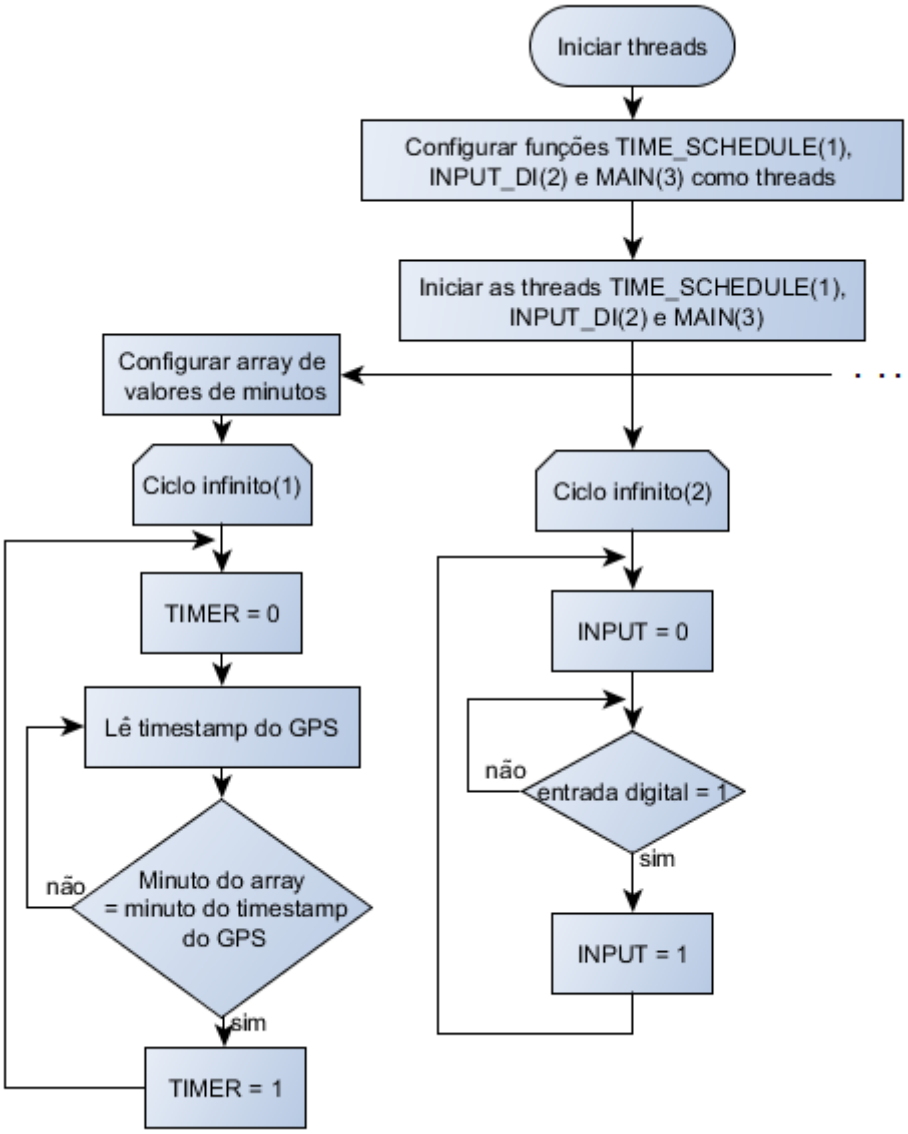


Figura 4.11 – Fluxograma das *threads* (Parte 1)

Como descrito anteriormente, no início desta rotina as funções MAIN, TIMER e "INPUT_DI", são configuradas como *threads* e posteriormente iniciadas (Figura 4.11).

Na primeira parte do fluxograma das *threads* (Figura 4.11), temos os ciclos infinitos: "TIMER"(1) e "INPUT_DI"(2), e como demonstrado neste fluxograma, antes de iniciar o ciclo infinito(1), é configurado o *array* de valores de minutos, isto é, como sabemos, uma hora corresponde a 60 minutos, assim sendo, os valores de rotina de aquisição permitidos, são os divisores inteiros de 60, estes são:

- {1,2,3,4,5,6,10,12,15,20,30,60}

Mas como o objetivo desta *gateway* são aquisições de rotina rápidas, os valores escolhidos para o tempo de rotina de aquisição são sempre entre o minuto 1 e o minuto 5. Sendo assim, ao ser escolhido o tempo de rotina, por exemplo se for escolhido 5 minutos para o tempo de rotina, é criado o *array* de valores de minuto seguinte:

- {00,05,10,15,20,25,30,35,40,45,50,55}

Depois de configurado o *array* de valores de minutos, é iniciado o ciclo infinito(1), começando com a variável "TIMER" = "0", é lido constantemente o *timestamp* do RTC, até chegar ao minuto igual a um dos valores configurados no *array* de valores de minutos. Assim quando o valor de minuto do RTC corresponde a um valor do *array*, por consequência o valor dos segundos é igual a 0, a variável "TIMER" muda para o estado "1". Assim sendo, é informado o processo "MAIN" do evento ativado pelo processo TIMER, e depois a variável "TIMER" volta ao seu estado inicial "0", até ser encontrado um novo evento.

No ciclo infinito(2), que corresponde ao processo "INPUT_DI", este tem como função ler constantemente a entrada digital, "DIGITAL_INPUT". Ao ser iniciado o ciclo infinito(2), a variável "INPUT" é iniciada no estado "0" e quando o estado da entrada digital for "1", a variável "INPUT" muda para o estado "1", que tal como o processo "TIMER", informa o processo "MAIN" do evento ocorrido, e depois volta ao estado "0".

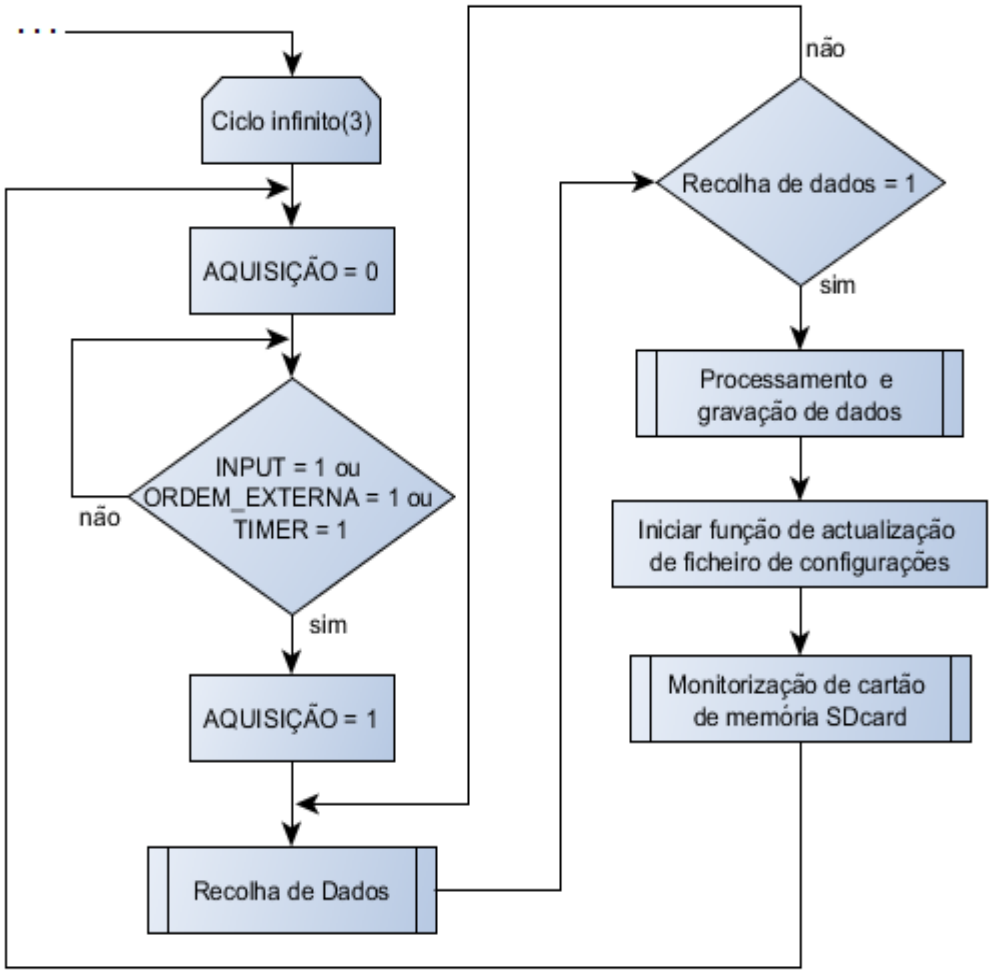


Figura 4.12 – Fluxograma das *threads* (Parte 2)

Com a parte 1 do fluxograma das *threads* (Figura 4.11) demonstrada, temos a parte 2 do fluxograma das *threads* (Figura 4.12), que corresponde ao processo "MAIN". Este processo engloba várias rotinas, como: recolha de dados, processamento e gravação de dados, função de atualização de ficheiro de configurações e monitorização do cartão de memória *SD Card*.

Tal como demonstrado, com a inicialização do ciclo infinito(3), temos a configuração da variável "AQUISIÇÃO" no estado "0". Esta variável. além de ser uma variável de informação, também irá interagir com a rotina de monitorização do cartão de memória *SD Card*. Posteriormente à configuração da variável "AQUISIÇÃO", é esperado um evento do processo "TIMER", que corresponde à variável "TIMER", do processo "INPUT_DI", que por sua vez corresponde à variável "INPUT" ou da função de comunicação com o servidor, associada à variável "ORDEM_EXTERNA". Esta comparação do estado das variáveis anteriormente referidas tem certas proteções e uma certa prioridade. O evento com mais prioridade é o "INPUT", depois o evento de "ORDEM_EXTERNA" e por último o evento "TIMER".

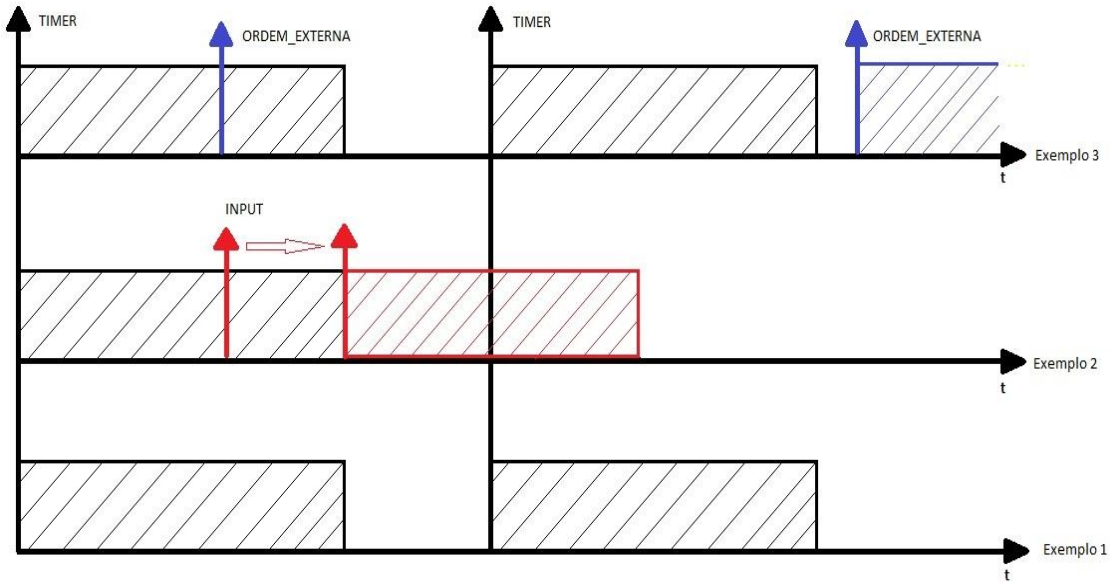


Figura 4.13 – Exemplos de proteções e prioridades de eventos no processo "MAIN"

Como demonstrado na Figura 4.13, nenhuma tarefa de recolha de dados e processamento dos mesmos, é sobreposta ou interrompida por outros eventos. No exemplo 1 temos apenas o evento de rotina "TIMER".

No exemplo 2 temos o evento "INPUT" durante o evento "TIMER", este evento não é cancelado, mas quando é finalizado este evento, posteriormente, é realizada a recolha e processamento de dados proveniente do evento "INPUT", e como este evento atinge o evento de rotina "TIMER", este não é realizado.

Já no exemplo 3 temos o evento de "ORDEM_EXTERNA" durante o primeiro evento de "TIMER", diferente do evento "INPUT", este não é realizado posteriormente ao evento "TIMER". Ainda neste exemplo, depois do segundo evento "TIMER", temos novamente

a ocorrência do evento "ORDEM_EXTERNA", este já é realizado por não haver qualquer evento durante a sua ocorrência, e por consequência irá apenas terminar, depois da ocorrência do evento "TIMER", este também não será realizado.

Assim sendo, tal como demonstrado na Figura 4.12, depois da deteção de um evento, a variável AQUISIÇÃO muda para o estado "1", e é executada a rotina de recolha de dados. Tal como podemos ver no fluxograma, se a recolha de dados falhar, é realizada novamente. Depois de ser bem-sucedida é executada a rotina de processamento e gravação dos dados recolhidos que, quando finalizada, no tempo restante até ao próximo evento, é executada a função de atualização de ficheiro de configurações.

Esta função tem como objetivo a deteção de alterações no ficheiro de configurações, no caso de ser alterado manualmente, ou de receber a informação, proveniente da função iniciada antes das inicializações, que recebe as alterações do ficheiro de configurações, via servidor.

Depois de executada esta função, é iniciada a rotina de monitorização do cartão de memória *SDcard*, e no fim desta rotina, é reiniciado o ciclo do processo "MAIN".

4.2.5 Recolha de dados

A recolha de dados é a primeira rotina do processo "MAIN" e, em relação à rotina de recolha de dados da Versão Teste (Figura 4.3), existem alguns incrementos e proteções, no entanto, a lógica mantém-se.

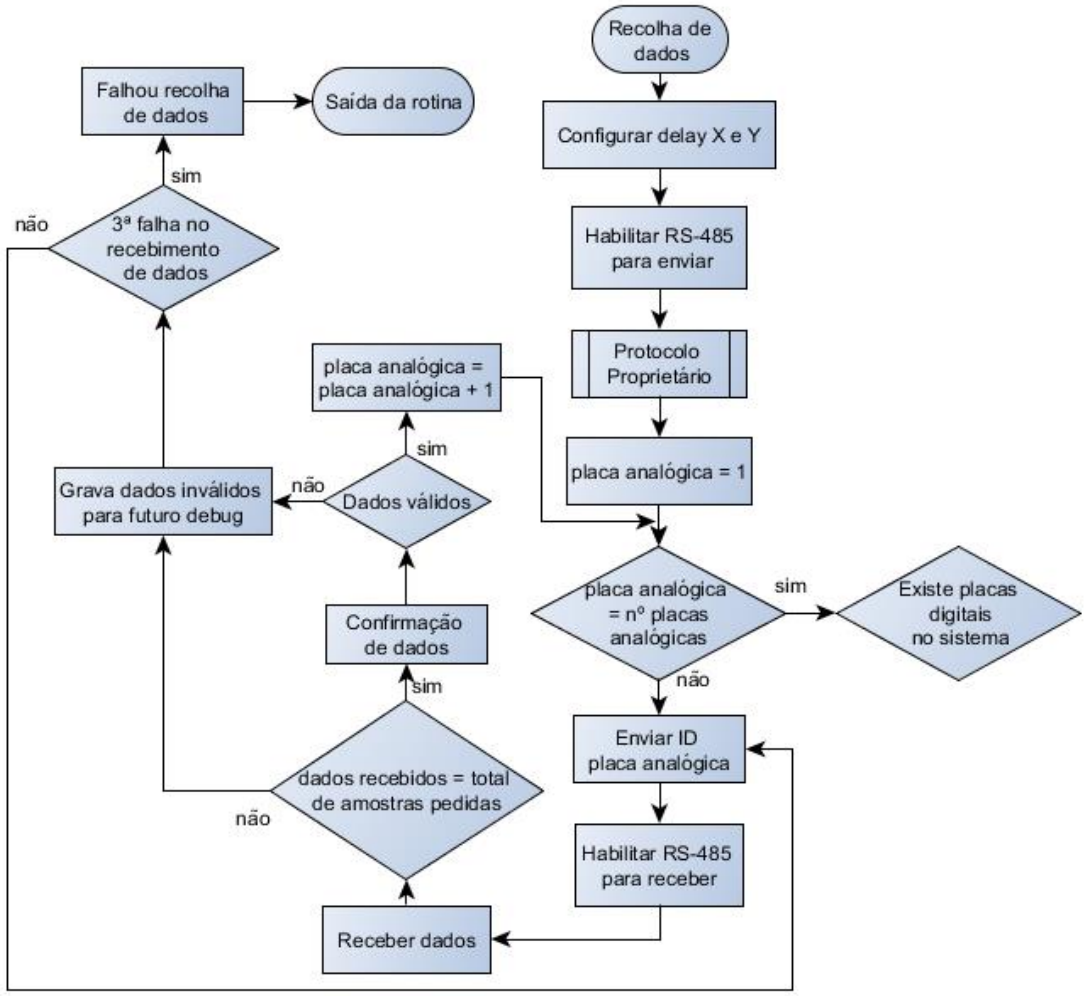


Figura 4.14 – Fluxograma de recolha de dados (Parte 1)

Tal como demonstrado na primeira parte (recolha de dados analógicos) do fluxograma de recolha de dados (Figura 4.14), inicialmente é configurado o *delay* X e Y, que já foram apresentados na secção 4.1.2, depois o conversor RS-485 é habilitado para modo de transmissão, seguido do protocolo proprietário (Figura 4.3) que foi exemplificado na mesma secção.

Seguido do protocolo proprietário vem o ciclo "*for*" em relação ao número de placas analógicas, começando com a variável inteira "placa analógica" a 1, e este ciclo é iniciado pelo envio do ID da placa analógica que corresponde ao número inteiro da variável "placa analógica", seguido da habilitação do conversor RS-485 para modo de receção e são recebidos os dados. Depois de recebidos os dados existem várias proteções para a validação dos mesmos.

A primeira proteção para os dados recebidos é a confirmação do número total de *bytes* recebidos. Se a confirmação falhar, o número de dados recebidos, os dados recebidos e o seu *timestamp* são gravados num ficheiro texto para um futuro *debug* da falha encontrada. A segunda proteção, já existia na rotina de recolha de dados da Versão Teste, e foi exemplificado na secção correspondente, mas agora, tal como na proteção, os dados são

gravados num ficheiro texto para futuro *debug*. Se os dados forem validados na segunda proteção, é incrementada a variável "placa analógica" dentro do ciclo "*for*" do número de placas analógicas.

Se houver uma terceira falha na recolha de dados, é informado o processo "MAIN" da mesma, e tal como demonstrado no fluxograma do processo "MAIN" (Figura 4.12), este processo é reiniciado.

Se houver validação de todos os dados recebidos em todas as placas analógicas do sistema, é verificada a existência de placas digitais no sistema, e se for confirmada a sua existência é iniciada a recolha dos dados respectivos, caso contrário é iniciado o processamento e gravação dos dados analógicos (Figura 4.15).

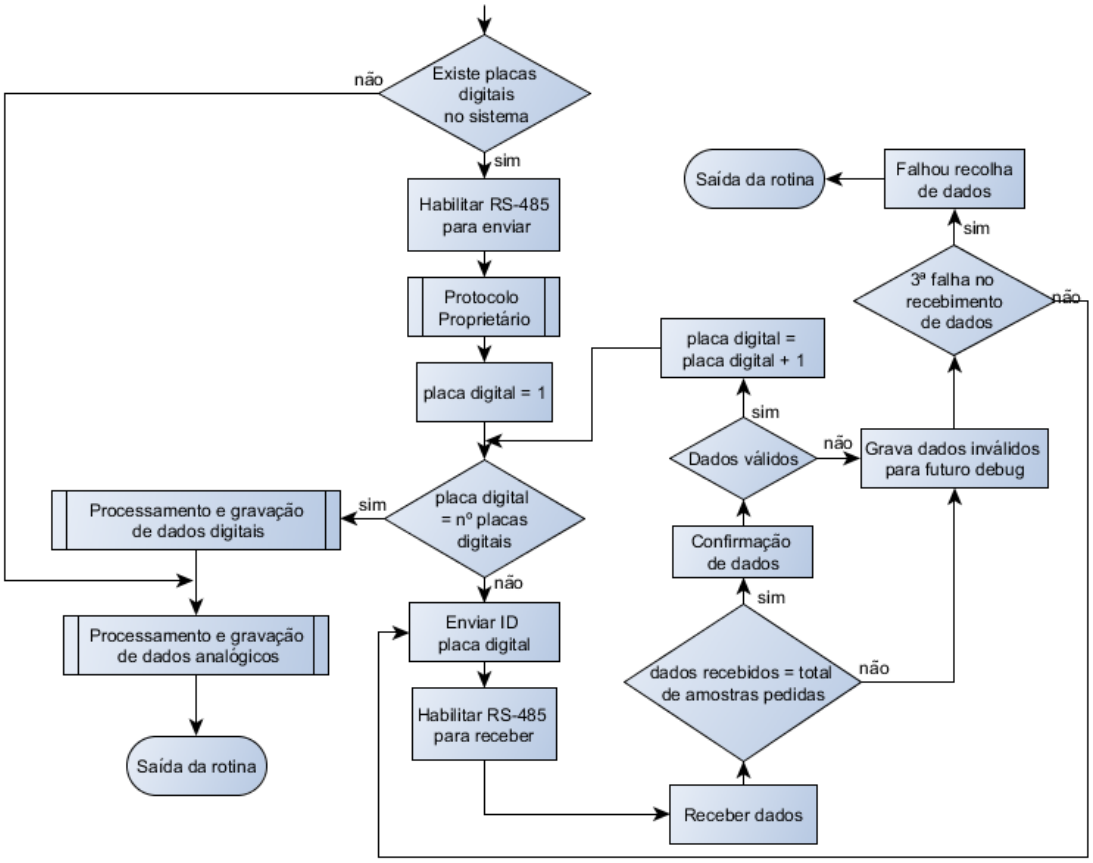


Figura 4.15 – Fluxograma de recolha de dados (Parte 2)

Se existirem placas digitais no sistema, tal como demonstrado na segunda parte (recolha de dados digitais), é realizado o mesmo procedimento e as mesmas proteções que foram realizadas para as placas analógicas, desde todas as validações de dados, bem como retornar um parâmetro de sucesso ou erro nos dados recebidos. Quando o ciclo "*for*" do número de placas digitais for concluído, acaba a rotina de recolha de dados e é realizado o processamento e gravação de dados digitais, e posteriormente o processamento gravação dos dados analógicos.

4.2.6 Processamento e gravação de dados

Depois da rotina de recolha de dados, vem a penúltima rotina no processo "MAIN", a rotina de processamento e gravação dos dados recolhidos. Em comparação com a rotina de processamento e gravação de dados da Versão Teste (Figura 4.4), há o acréscimo do processamento e gravação de dados digitais, como também a calibração de valores analógicos.

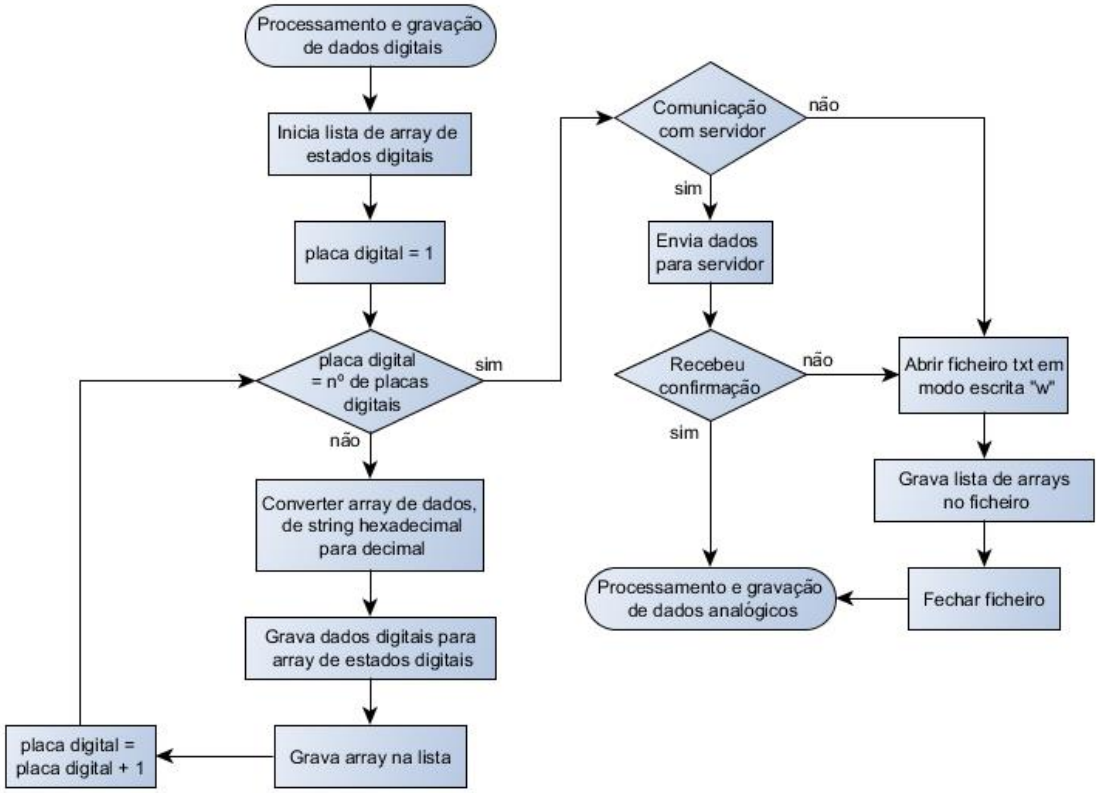


Figura 4.16 – Fluxograma de processamento e gravação de dados digitais

Primeiro é realizada a rotina de processamento e gravação de dados digitais, e tal como demonstrado no fluxograma desta rotina (Figura 4.16), inicialmente é iniciada uma lista de *arrays*, onde vão ser alocados os dados processados. Depois é iniciada a variável inteira "placa digital" a 1, para ser iniciado um ciclo "*for*" para o número de placas digitais. Este ciclo "*for*" começa por converter os dados de *string* hexadecimal para valores decimais, que são gravados num *array*, e posteriormente gravado na lista de *arrays* configurada no início da rotina. Por fim é incrementada a variável "placa digital", e quando o ciclo "*for*" é finalizado, é realizada a comunicação com o servidor. Se for validada a comunicação, é enviada a lista de *arrays* e o *timestamp* da aquisição de dados (em segundos – *Unix timestamp*), depois é recebida a confirmação que os dados foram recebidos no servidor, caso contrário, ou se inicialmente a comunicação com o servidor falhar, é aberto um ficheiro de texto em modo escrita ("w") e os dados digitais processados são gravados de acordo com a seguinte estrutura (Figura 4.17):

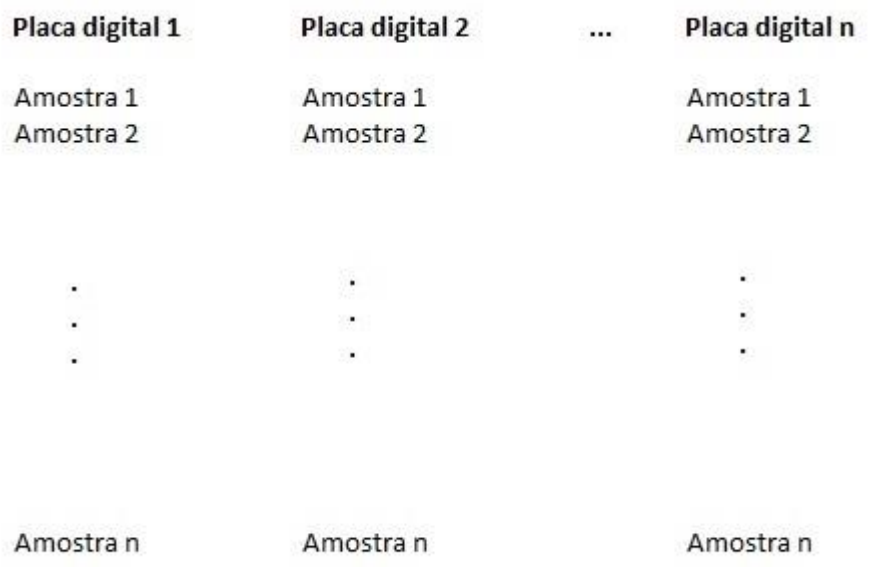


Figura 4.17 – Estrutura de dados digitais gravados em ficheiro texto

A estrutura do ficheiro de dados digitais (Figura 4.17), é composta pelos dados de cada placa digital dispostos verticalmente, e usando o separador "TAB" entre placas, e tal como monstrado, se houver mais do que uma placa digital no sistema, todos os dados digitais são gravados no mesmo ficheiro.

Estes ficheiros de texto acima referidos, se não forem enviados para o servidor devido a falhas de comunicação, são todos gravados no cartão de memória e a estrutura do nome do ficheiro é composta pelos seguintes campos, separados por *underscore* ("_"): Nome de cliente, Local, Tipo e *timestamp* (Ex: "Cliente_Local_D_2017-06-02 15:00:00"), e tal como mostra o exemplo anterior, neste caso, no tipo é inserido o caracter "D", pois trata-se de dados digitais, no caso dos dados analógicos é inserido o caracter "A".

Depois do processamento e gravação de dados digitais, vem a rotina de processamento e gravação de dados analógicos (Figura 4.18 e Figura 4.20), e em comparação com a rotina de processamento e gravação de dados analógicos da Versão Teste (Figura 4.4), existem dois processos novos. O envio de dados para o servidor, tal como demonstrado no processamento e gravação de dados digitais e um algoritmo de calibração para os dados recebidos.

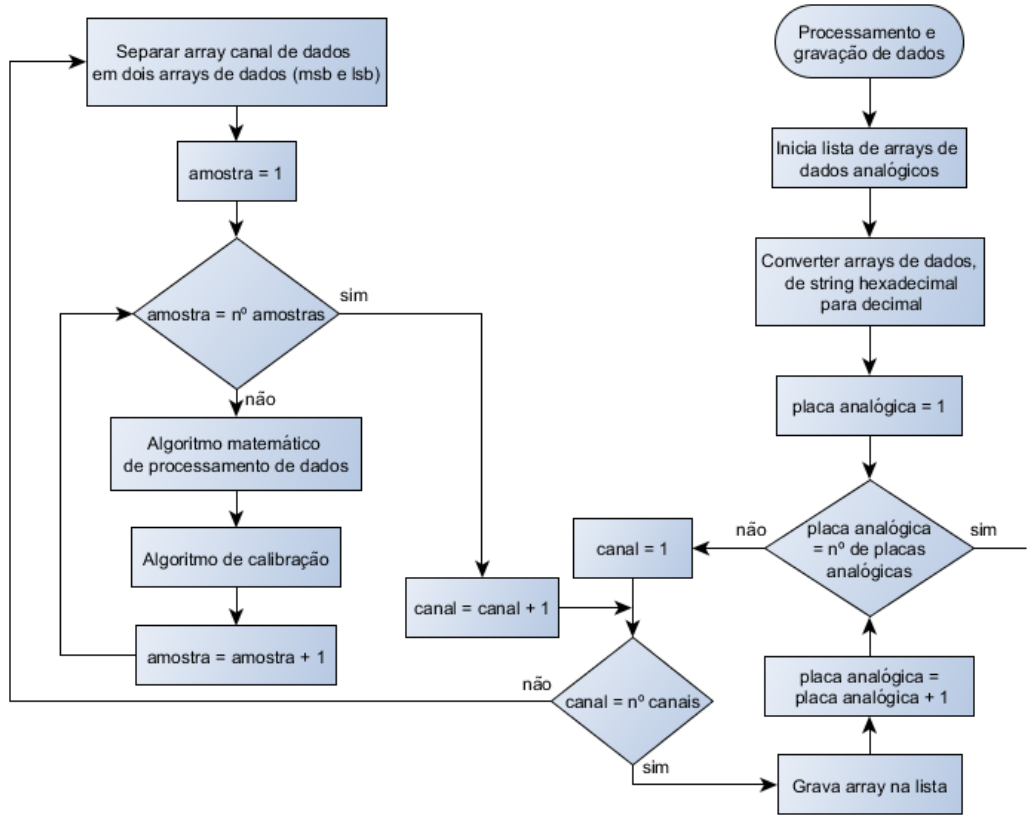


Figura 4.18 – Fluxograma de processamento e gravação de dados analógicos (Parte 1)

Tal como demonstrado na primeira parte do fluxograma de processamento e gravação de dados analógicos, inicialmente é configurada uma lista de *arrays* onde cada *array* corresponde ao total de dados por placa, ou seja, inclui os dados de todos os canais correspondentes a uma placa (Figura 4.19):

```
Lista {
        Array Placa 1 [
                Array Canal 1 [ Amostra 1, ..., Amostra n ], ... , Array Canal N [Amostra 1, ..., Amostra N ]
                        ],

        Array Placa n [
                Array Canal 1 [ Amostra 1, ..., Amostra n ], ... , Array Canal N [Amostra 1, ..., Amostra N ]
                        ]
    }
```

Figura 4.19 – Estrutura da lista de *arrays* de dados analógicos

Depois da lista de *arrays* ser configurada, os dados recebidos são convertidos de *strings* hexadecimais para valores decimais, e posteriormente é realizado um ciclo "*for*" em relação ao número total de placas analógicas, começando com a variável inteira "placa analógica" a 1, depois é iniciado um segundo ciclo "*for*" em relação ao número de canais, também iniciada com a variável inteira "canal" a 1, seguido com a separação do *array* de dados de canal separado em dois *arrays*: o *array* "MSB" e *array* "LSB".

Seguidamente é iniciado o terceiro e último ciclo "*for*" em relação ao número de amostras, iniciado com a variável inteira "amostra" a 1, onde cada amostra corresponde a um valor do *array* "MSB" e outro do *array* "LSB", tal como exemplificado na equação 3. Neste último ciclo "*for*" é realizado um algoritmo de conversão de acordo com o *datasheet* do ADC que integra a placa de sinais analógicos, seguido do algoritmo de calibração, que corresponde à multiplicação do valor final do algoritmo matemático pelo valor de calibração configurado no ficheiro de configurações.

Finalmente é incrementada a variável "amostra" quando este ciclo "*for*" é finalizado com a variável inteira "amostra" é igual ao número total de amostras, é incrementada a variável inteira "canal", e quando esta igualar o número total de canais, é incrementada a variável inteira "placa analógica", e quando esta igualar o número total de placas é finalizado o processamento de dados e iniciado o processo de envio/gravação de dados (Figura 4.20).

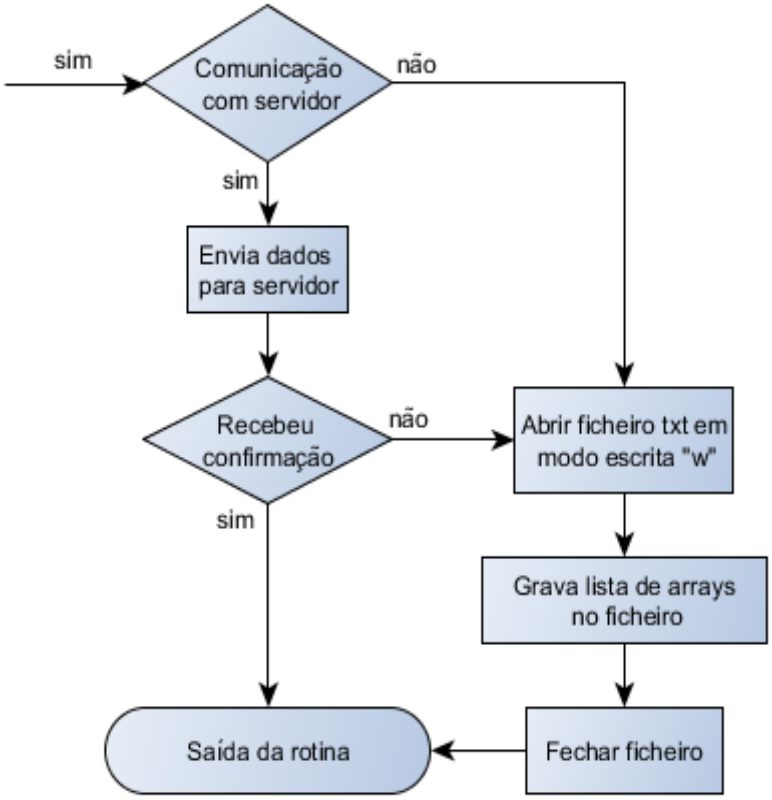


Figura 4.20 – Fluxograma de processamento e gravação de dados analógicos (Parte 2)

Tal como exemplificado na segunda parte do fluxograma de processamento e gravação de dados analógicos (Figura 4.20), este processo é idêntico à parte de envio de dados do fluxograma de processamento e gravação de dados digitais (Figura 4.16), onde inicialmente é realizada a comunicação com o servidor.

Se validada, os dados são enviados, e espera pela confirmação, se receber confirmação, acaba a rotina de processamento e gravação de dados analógicos. Se a comunicação falhar

ou não receber confirmação, é aberto um ficheiro texto em modo escrita "w", e os dados são gravados de acordo com a seguinte estrutura (Figura 4.21):

Placa 1		Placa 2		Placa n	
Canal 1	**Canal n**	**Canal 1**	**Canal n**	**Canal 1**	**Canal n**
Amostra 1	Amostra 1	Amostra 1	Amostra 1	Amostra 1	Amostra 1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
Amostra n	Amostra n	Amostra n	Amostra n	Amostra n	Amostra n

Figura 4.21 – Estrutura de dados analógicos gravados no ficheiro texto

Tal como exemplificado na Figura 4.21, todos os dados são gravados no mesmo ficheiro, onde cada coluna corresponde a um canal, estando estes ordenados tal como as placas, e separados pelo separador "TAB". O nome do ficheiro de dados analógicos tem a mesma estrutura que o nome do ficheiro de dados digitais: "Cliente_Local_A_2017-06-02 15:00:00", mas neste caso o caracter "D" foi substituído pelo caracter "A", por se tratar de dados analógicos.

4.2.7 Monitorização do cartão de memória SD *Card*

A rotina de monitorização do cartão de memória SD *Card* é a última rotina do processo "MAIN", que é realizada depois do desencadeamento da função de atualização do ficheiro de configurações, tal como demonstrado na segunda parte do fluxograma das *threads* (Figura 4.12). Esta rotina tem como objetivo a monitorização o cartão de memória SD *card*, onde são gravados os ficheiros de texto, que contêm os dados processados que não são enviados para o servidor.

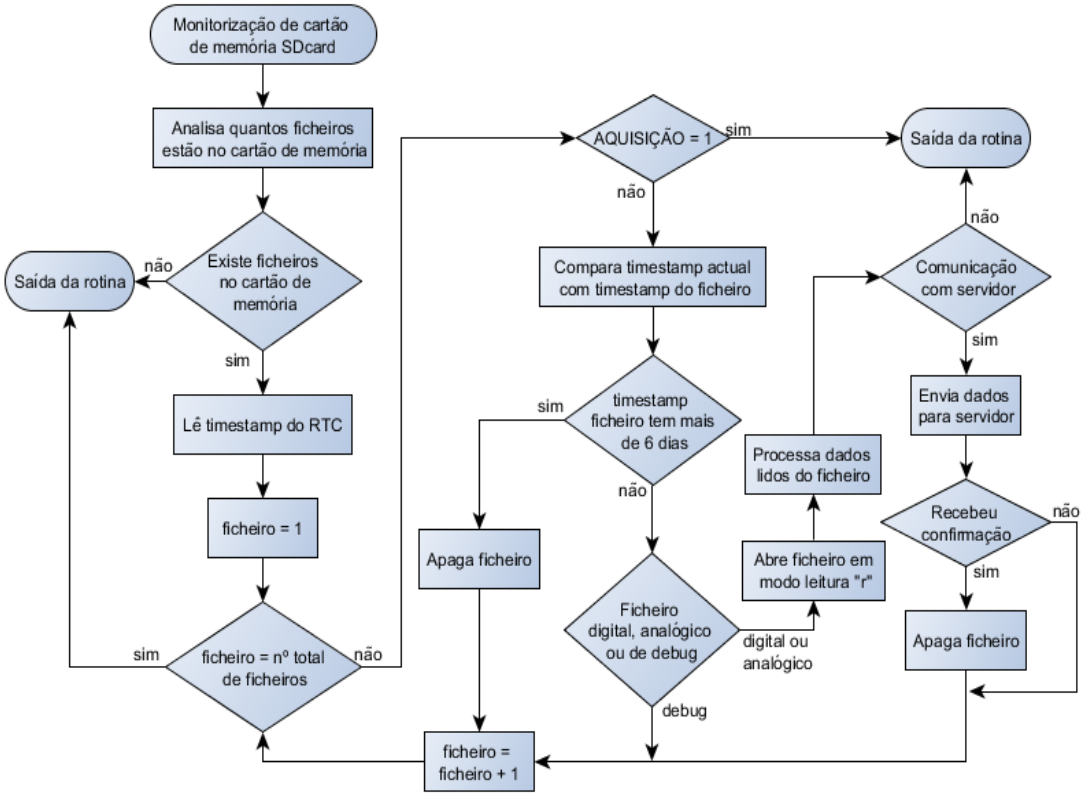


Figura 4.22 – Fluxograma de monitorização do cartão de memória SD *Card*

Tal como demonstrado no fluxograma de monitorização do cartão de memória SD *Card* (Figura 4.22), a primeira ação desta rotina, é verificar quantos ficheiros de texto estão no cartão de memória. Se não existirem ficheiros, a rotina é finalizada automaticamente, se existirem ficheiros é lido o *timestamp* do RTC e é configurada a variável inteira "ficheiro" igual a 1, é iniciado um ciclo "*for*" em relação ao número total de ficheiros.

Este ciclo "*for*" começa por verificar o estado da variável "AQUISIÇÃO", se a variável se encontrar no estado "1" a rotina é finalizada pois está em curso uma nova aquisição. Se a variável se encontrar no estado "0", é porque não existe nenhuma aquisição a decorrer, ou seja, pode decorrer a monitorização dos ficheiros no cartão de memória. Feita esta verificação é comparado o *timestamp* do ficheiro atualmente a ser monitorizado com o *timestamp* lido no início da rotina, e tal como exemplificado na secção anterior, este *timestamp* encontra-se no nome do ficheiro. Se o ficheiro foi criado há mais de 6 dias é automaticamente apagado, se não, dá-se continuidade à sua monitorização, onde é realizada a verificação do tipo de ficheiro.

O ficheiro pode ser do tipo analógico ("A"), digital ("D") ou de "*debug*" ("*debug*"), se for um ficheiro do tipo "*debug*" não é realizada qualquer ação em relação ao ficheiro e é incrementada a variável "ficheiro" para se proceder à monitorização do próximo ficheiro. Se o ficheiro for do tipo analógico ou digital o ficheiro é aberto em modo de leitura ("r"), e é realizado o processamento dos dados de acordo com o seu tipo, depois de processados os dados para uma lista de *arrays*, é realizada a comunicação com o servidor. Se não

houver comunicação a rotina é finalizada, pois se não houver comunicação, não há necessidade de monitorizar mais ficheiros, mas se existir comunicação com o servidor, os dados do ficheiro são enviados. Se existir confirmação, o ficheiro é apagado e a variável "ficheiro" é incrementada e caso contrário, o ficheiro não é apagado e a variável "ficheiro" é incrementada, passando à monitorização do próximo ficheiro. Caso a variável "ficheiro" seja igual ao número de ficheiros, o ciclo "*for*" é finalizado e a rotina termina.

4.3 Versão Específica – GPS

Com a Versão Final (Genérico) finalizada, surgiu a necessidade de desenvolver uma versão específica, esta versão tem como objetivo sincronizar duas *gateway*s em locais distintos através da leitura do *timestamp* de um GPS.

Nas versões anteriores, os sistemas desenvolvidos seguiam da topologia de "mestre-escravo", onde a *gateway* assume o papel de "mestre" e as placas de sinais analógicos e sinais digitais os "escravos", nesta versão específica, esta topologia mantém-se em cada uma das duas *gateway*s, mas ao ter duas *gateway*s, temos dois "mestres". Mas como as *gateway*s têm de comunicar entre si no final de cada recolha e processamento de dados, para o envio dos mesmo, porque apenas uma das *gateway*s envia a totalidade dos dados, uma das *gateway*s é "mestre" e a outra torna-se o seu "escravo" (Figura 4.23).

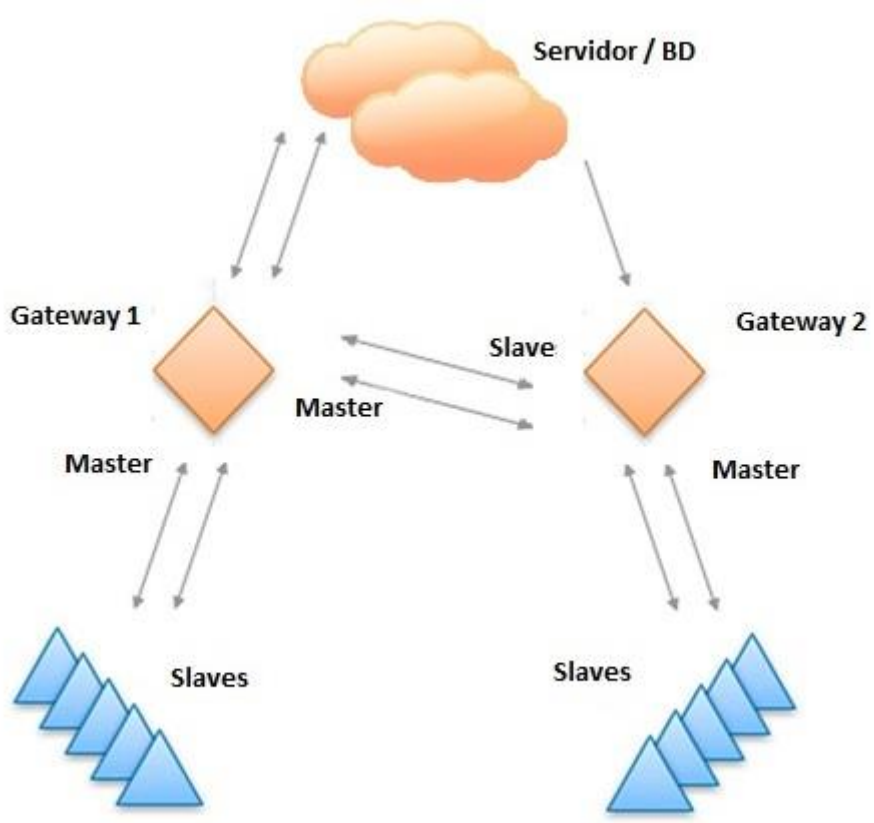


Figura 4.23 – Topologia da Versão Específica – GPS

4.3.1 Fluxograma Geral e Inicializações

Como se trata de uma versão específica, os sistemas onde estão inseridas as *gateway*s também são particulares, onde o "Sistema 1" corresponde à *gateway* 1 ("Mestre"), a três placas de sinais analógicos e a uma placa de sinais digitais, o "Sistema 2" corresponde à *gateway* 2 ("Escravo") e a uma placa de sinais analógicos. Para casos futuros, os *softwares* das duas *gateway*s são dinâmicos, o que permite a alteração do número de placas em qualquer um dos sistemas sem que o *software* sofra qualquer alteração.

Tal como na versão final (Figura 4.6), o fluxograma geral da versão específica (Figura 4.24) é composto pelas rotinas de inicializações, iniciação de variáveis e funções de comunicação com servidor, depois temos as rotinas infinitas *threads*, onde a *thread* "Main" inclui as rotinas de recolha, processamento e gravação de dados, seguido pela função de atualização de ficheiro de configurações e a rotina de monitorização do cartão de memória *SDcard*.

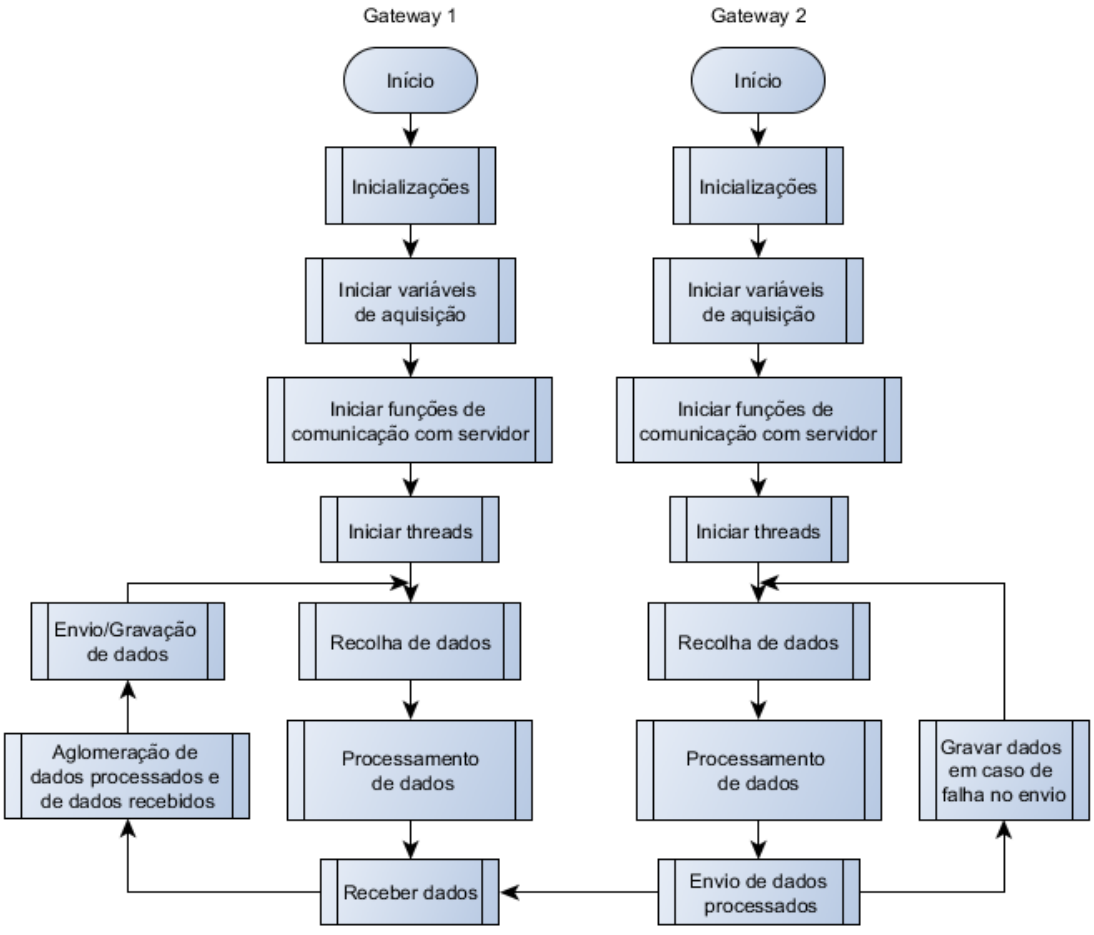


Figura 4.24 – Fluxograma Geral da Versão Específica – GPS

Tal como demonstrado no fluxograma geral da versão específica – GPS (Figura 4.24), temos dois sistemas bastantes similares. As principais diferenças existem depois da rotina de processamento de dados, onde a *gateway* 2 envia os dados processados para a *gateway* 1, esta por sua vez, vai receber esses dados e aglomerar na lista de *arrays* com os dados

processados, que posteriormente serão enviados para o servidor/base de dados ou gravados no cartão de memória em caso de falha de comunicação com o servidor.

As três primeiras rotinas do *software* de cada *gateway* nesta versão, "inicializações", "iniciar variáveis de aquisição" e "iniciar funções de comunicação com o servidor", são totalmente idênticas. Em comparação com as inicializações da Versão final (Figura 4.7), apenas é acrescentada uma nova variável: entrada digital "PPS" (pulsos por segundo) e mantêm-se todas as outras inicializações. A rotina de iniciação de variáveis mantém-se igual à Versão final e Versão inicial (Figura 4.2), tal como a rotina de iniciação de funções de comunicação com o servidor (Figura 4.10).

4.3.2 *Threads*

Tal como na rotina das *threads* da Versão final (Figura 4.11 e Figura 4.12), nesta versão a filosofia das *threads* mantém-se. A *gateway* 1 contém exatamente as mesmas *threads* da Versão final: "TIME_SCHEDULE" (1), "INPUT_DI" (2) e "MAIN" (3) (Figura 4.25), onde cada uma destas *threads* representa um ciclo infinito, e em comparação com os processos da Versão final, os processos "TIME_SCHEDULE" (1) e "INPUT_DI" (2) sofrem algumas alterações, que vão ser abordados nas próximas secções. O processo "MAIN" (3) é totalmente igual ao mesmo processo na Versão final (Figura 4.12).

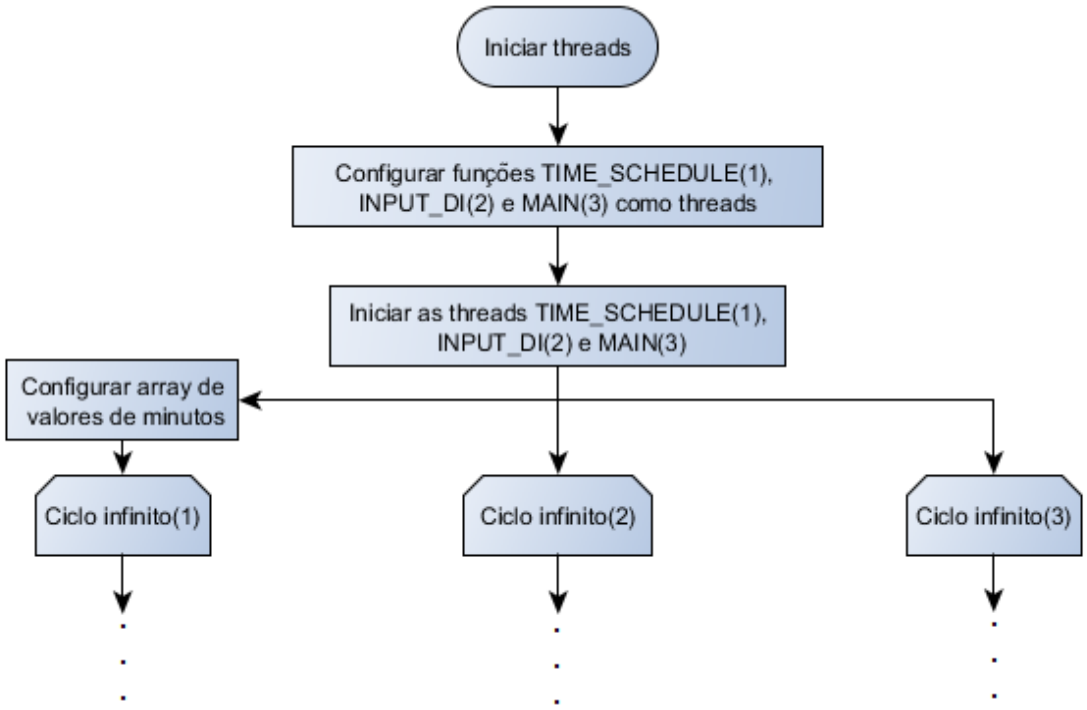


Figura 4.25 – Fluxograma das *threads* da *gateway* 1

Na rotina das *threads* na *gateway* 2 (Figura 4.26) em comparação com a *gateway* 1, há a inclusão de uma nova *thread*, o "SOCKET_SERVER" (4), onde inicialmente é configurado o IP e porta do servidor, e posteriormente iniciado o ciclo infinito do servidor *socket*. Este servidor tem como objetivo receber ordens da *gateway* 1 e este processo

também será abordado nas próximas secções. Quanto aos restantes processos, "TIME_SCHEDULE" (1) e "MAIN" (3), são iguais aos da *gateway* 1 e o processo "INPUT_DI" (2) é ligeiramente diferente do processo da *gateway* 1, que também será abordado nas próximas secções.

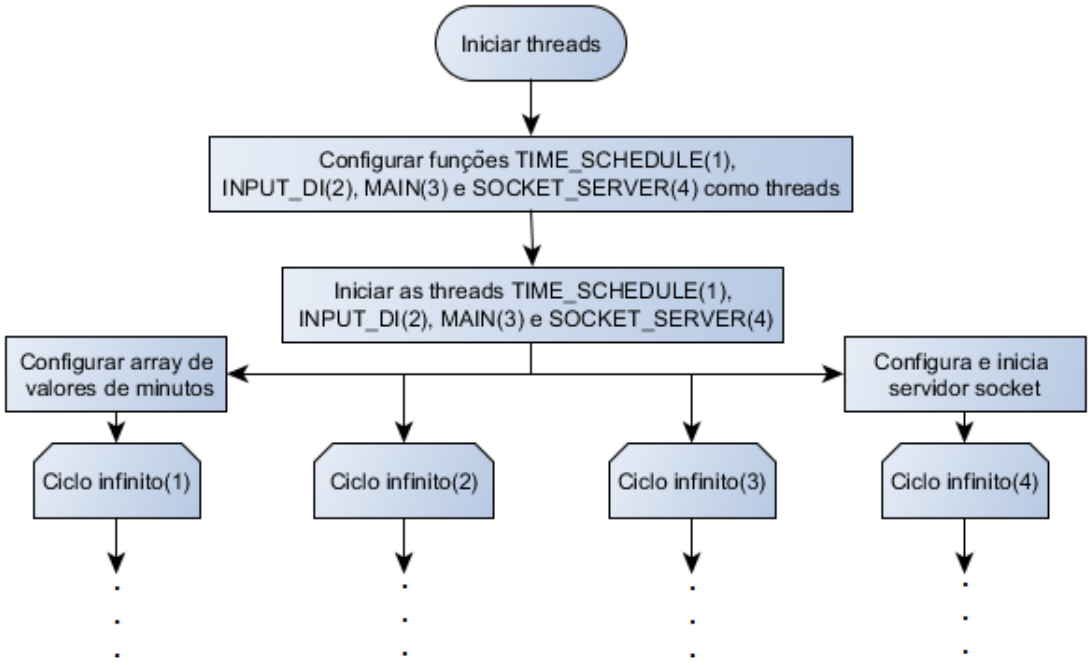


Figura 4.26 – Fluxograma das *threads* da g*ateway* 2

4.3.3 *Threads* "TIME_SCHEDULE" e "MAIN"

Tanto o processo "TIME_SCHEDULE" (1) como o processo "MAIN" (3) são iguais nas duas *gateway*s, sendo o processo "MAIN" (3) igual ao da Versão final (Figura 4.12) onde inicialmente é configurada a variável "AQUISIÇÃO" a "0", depois é esperado um evento dos processos "TIME_SHCEDULE" e "INPUT_DI" ou da função "ORDEM_EXTERNA", que correspondem às variáveis "TIMER", "INPUT" e "ORDEM_EXTERNA", respetivamente. Se for validado o estado de uma destas três variáveis anteriormente referidas a "1", é iniciada a rotina de recolha de dados, seguido da rotina de processamento e envio/gravação dos dados recolhidos. Depois é iniciada a função de atualização do ficheiro de configurações e finalmente a rotina de monitorização do cartão de memória. Já o processo "TIME_SCHEDULE" é ligeiramente diferente da Versão final, pois agora lê-se o *timestamp* a partir do GPS ao invés do relógio interno da BeagleBone Black, que contabilizava o tempo de rotina.

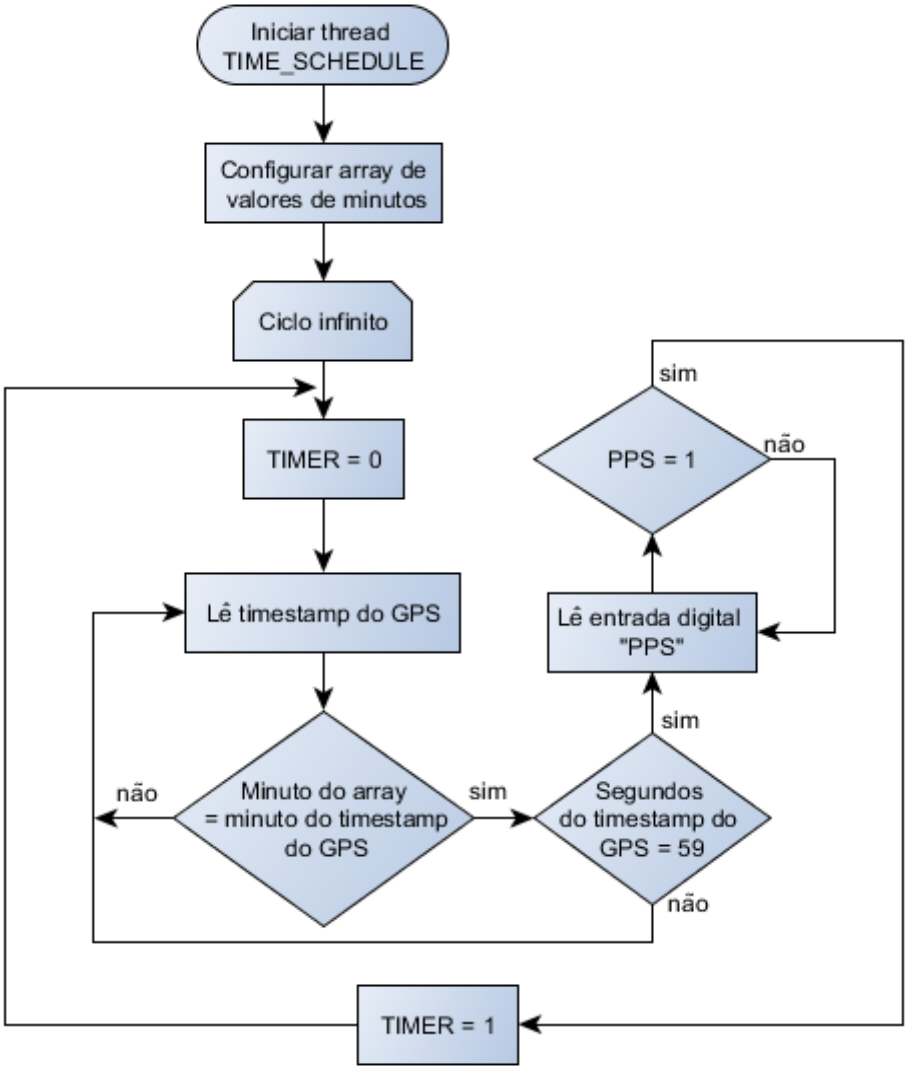


Figura 4.27 – Fluxograma da *thread* "TIME_SCHEDULE" (1)

Tal como demonstrado no fluxograma da *thread* "TIME_SCHEDULE" (1) (Figura 4.27), inicialmente é configurado o *array* de valores de minuto, tal como na Versão final, mas agora é retirado um minuto ao resultado da divisão do valor 60 pelo valor de rotina de aquisições. Anteriormente ao escolher o valor 5 (minutos), para o valor de rotina de aquisições, obtínhamos o *array*:

- {00,05,10,15,20,25,30,35,40,45,50,55}

Agora com o valor 5 obtemos o *array* (o valor 00 é substituído por 59):

- {59,04,09,14,19,24,29,34,39,44,49,54}

Para além desta modificação nos valores no *array* de minutos, a variável inteira "segundos" é alterada de 0 para 59.

Depois de configurado o *array* de valores de minutos, é iniciado o ciclo infinito (1) com a variável "TIMER" no estado "0", e posteriormente é lido o *timestamp* do GPS, de forma diferente do RTC. No *timestamp* é lido através da *interface* SPI, onde se lê o ano, depois

o mês, dia, hora, minuto e segundos e se junta tudo numa *string* com o formato "aa-mm-dd hh:mm:ss". Agora, com o GPS, lê-mos o *timestamp* através da *interface* "RS-232", onde o *timestamp* é lido por completo numa só *string*, tal como é exemplificado na Figura 4.28.

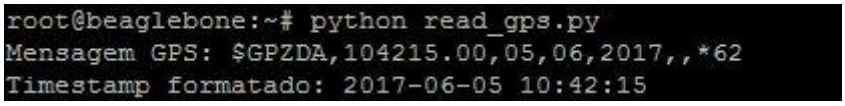


Figura 4.28 – Formato da mensagem proveniente do GPS

4.3.4 *Thread* "INPUT_DI" e função "ORDEM_EXTERNA"

Como referido anteriormente, esta versão apenas possui uma placa de sinais digitais no sistema da *gateway* 1, por isso surgiu a necessidade de criar um novo algoritmo na *thread* "INPUT_DI" e na função de "ORDEM_EXTERNA". Esta que recebe uma ordem de aquisição a partir do servidor, dado que a *gateway* 1 é a "master" do sistema completo, apenas esta recebe esta ordem proveniente do servidor.

Este novo algoritmo, basicamente consiste na *gateway* 1 que recebe uma ordem de aquisição através da função "ORDEM_EXTERNA" ou da entrada digital associada à *thread* "INPUT_DI", lê o *timestamp* atual, soma 4 segundos a esse *timestamp*, envia uma ordem de aquisição para a *gateway* 2, depois esta envia a confirmação para a *gateway* 1, que por sua vez envia o *timestamp* com os 4 segundos somados. As duas *gateway*s irão realizar a aquisição no próximo segundo (próximo PPS) a esse *timestamp*, ou seja, a aquisição é realizada 5 segundos depois de um evento da *thread* "INPUT_DI" ou da função "ORDEM_EXTERNA".

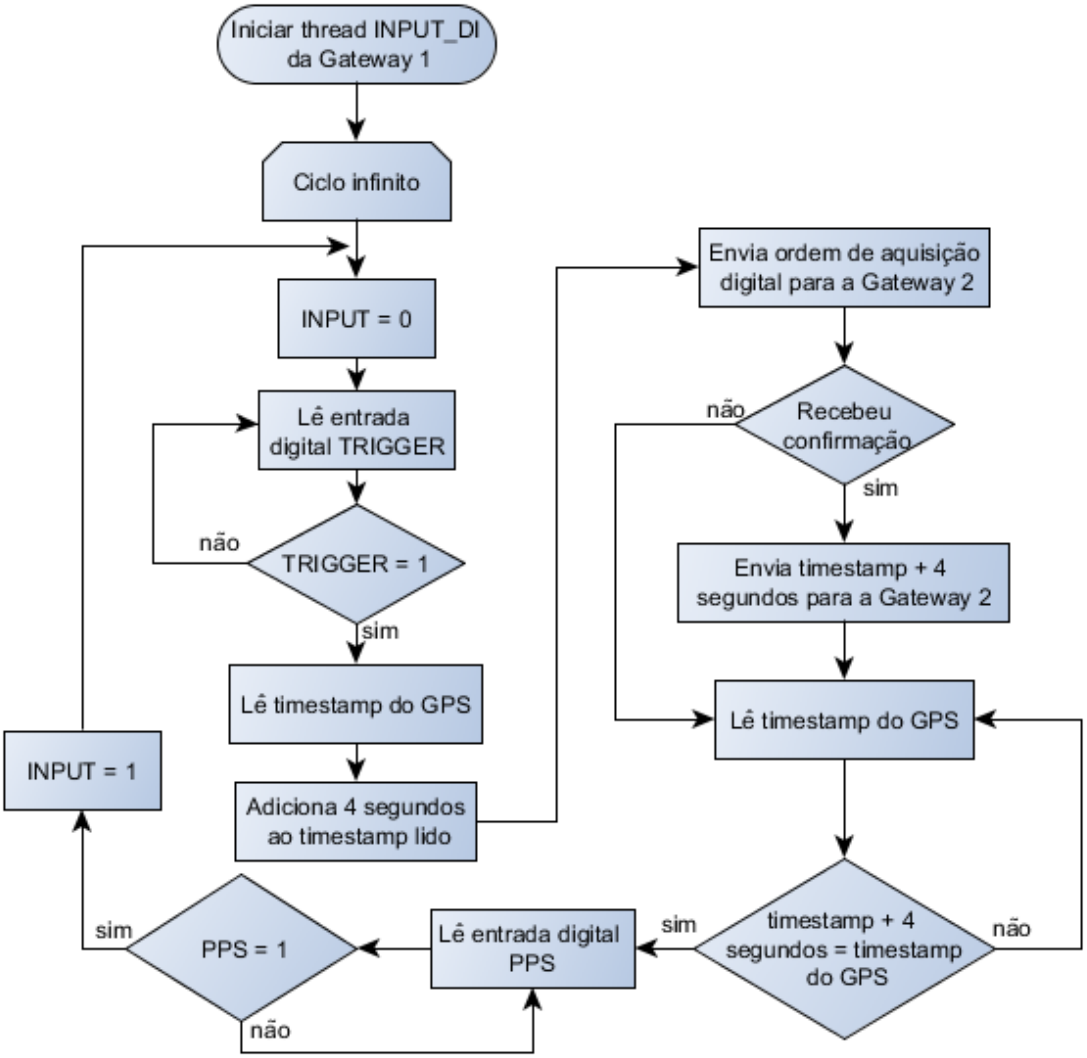


Figura 4.29 – Fluxograma da *thread* "INPUT_DI" da *gateway* 1

Tal como exemplificado no fluxograma da *thread* "INPUT_DI" da *gateway* 1 (Figura 4.29), o processo é iniciado com um ciclo infinito, onde é iniciada a variável "INPUT" no estado "0", lida a entrada digital "TRIGGER" até esta mudar para o estado "1". Quando é detetado este evento, é lido o *timestamp* do GPS e de seguido são adicionados 4 segundos ao *timestamp* lido. Depois é enviada uma ordem de aquisição digital para a *gateway* e esperada a confirmação da mesma. Se não for recebida a confirmação, a aquisição irá ser realizada na mesma, mas depois irá ser descartada na monitorização do *SD Card* (este processo será detalhado no secção referente à monitorização do *SD Card*). Se receber a confirmação, é enviado o *timestamp* já com os 4 segundos somados para a *gateway* 2, De seguida é lido o *timestamp* do GPS até este ser igual ao *timestamp* com os 4 segundos somados, quando esta verificação for validada é lida a entrada digital "PPS" até esta complementar o estado "1". Finalmente a variável "INPUT" muda para o estado "1" e o ciclo é reiniciado com a variável "INPUT" a "0".

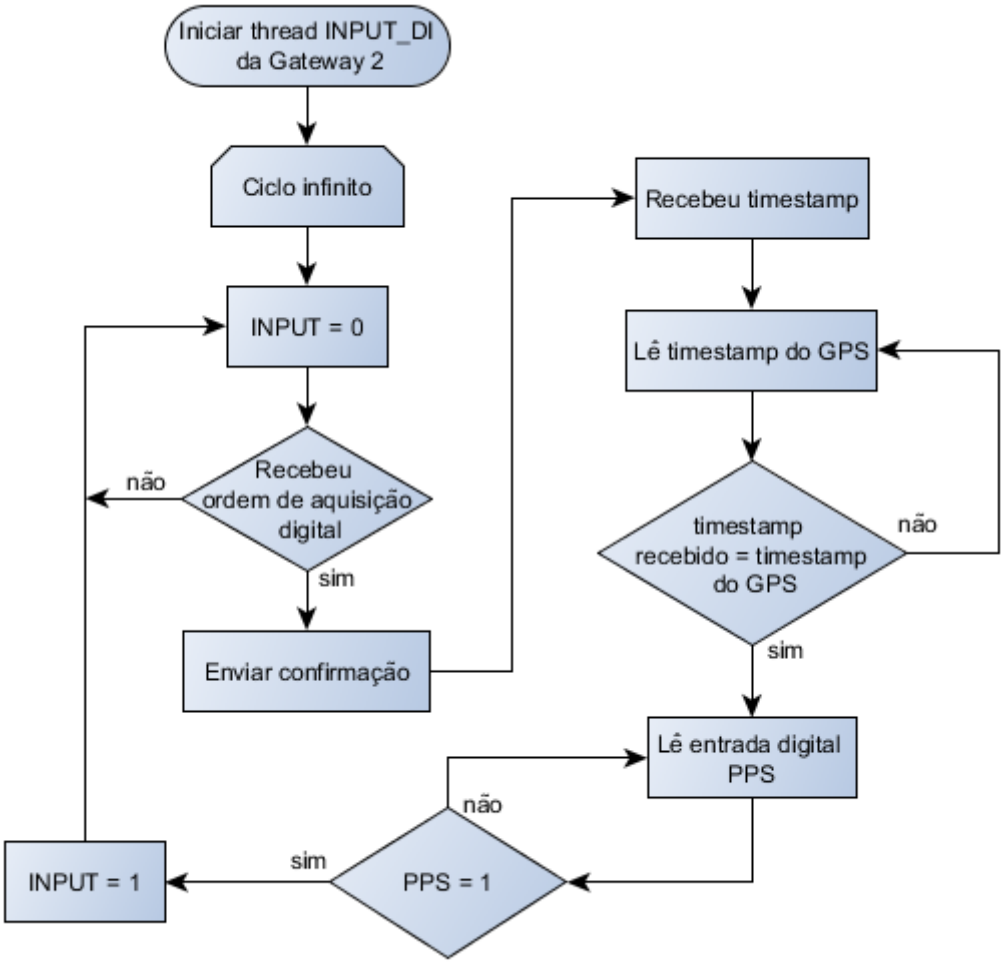


Figura 4.30 – Fluxograma da *thread* "INPUT_DI" da *gateway* 2

Tal como exemplificado no fluxograma da *thread* "INPUT_DI" da *gateway* 2 (Figura 4.30), o ciclo infinito é iniciado com a variável "INPUT" no estado "0", depois é esperada a receção de uma ordem de aquisição digital. Ao ser recebida esta ordem, é enviada uma confirmação, e de seguida é recebido o *timestamp* de aquisição, é lido o *timestamp* do GPS até este ser igual ao *timestamp* recebido. Quando é verificado este evento, é lida a entrada digital "PPS" até esta estar no estado "0", depois de validada esta verificação, a variável "INPUT" muda para o estado "1" e depois o ciclo é reiniciado com esta variável no estado "0".

Relativamente à função "ORDEM_EXTERNA" o algoritmo é exatamente o mesmo que a *thread* "INPUT_DI", a única diferença é que na *thread* "INPUT_DI" é lida a entrada digital "TRIGGER" e na função "ORDEM_EXTERNA" é esperada uma instrução de aquisição proveniente do servidor.

4.3.5 *Thread* "SOCKET_SERVER"

A *thread* "SOCKET_SERVER" é a quarta *thread* do *software* da *gateway* 2, e tal como o nome indica corresponde a um servidor *socket*, que recebe as instruções da *gateway* 1 (cliente), como por exemplo: "data" para o pedido de dados processados, "trig" para a

ordem de aquisição proveniente do servidor, “trdi” para a ordem de aquisição proveniente da placa digital e “part” para pedido de dados gravados num ficheiro texto, que por sua vez é gravado no cartão de memória.

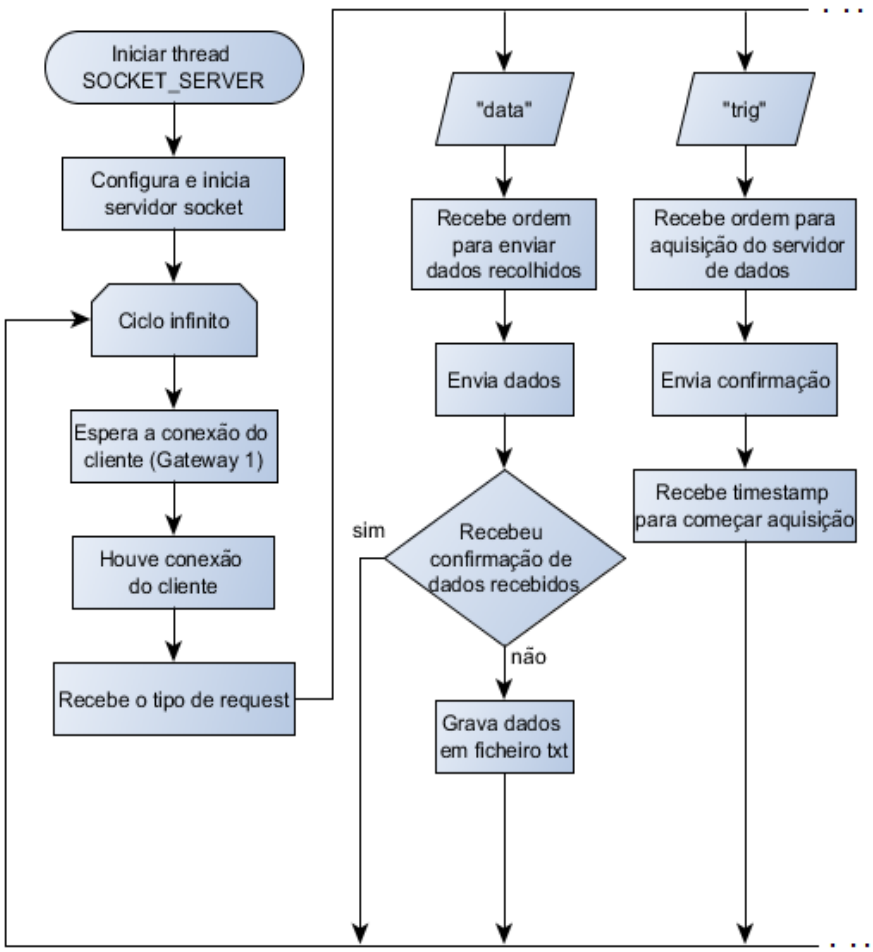


Figura 4.31 – Fluxograma da *thread* “SOCKET_SERVER” da *gateway* 2 (Parte 1)

Tal como exemplificado na parte 1 do fluxograma da *thread* “SOCKET_SERVER” (Figura 4.31), inicialmente é configurado (IP e porta) e iniciado o servidor *socket*, seguido do ciclo infinito para o servidor estar constantemente ativo, pronto a receber instruções do seu cliente (*gateway* 1). Depois é esperada uma conexão do cliente e quando existir conexão do cliente, o servidor *socket* recebe o tipo de instrução que, como referido anteriormente, pode ser: “data”, “trig”, “trdi” e “part”.

A instrução “data” (Figura 4.31) é enviada pela *gateway* 1 depois de esta processar os dados adquiridos, para pedir os dados adquiridos pela *gateway* 2 (dados já processados). Se não existir pedido de dados ou não houver confirmação por parte da *gateway* 1, os dados serão gravados num ficheiro texto.

A instrução “trig” (Figura 4.31) é uma ordem de aquisição por parte da função “ORDEM_EXTERNA” da *gateway* 1, que depois de enviar a confirmação que recebeu

esta instrução, recebe o *timestamp* da aquisição a efetuar (*timestamp* já com os 4 segundos somados, tal como exemplificado no secção anterior).

A instrução "trdi" (Figura 4.32) é uma ordem de aquisição por parte da *thread* "INPUT_DI" da *gateway* 1, e tal como a instrução "trig", depois de enviar a confirmação que recebeu a instrução, recebe o *timestamp* da aquisição a efetuar (*timestamp* já com os 4 segundos somados, tal como exemplificado no secção anterior).

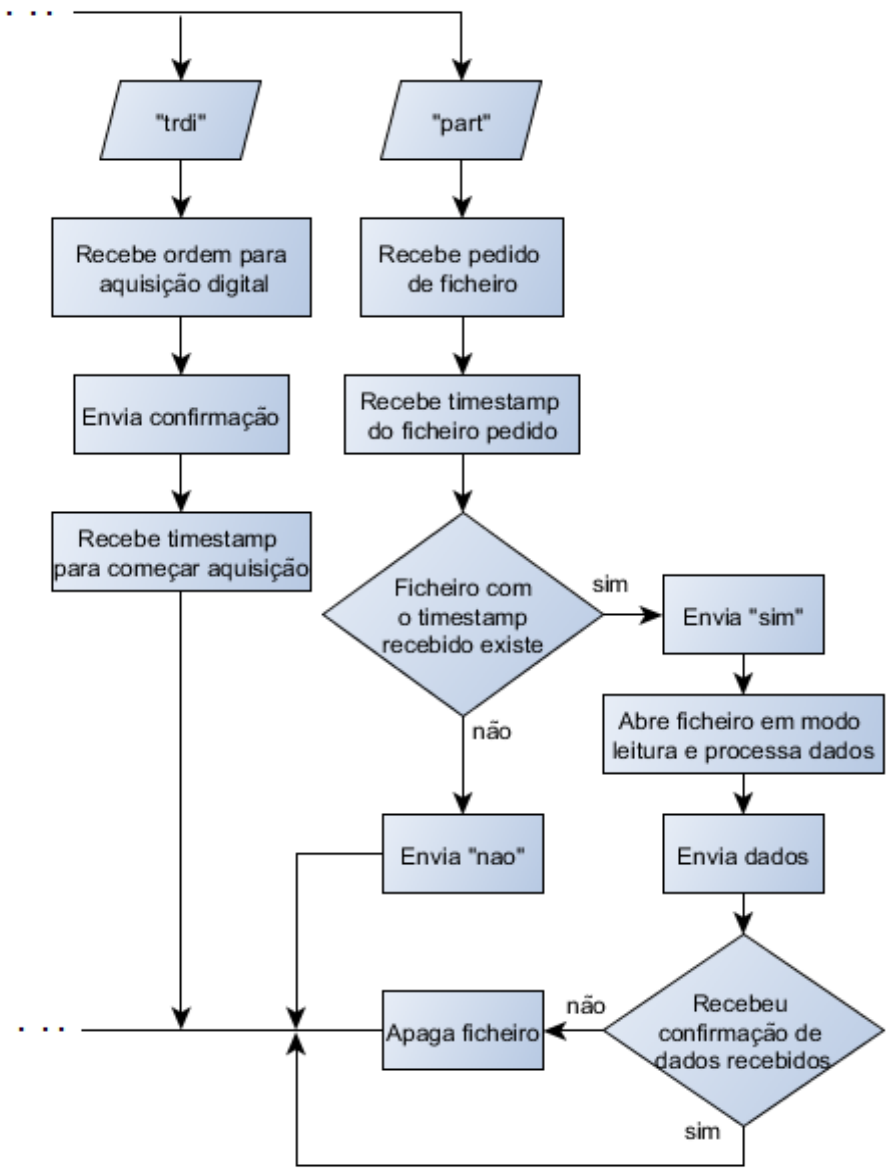


Figura 4.32 – Fluxograma da *thread* "SOCKET_SERVER" da *gateway* 2 (Parte 2)

A instrução "part" corresponde a um pedido de dados da *gateway* 1 gravados num ficheiro texto na *gateway* 2. Esta instrução acontece durante a rotina de monitorização do cartão de memória da *gateway* 1, isto é, acontece quando não existe conexão entre as duas *gateway*s, e os dados são gravados em ficheiros texto em cada uma das *gateway*s. Depois de receber esta instrução, é recebido o *timestamp* do ficheiro pedido e é verificado se o ficheiro existe. Se sim, é enviada a *string* "sim", o ficheiro é aberto em modo de

leitura (“r”) e são processados os dados para uma lista de *arrays* que são então enviados. Se for recebida confirmação dos dados recebidos, o ficheiro é apagado e não for recebida confirmação não é realizada qualquer ação. Se o ficheiro com o *timestamp* recebido não existir, é enviada a *string* “não” para a *gateway* 1 apagar o ficheiro com o *timestamp* corresponde, pois não existiu aquisição do lado da *gateway* 2.

4.3.6 Processamento e gravação de dados

Depois de todas as *threads* configuradas e iniciadas, dentro da *thread* “MAIN” existem várias rotinas (Figura 4.12), são estas: Recolha de dados, Processamento e gravação de dados e Monitorização do cartão de memória. Quanto à primeira rotina, a Recolha de dados, é totalmente idêntica à da Versão Final (Figura 4.13 e 4.14), tanto na *gateway* 1 como na *gateway* 2. Quanto à rotina de Processamento e gravação de dados existem várias alterações.

Quanto à primeira parte do processamento de dados, em comparação com a da Versão Final (Figura 4.18), é totalmente idêntico tanto na *gateway* 1 como na *gateway* 2, desde a conversão de *strings* hexadecimais para valores decimais, a separação do *array* de dados em *array* “MSB” e *array* “LSB”, o algoritmo matemático e o algoritmo de calibração.

A segunda parte do processamento e gravação de dados, começa depois do ciclo “*for*” relativamente ao número de placas analógicas a ser utilizado, e é nesta segunda parte que ambas as *gateway*s têm várias alterações no algoritmo.

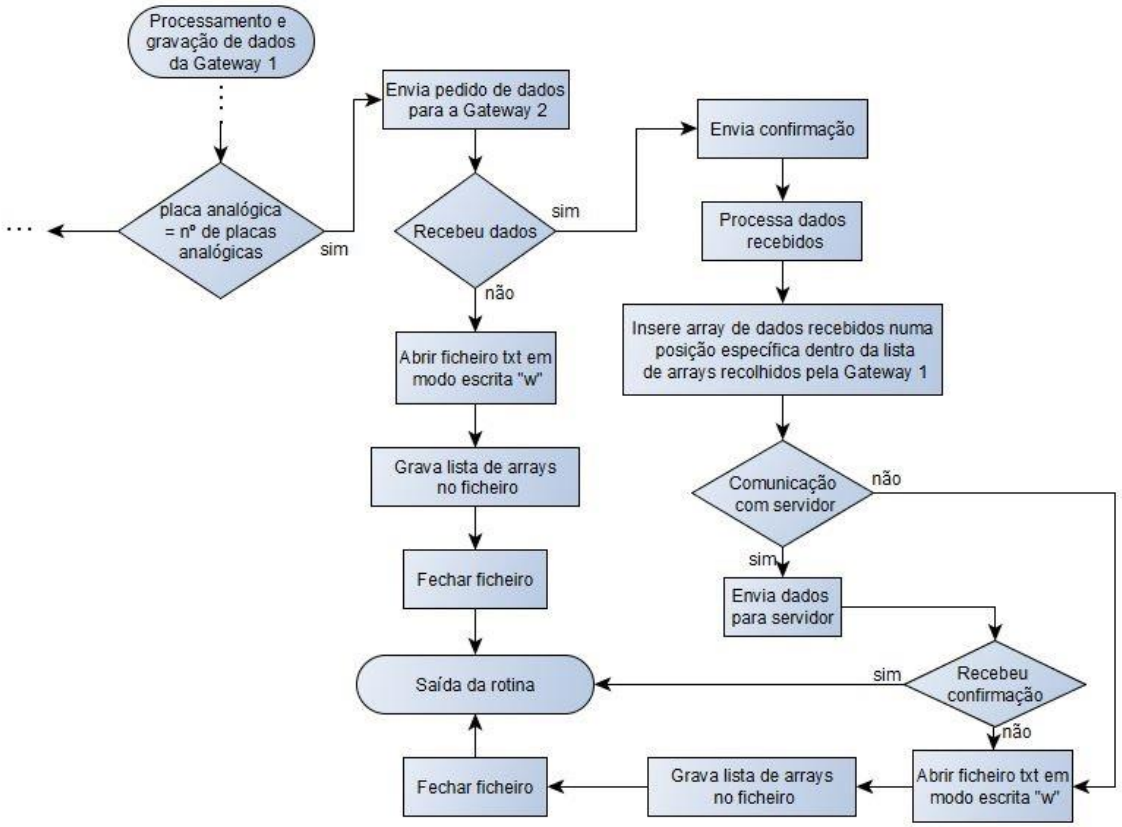


Figura 4.33 – Fluxograma de processamento e gravação de dados da *gateway* 1

Tal como demonstrado no fluxograma do processamento e gravação de dados da *gateway* 1 (Figura 4.33), com o ciclo "*for*" finalizado relativamente ao número de placas analógicas, começa a segunda parte do processamento e gravação de dados, começando por enviar um pedido de dados para a *gateway* 2. Se os dados forem recebidos é enviada uma confirmação, seguido do processamento dos dados recebidos que complementa a organização dos dados recebidos num *array* de *arrays*, que posteriormente é inserido numa posição específica dentro da lista de *arrays* de dados recolhidos e processados pela *gateway* 1 que depois serão enviados para o servidor. Caso não haja confirmação da parte do servidor, os dados são gravados num ficheiro texto, onde o nome do ficheiro indica a totalidade dos dados (Dados da *gateway* 1 e *gateway* 2). Se não forem recebidos dados da *gateway* 2 após o pedido, os dados recolhidos e processados pela *gateway* 1 não são enviados para o servidor, mas sim gravados diretamente num ficheiro texto, onde o nome deste ficheiro indica a parcialidade dos dados (Dados apenas da *gateway* 1).

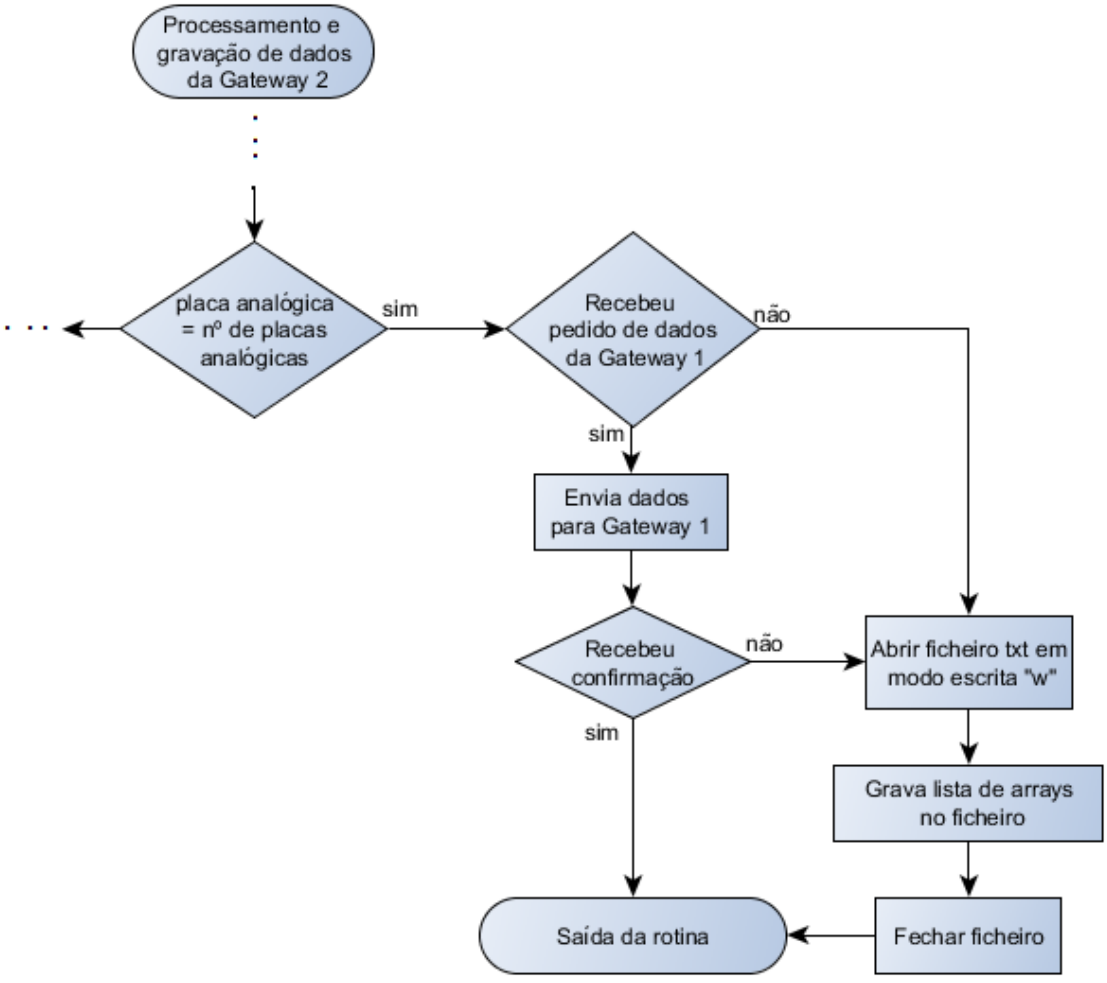


Figura 4.34 – Processamento e gravação de dados da g*ateway* 2

Na rotina de processamento e gravação de dados da *gateway* 2, a primeira parte desta rotina também é idêntica à primeira parte da rotina da Versão Final (Figura 4.18), e tal como demonstrado no fluxograma do processamento e gravação de dados da *gateway* 2

(Figura 4.34), a segunda parte desta rotina começa com a finalização do ciclo "*for*" relativamente ao número de placas analógicas, e espera por um pedido de dados por parte da *gateway* 1. Se for recebido o pedido, os dados são enviados e é esperada uma confirmação. Se esta for recebida, a rotina acaba, se não houver pedido de dados ou for recebida a confirmação, os dados são gravados num ficheiro texto.

4.3.7 Monitorização do cartão de memória *SD Card*

A monitorização do cartão de memória SD *Card* da *gateway* 1 é bastante idêntica à rotina de monitorização do cartão de memória da Versão Final (Figura 4.22). Apenas existe uma diferença devido à inclusão de um novo tipo de ficheiro, o ficheiro parcial que contém apenas os dados de sinais analógicos recolhidos e processados pela *gateway* 1, em vez da totalidade dos dados de sinais analógicos recolhidos e processados pelas duas *gateway*s. Isto acontece quando existem falhas de comunicação entre as duas *gateway*s, tal como explicado anteriormente. A rotina de monitorização do cartão de memória da *gateway* 2 apenas contém uma rotina para apagar ficheiros, pois esta não envia dados para o servidor.

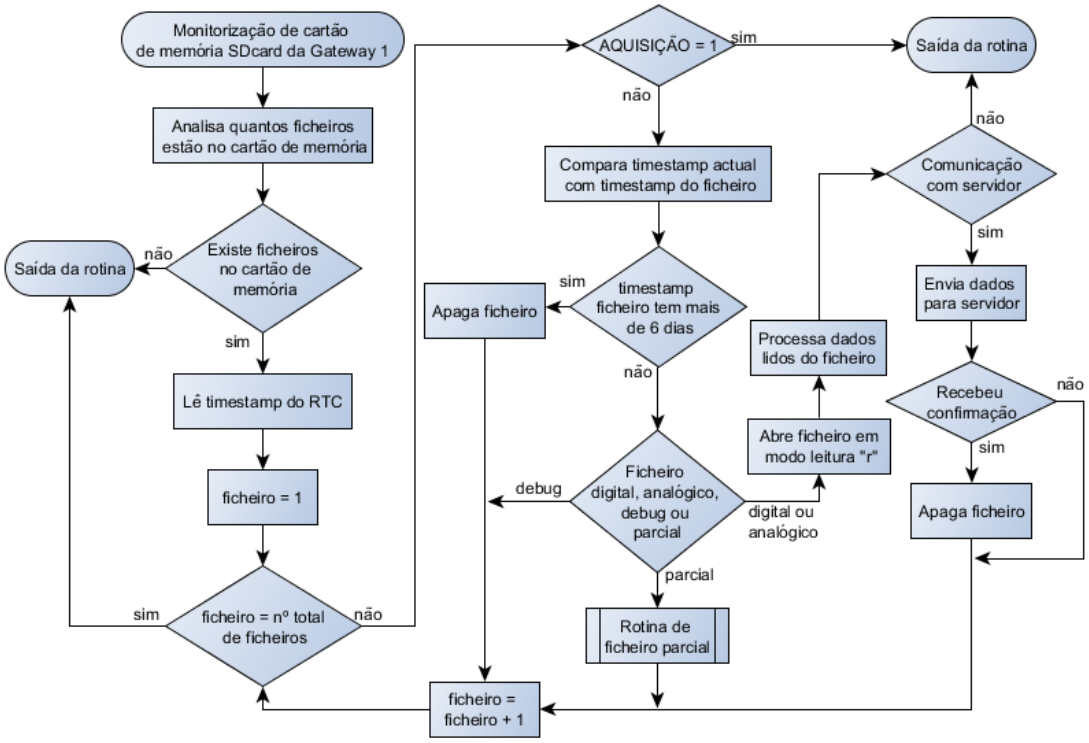


Figura 4.35 – Fluxograma de monitorização do cartão de memória da g*ateway* 1

Tal como exemplificado no fluxograma de monitorização do cartão de memória da *gateway* 1 (Figura 4.35), a rotina é iniciada pela deteção do número de ficheiros no cartão de memória. Se não existirem ficheiros, automaticamente é fechada a rotina. Se existirem ficheiros, é lido o *timestamp* do GPS e iniciado um ciclo "*for*" em relação ao número total de ficheiros. No início deste ciclo é sempre verificada a variável "AQUISIÇÃO". Se estiver no estado "1" a rotina é fechada, se não, continua e é comparado o *timestamp* atual com o *timestamp* do ficheiro a ser analisado. Se tiver mais de 6 dias o ficheiro é apagado

e incrementada a variável "ficheiro", se não é verificado o tipo de ficheiro. Se for um ficheiro de "debug", não é executada qualquer ação e a variável "ficheiro" é incrementada, se for um ficheiro do tipo "analógico" (ficheiros de tipo "analógico" contêm a totalidade de dados recolhidos e processados pelas duas *gateway*s) ou "digital", o ficheiro é aberto em modo de leitura "r", os dados são processados e se não existir comunicação com o servidor a rotina é fechada. Se existir comunicação com o servidor, os dados são enviados e é esperada uma confirmação. Se esta for recebida o ficheiro é apagado e a variável "ficheiro" é incrementada, se não for recebida confirmação, o ficheiro não é apagado e é incrementada a variável "ficheiro". Se o ficheiro for do tipo "parcial" é executada a rotina de ficheiro parcial (Figura 4.36), e no final desta rotina a variável "ficheiro" é incrementada.

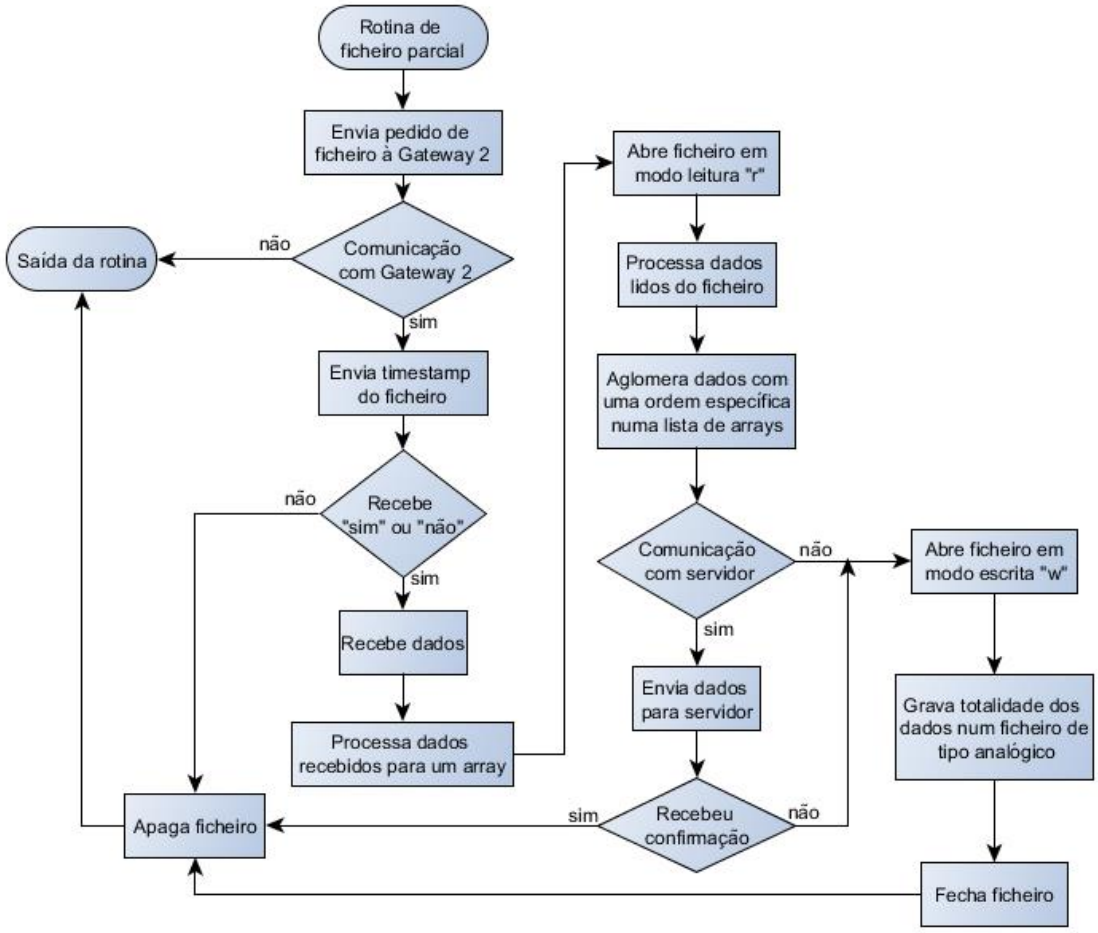


Figura 4.36 – Fluxograma da rotina de ficheiros parciais (g*ateway* 1)

A rotina de ficheiro parcial é iniciada pelo envio de um pedido de ficheiro à *gateway* 2 (*string* "part", ver fluxograma da *thread* "SOCKET_SERVER" (Figura 4.32)). Se não existir comunicação a rotina é fechada, se existir é enviado o *timestamp* do ficheiro parcial a ser analisado e é validada a existência do ficheiro na *gateway* 2. Se esta verificar que não existe envia "não" e o ficheiro parcial é apagado. Se esta verificar que o ficheiro existe envia "sim" e posteriormente envia os dados. Ao receber estes dados a rotina de ficheiro parcial processa os dados para um *array*, abre o ficheiro parcial em modo de

leitura "r" e processa os dados para uma lista de *arrays*. Os dados lidos e os dados recebidos são aglomerados na mesma lista de *arrays* numa ordem específica e é verificada a comunicação com o servidor. Se esta for validada os dados são enviados e se for recebida confirmação o ficheiro parcial é apagado e a rotina é finalizada. Se não existir comunicação ou não for recebida confirmação, a lista de *arrays* é gravada num ficheiro texto de tipo "analógico" (totalidade dos dados), o ficheiro parcial é apagado e a rotina é finalizada.

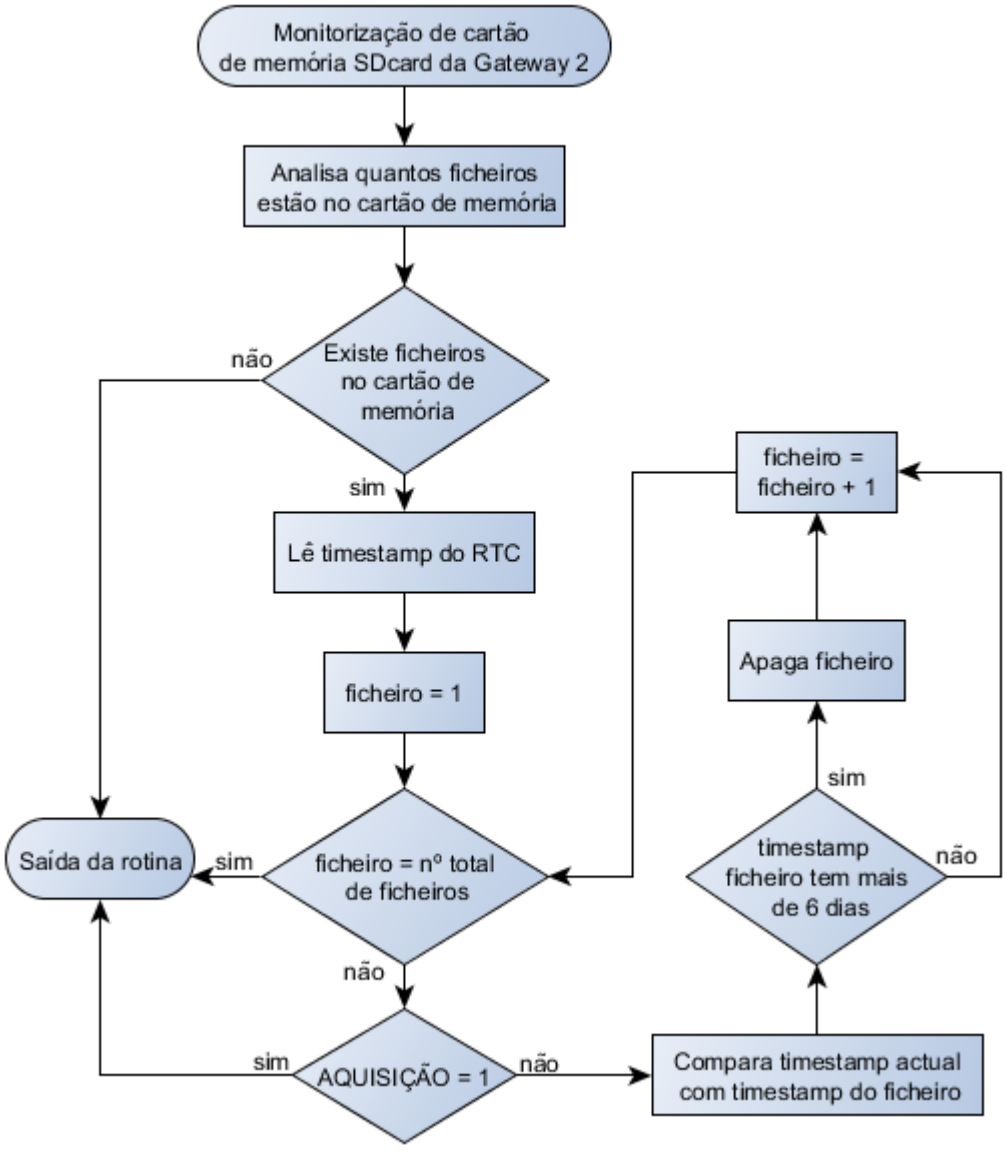


Figura 4.37 – Fluxograma de monitorização do cartão de memória da *gateway* 2

Na rotina de monitorização do cartão de memória da *gateway* 2 (Figura 4.37) apenas é executada uma rotina para apagar ficheiros com mais de 6 dias. A rotina é iniciada pela verificação do número de ficheiros presentes no cartão de memória. Se não existir nenhum a rotina é fechada, se existir, é lido o *timestamp* do GPS e iniciado um ciclo "*for*" em relação ao número total de ficheiros. Quando este ciclo é finalizado a rotina termina

e no início deste ciclo é verificado o estado da variável "AQUISIÇÃO". Se esta estiver no estado "1" a rotina acaba, se não, é comparado o *timestamp* atual com o *timestamp* do ficheiro a ser analisado. Se este tiver mais de 6 dias, o ficheiro é apagado e a variável "ficheiro" incrementada, se não, não é executada qualquer ação e a variável "ficheiro" é incrementada para analisar o próximo ficheiro.

5 Testes do Sistema e Implementação

Este capítulo pretende validar as funcionalidades do sistema desenvolvido através da elaboração de dois ensaios experimentais, um ensaio em ambiente laboratorial e um ensaio industrial. No ensaio laboratorial foram recolhidos e processados os dados de quatro placas analógicas. As placas analógicas adquiriram sinais provenientes de um sensor de tensão, estando a entrada deste sensor ligada à rede elétrica (230V/50Hz) e uma placa digital. O segundo ensaio foi realizado em ambiente industrial, onde foram recolhidos e processados os dados de duas placas analógicas e uma digital e onde as duas placas analógicas adquiriram sinais provenientes de um transformador elétrico. Neste capítulo é ainda apresentada uma abordagem direcionada para a perspetiva da implementação industrial do sistema de recolha e processamento de dados desenvolvido.

5.1 Teste em ambiente laboratorial

5.1.1 Configuração do teste laboratorial

A configuração do ensaio laboratorial segue a disposição do esquemático representado na Figura 5.1. Neste ensaio laboratorial foi utilizada a Versão Final do sistema, este que era composto pela *gateway*, 4 Placas Analógicas e 1 Placa Digital todas alimentadas por 24 VDC provenientes de uma fonte de alimentação (230 VAC/ 24 VDC) ligado à rede elétrica (230 V / 50 Hz). As placas analógicas adquiriram o sinal analógico proveniente de dois sensores de tensão AC (estes desenvolvidos pela Enging) separados em dois barramentos e com a placa digital sem qualquer componente digital ligada aos seus terminais. A *gateway* recolheu e processou os dados recolhidos pelas várias placas através do barramento RS-485, que posteriormente foram armazenados no cartão de memória e recolhidos para o computador através de acesso remoto (*Software* WinSCP, Protocolo SFTP) e sendo finalmente analisados num *software* próprio da empresa Enging.

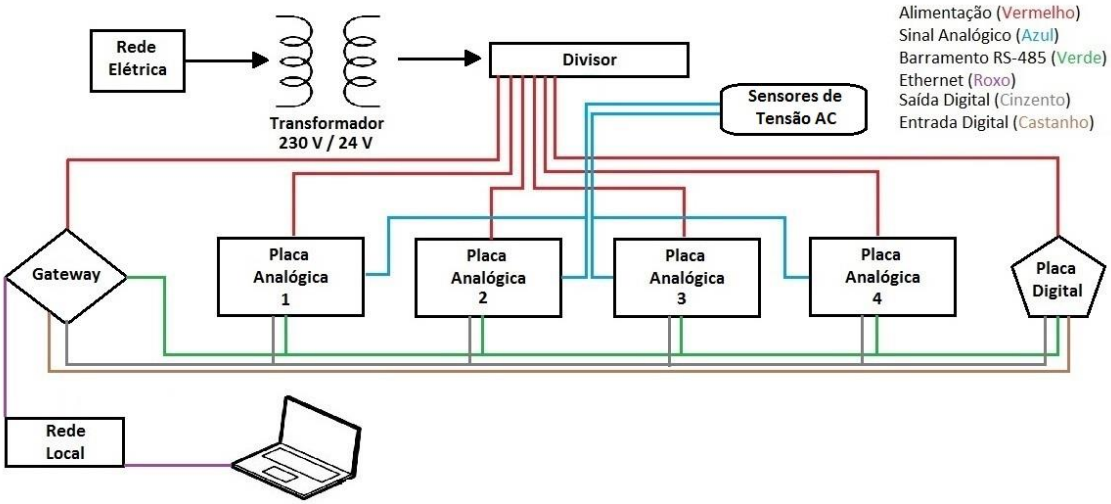
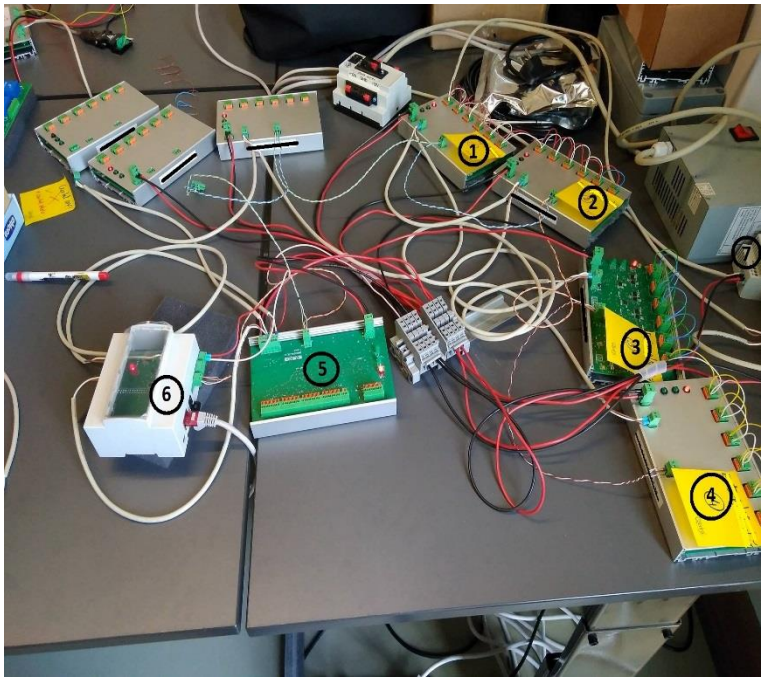


Figura 5.1 – Esquemático do ensaio laboratorial realizado com o sistema da Versão Final

A Figura 5.2 apresenta uma fotografia tirada durante o ensaio laboratorial realizado no laboratório da empresa Enging, onde se pode observar todos os elementos mencionados anteriormente, exceto os sensores de tensão de onde provém o sinal analógico adquirido pelas placas analógicas.



Legenda:

1 – Placa Analógica nº1

2 – Placa Analógica nº2

3 – Placa Analógica nº3

4 – Placa Analógica nº4

5 – Placa Digital

6 – *Gateway*

7 – Fonte de alimentação 230 VAC / 24 VDC

Figura 5.2 – Ensaio laboratorial realizado com o sistema da Versão Final

As placas analógicas realizaram uma aquisição do sinal analógico proveniente do sinal de saída do sensor de tensão AC, cujo sinal de entrada era a tensão da rede elétrica (230V/50Hz). As placas analógicas foram configuradas para uma aquisição de 20000 pontos, a uma frequência de aquisição de 20 kHz, o que resulta no total de 1 segundo de aquisição (Figura 5.3). Também foram configuradas para usar a totalidade dos canais, ou seja, 6 canais, como podemos ver pela Figura 5.2.

De seguida são apresentados os resultados obtidos neste ensaio laboratorial, como também uma breve discussão sobre os mesmos.

5.1.2 Discussão de resultados

Esta secção tem como objetivo apresentar e discutir os resultados obtidos durante o ensaio laboratorial. Tal como referido anteriormente, o ficheiro texto que contém os dados recolhidos e processados é retirado do cartão de memória da *gateway* através do *software* WinSCP para o computador pessoal, isto porque neste ensaio não houve comunicação com o servidor/base de dados.

Os dados são inseridos num *software* próprio da empresa Enging, mas também podem ser analisados numa folha Excel. A Figura 5.3 representa o gráfico completo de todos os dados recolhidos e processados no ensaio laboratorial. O sinal resultante não contém o ganho do ADC utilizado na placa analógica, o que resulta num sinal AC, com valor mínimo de cerca de -4,4 V e valor máximo de cerca de 4.4 V.

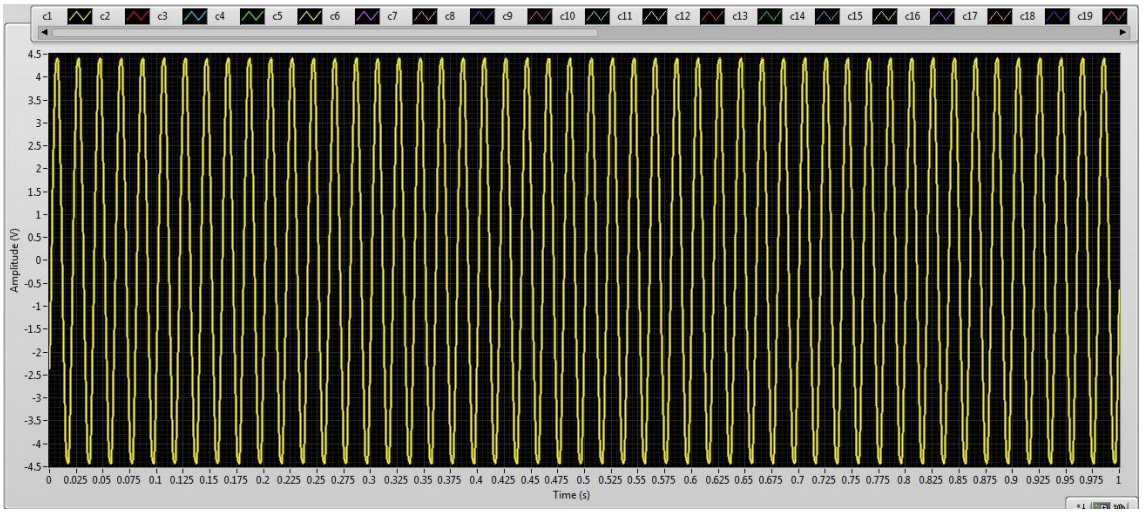


Figura 5.3 – Gráfico de todos os dados recolhidos e processados pela *gateway*

O gráfico da Figura 5.4 representa os primeiros dois períodos do sinal recolhido no teste laboratorial (Figura 5.2), onde é possível observar com mais pormenor o sinal adquirido. Estes dois primeiros períodos correspondem aos primeiros 0.04 segundos da aquisição. Como podemos observar, o sinal da rede elétrica não é uma sinusoide "perfeita". Estas imperfeições na onda sinusoidal pode ter origem em vários fatores, desde ruído no sensor de tensão, bem como nos ADCs das placas analógicas, interferência nos condutores da montagem, e pela potência pedida à rede elétrica no momento do ensaio (distorção).

Figura 5.4 – Primeiros 2 períodos do sinal recolhido no teste laboratorial.

Como é possível observar pela Figura 5.5, o valor máximo do sinal adquirido é de 4.42 VAC e 4.40 VAC. Esta figura corresponde a um *zoom* realizado no valor máximo do primeiro período do sinal adquirido (Figura 5.3). Nesta imagem já é possível observar a diferença entre os dois barramentos do sinal analógico (linha azul na Figura 5.1) e os vários sinais, onde cada cor corresponde a um canal, sendo este gráfico composto por 24 sinais (4 placas com 6 canais por placa). Os primeiros 12 canais pertencem às placas 1 e 2, que no gráfico corresponde aos canais que apresentam um valor de tensão máxima menor (4.40 VAC), e os outros 12 canais pertencem às placas 3 e 4 que apresentam um

valor de tensão máxima maior (4.42 VAC), a discrepância entre os dois barramentos é de cerca de 0.02 V, que pode ter origem em várias causas, sendo que a causa mais provável é a tolerância das resistências de potência utilizadas nos diversos sensores de tensão.

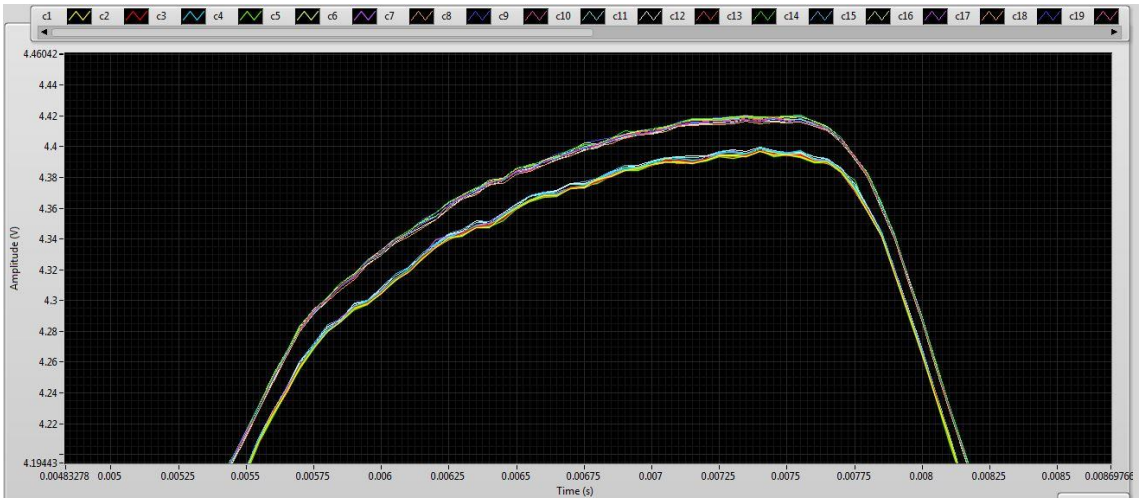


Figura 5.5 – Valor máximo do sinal adquirido

A Figura 5.6 corresponde a um *zoom* realizado à Figura 5.5, onde podemos ver os vários sinais adquiridos, sendo que cada sinal corresponde a um canal de uma placa analógica, dando um total de 24 canais. Como podemos verificar, há pequenas discrepâncias entres os vários sinais na ordem das dezenas de mV, sendo a causa mais provável para estas diferenças a tolerância do ganho dos canais analógicos de cada placa analógica.

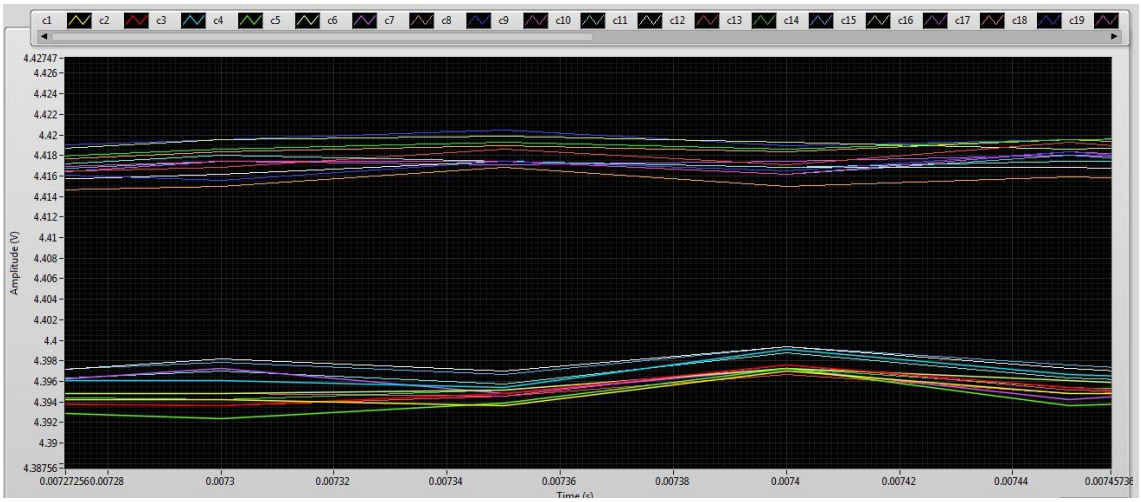


Figura 5.6 – Valor máximo do sinal adquirido (mais aproximado)

Observando a Figura 5.7, que representa o ficheiro obtido da aquisição digital, onde se verifica que todos os canais se encontram no estado "0", dado que neste teste não foi realizada qualquer ligação em nenhum dos 32 canais da placa digital, como é possível verificar pela Figura 5.2.

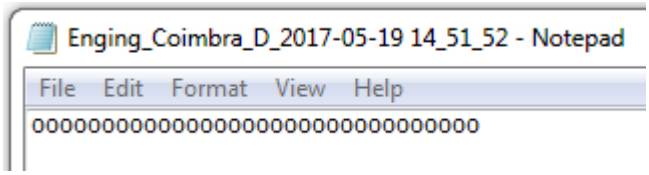


Figura 5.7 – Dados recolhidos da placa digital

Este ensaio realizado em ambiente laboratorial permitiu confirmar o funcionamento integrado do sistema da Versão Final. Testou-se a recolha e processamento de dados adquiridos pelas várias placas no sistema, que continha 4 placas analógicas e 1 placa digital, que adquiriram um sinal proveniente de dois sensores de tensão ligados à rede elétrica. Sendo demonstradas as várias funcionalidades do sistema, desde comunicação, configurações e *software*.

5.2 Teste em ambiente industrial

5.2.1 Instalação industrial

Depois de vários testes em ambiente laboratorial, aperfeiçoamentos ao sistema e correções de pequenos *bugs* de *software* foi realizado um ensaio em ambiente industrial. Neste teste, foram adquiridas as correntes de entrada e saída, e as tensões de entrada e saída de um transformador trifásico em funcionamento.

O sistema foi composto por 1 *gateway*, 2 placas analógicas, 1 placa digital, 1 *router*, 1 transformador e vários sensores de tensão e de corrente. Tal como podemos verificar pelo esquemático do ensaio laboratorial realizado com o sistema da Versão Final (Figura 5.8), todas as placas envolvidas no sistema, bem como o *router* utilizado são alimentados a 24 V (linha vermelha).

A *gateway* e as placas analógicas e a placa digital comunicam através de um barramento RS-485 (linha verde), e entre as mesmas existe uma saída digital (linha cinza) com origem na *gateway*, e uma entrada digital entre a *gateway* e a placa digital, com origem nesta. A *gateway* é ligada a um router através de um cabo Ethernet, que comunica com o servidor/base de dados, e por fim, os sinais analógicos das correntes e tensões de entrada e saída do transformador trifásico, antes de chegarem às placas analógicas, passam por sensores de corrente e tensão, estes que têm como objetivo diminuir as gamas de valores máximos e mínimos.

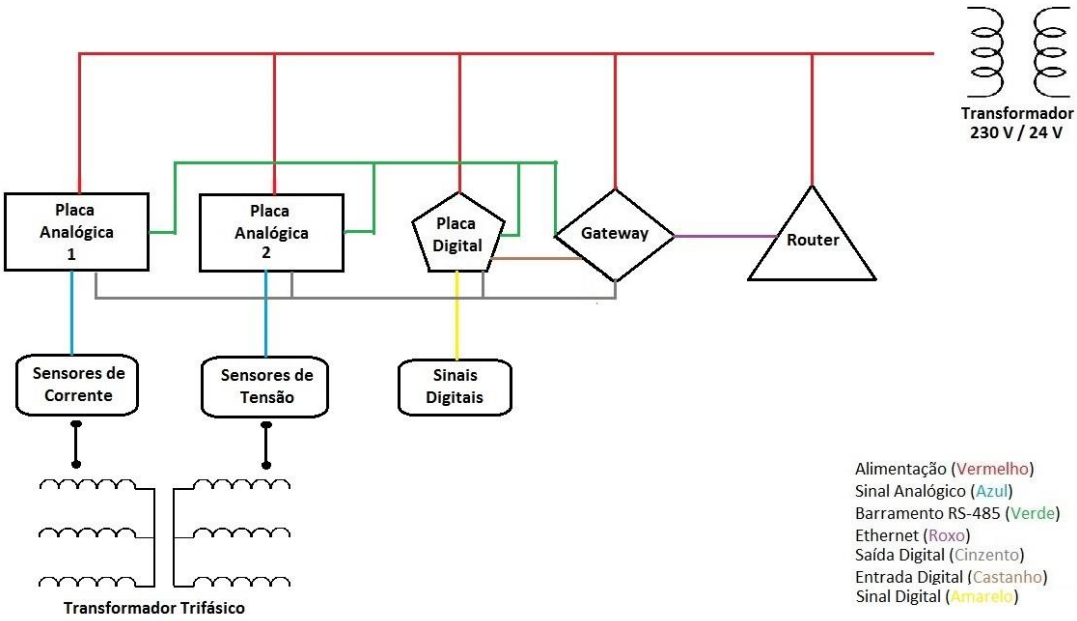
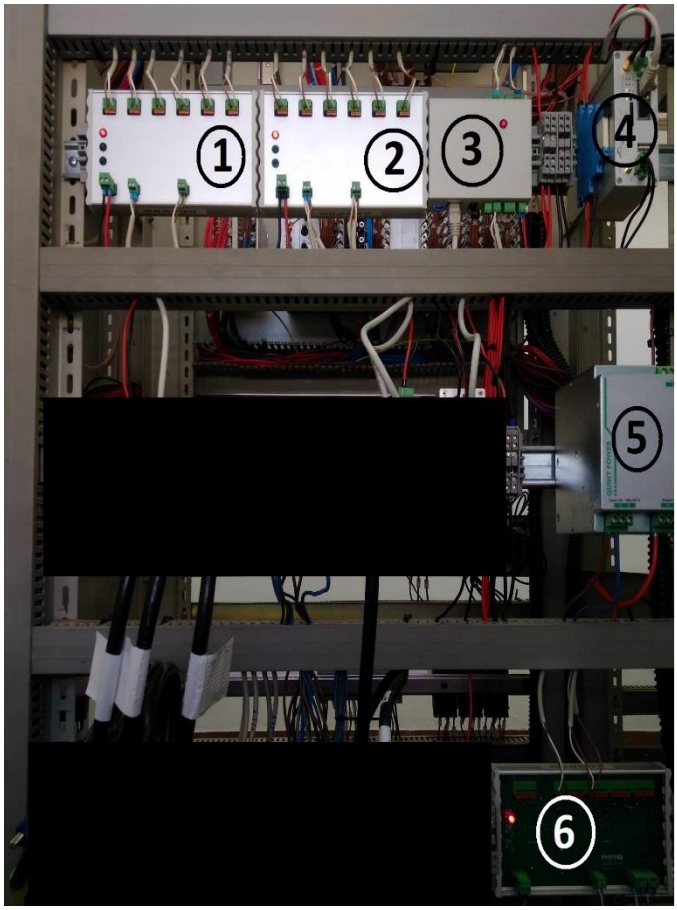


Figura 5.8 – Esquemático do ensaio laboratorial realizado com o sistema da Versão Final

A Figura 5.9 apresenta uma fotografia tirada durante o ensaio experimental realizado em ambiente industrial, onde se podem observar todos os elementos mencionados anteriormente, exceto os sensores de corrente e tensão.



Legenda:

1 – Placa Analógica nº1

2 – Placa Analógica nº2

3 – *Gateway*

4 – *Router*

5 – Transformador 230V/24V

6 – Placa Digital

Figura 5.9 – Ensaio industrial realizado com o sistema da Versão Final

As placa analógica número 1 adquiriu os três sinais de correntes de entrada e as três correntes de saída, e a placa analógica número dois adquiriu os três sinais de tensões de entrada e as três tensões de saída do transformador. As placas analógicas foram configuradas para uma aquisição de 20000 pontos, a uma frequência de aquisição de 20 kHz, o que resulta no total de 1 segundo de aquisição (Figura 5.10). As placas também foram configuradas para usar a totalidade dos canais, ou seja, 6 canais, como podemos ver pela Figura 5.9.

De seguida são apresentados os resultados obtidos neste ensaio laboratorial, assim como uma breve discussão sobre os mesmos.

5.2.2 Discussão de resultados

Esta secção tem como objetivo apresentar e discutir os resultados obtidos durante o ensaio experimental em ambiente industrial. Tal como referido anteriormente, o ficheiro de texto com os dados recolhidos e processados é retirado do cartão de memória na *gateway* através do *software* WinSCP para o computador pessoal, isto porque neste ensaio não houve comunicação com o servidor/base de dados. Os dados são inseridos num *software* próprio da empresa Enging, mas também podem ser analisados numa folha Excel.

A Figura 5.10 representa o gráfico completo de todos os dados recolhidos e processados no ensaio industrial, onde são apresentados todos os canais utilizados nas duas placas

analógicas, ou seja, um total de 12 canais. Os primeiros 6 canais (placa nº1) correspondem às correntes e os segundos 6 canais, do canal 7 ao canal 12 (placa nº2) correspondem às tensões. Os sinais resultantes não contêm o ganho do ADC utilizado na placa analógica.

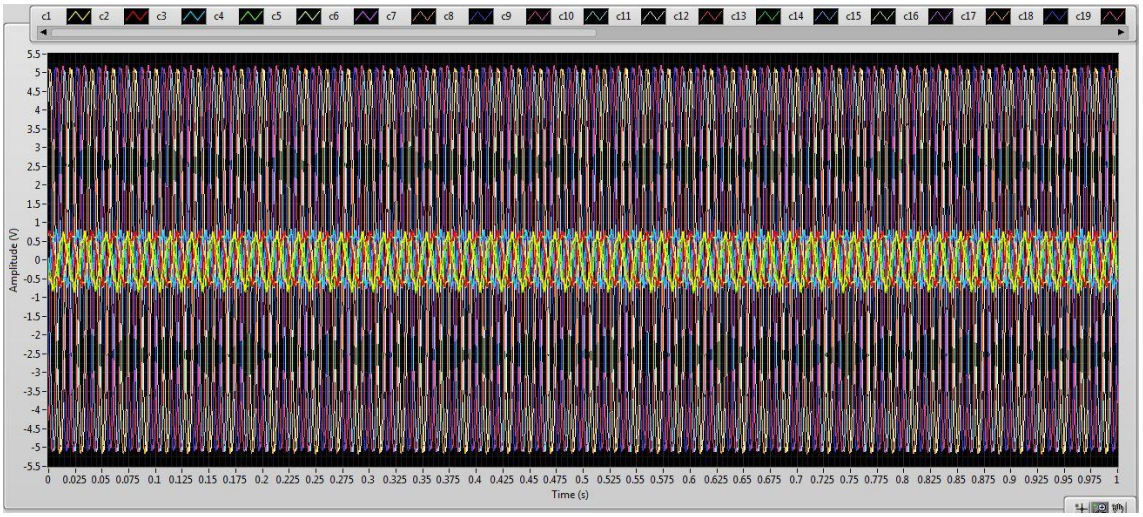


Figura 5.10 – Gráfico de sinais das correntes e tensões (12 canais)

Na Figura 5.11 temos representados os sinais completos (1 segundo de aquisição) das correntes de entrada e saída (canais 1,2,3,4,5 e 6) do transformador trifásico. Como os sinais resultantes das aquisições não contêm o ganho do ADC, os sinais apresentam um valor máximo de aproximadamente 0.8A e um valor mínimo de -0.8A (o *software* apresenta o eixo *y* como sendo Amplitude do sinal (Volts), pois este eixo é fixo, não difere entre correntes e tensões).

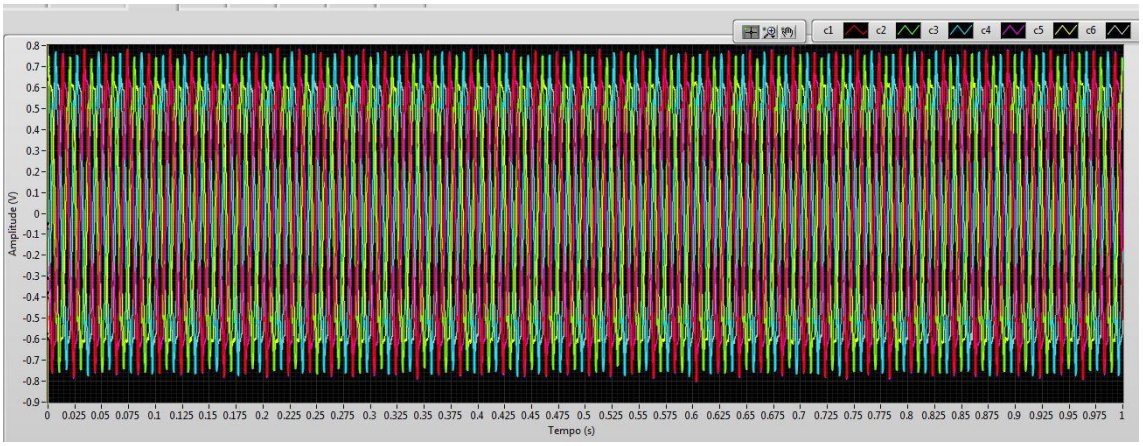


Figura 5.11 – Gráfico das correntes de entrada e saída (6 canais)

Na Figura 5.12 temos representados os sinais completos (1 segundo de aquisição) das tensões de entrada e saída (canais 7,8,9,10,11 e 12) do transformador trifásico, e como os sinais resultantes das aquisições não contêm o ganho do ADC, os sinais apresentam um valor máximo de aproximadamente 5 VAC e um valor mínimo de -5 VAC.

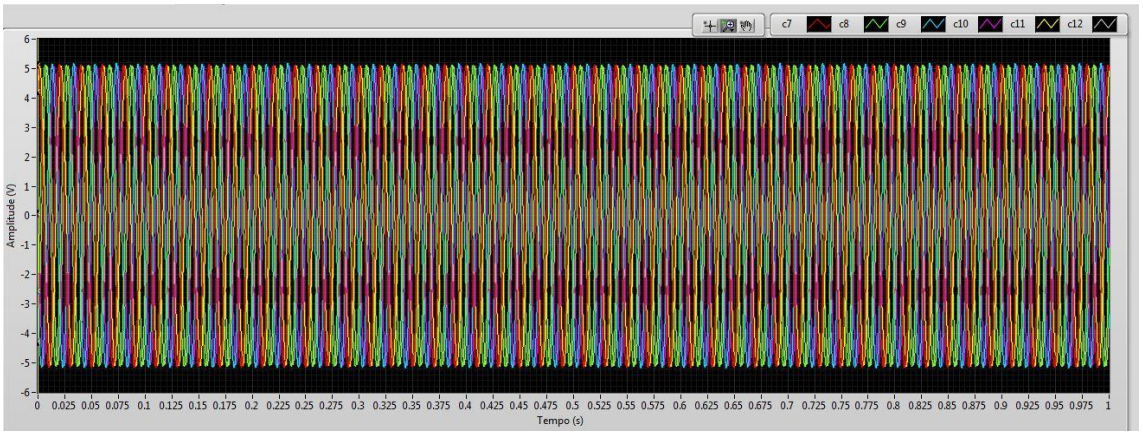


Figura 5.12 – Gráfico das tensões de entrada e saída (6 canais)

Na Figura 5.13 estão representados 2 períodos completos (0.04s) das três correntes de entrada do transformador trifásico: IfA, IfB e IfC, que correspondem ao canal 1 (vermelho), canal 2 (verde) e canal 3 (azul), respetivamente. Como podemos analisar na Figura 5.13, as correntes de entrada apresentam uma grande distorção harmónica.

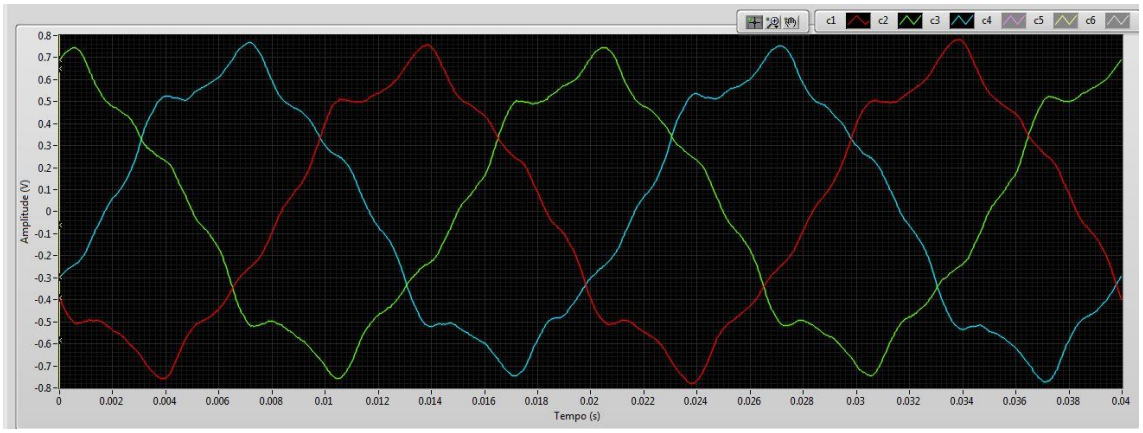


Figura 5.13 – Correntes de entrada (3 canais)

Na Figura 5.14 estão representados 2 períodos completos (0.04s) das três correntes de saída do transformador trifásico: Ifa, Ifb e Ifc, que correspondem ao canal 4 (roxo), canal 5 (amarelo) e canal 6 (cinza), respetivamente. Tal como nas correntes de entrada, as correntes de saída também apresentam uma grande distorção harmónica.

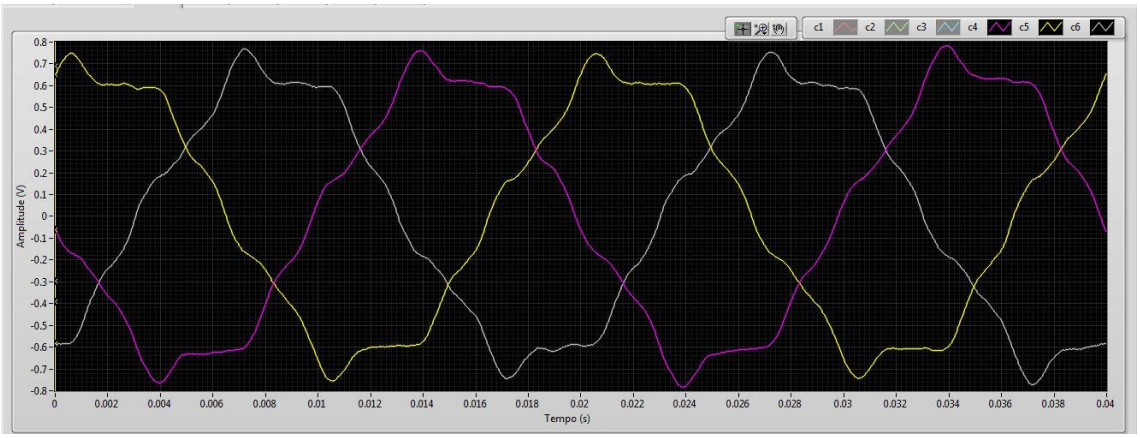


Figura 5.14 – Correntes de saída (3 canais)

Na Figura 5.15 estão representados 2 períodos completos (0.04s) das três tensões de entrada do transformador trifásico: VfA, VfB e VfC, que correspondem ao canal 7 (vermelho), canal 8 (verde) e canal 9 (azul), respetivamente. Em comparação com as correntes de entrada (Figura 5.12), as tensões de entrada formam 3 sinusoides "perfeitas", sem distorção harmónica visível, desfasadas de 120º entre cada fase.

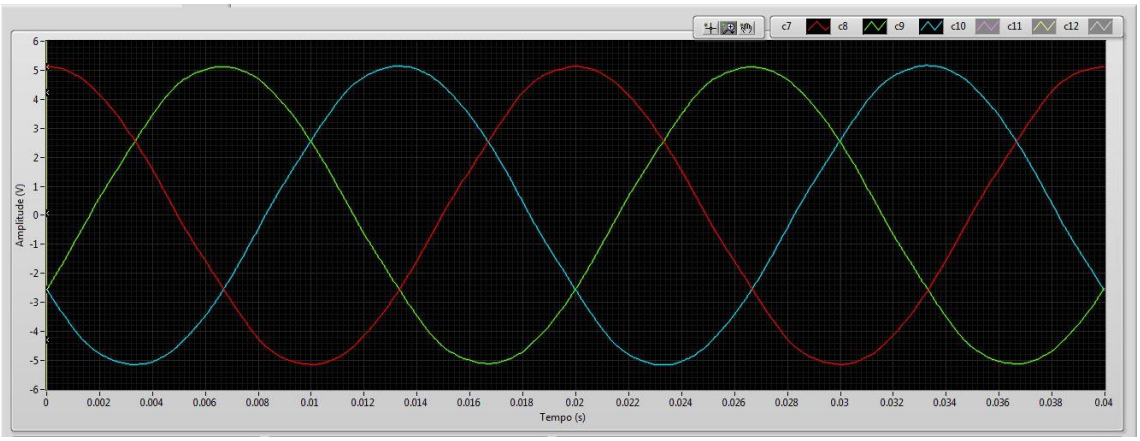


Figura 5.15 – Tensões de entrada (3 canais)

Na Figura 5.16 estão representados 2 períodos completos (0.04s) das três tensões de saída do transformador trifásico: Vfa, Vfb e Vfc, que correspondem ao canal 10 (roxo), canal 11 (amarelo) e canal 12 (cinza), respetivamente. Em comparação com as tensões de entrada (Figura 5.15) que não apresentam qualquer distorção harmónica, as tensões de saída apresentam uma ligeira distorção harmónica.

Figura 5.16 – Tensões de saída (3 canais)

Após ensaio experimental realizado em ambiente laboratorial com sucesso, surgiu a oportunidade de realizar o ensaio experimental em ambiente industrial através do sistema da Versão Final. Testou-se a recolha e processamento de dados adquiridos pelas várias placas do sistema, que continha 2 placas analógicas, onde a primeira adquiriu as correntes e a segunda as tensões do transformador trifásico. O sistema também continha uma placa digital, que recolheu sinais digitais.

Depois de realizado este ensaio experimental, o sistema continuou em funcionamento com um tempo entre aquisições de 3 minutos, durante cerca de 2 semanas, o que revelou apenas um problema na configuração do IP estático ao desligar e ligar o cabo de Ethernet. Este problema ficou resolvido logo de seguida com uma pequena instrução de código: "allow-hotplug eth0", tal como demonstrado no subcapítulo 3.5, de como configurar o IP estático na BeagleBone Black.

Este ensaio realizado em ambiente industrial permitiu confirmar o funcionamento integrado do sistema da Versão Final neste mesmo ambiente, onde está sujeito a diferentes fatores do sistema em ambiente laboratorial. Tal como no ensaio experimental em ambiente laboratorial, foram demonstradas as várias funcionalidades do sistema, desde comunicação, configurações e *software*.

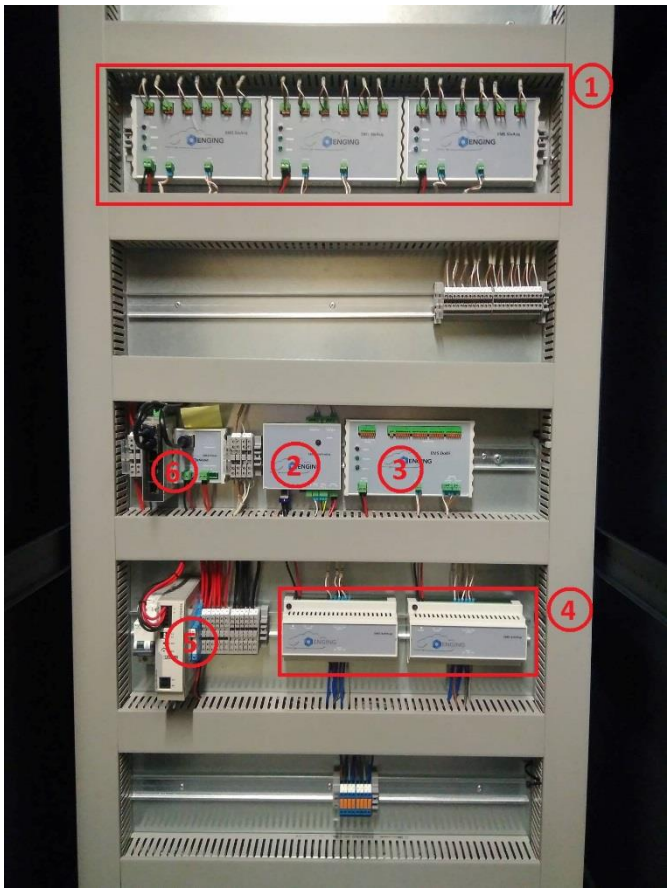
5.3 Implementação industrial

5.3.1 Industrialização

A industrialização pode ser um processo bastante complexo, composta por três fases principais: orçamento, certificação e comercialização. No sistema desenvolvido, não é possível descrever e analisar o orçamento do produto, pois este é composto por duas partes essenciais, a BeagleBone Black e a *Shield*. Apesar de conhecermos o preço da BeagleBone Black: 43.07€, a Shield foi um produto desenvolvido dentro da empresa, composto por uma PCB, circuitos integrados entre outros.

Para a empresa conseguir decidir se o produto é viável para industrialização, para além do orçamento que é o aspeto mais importante para esta decisão, o sistema necessitava de

mais testes para testar a viabilidade do produto. Assim a empresa possibilitou outros ensaios em ambientes industriais diferentes do primeiro, utilizando o bastidor da Figura 5.17. Este bastidor possui um sistema completo, que inclui a alimentação (5), composta por um disjuntor (esquerda), transformador (centro) e divisores (direita), um *router* (6) (esquerda), um Diorc (direita), este é um produto da Enging que funciona como um interruptor *IoT*, três placas analógicas (1), uma placa digital (3), dois sensores de tensão (4) e a *gateway* (2).



Legenda:

1 – Placas analógicas

2 – *gateway*

3 – Placa digital

4 – Sensores de tensão

5 – Alimentação (disjuntor, transformador e divisores)

6 – *Router* e Diork

Figura 5.17 – Bastidor com um sistema completo

5.3.2 Certificação

Uma certificação entende-se como uma declaração formal, emitida por entidades credíveis, que comprova que um determinado facto é verdadeiro. Neste caso, todo o conjunto denominado por *gateway* (Shield + BeagleBone Black) seria submetido a uma certificação CE, que representa um indicador de conformidade obrigatória para os produtos comercializados no Espaço Económico Europeu. A certificação CE pode ser obtida seguindo as seguintes etapas:

- Identificação dos requisitos aplicáveis a um determinado produto na União Europeia;
- Verificar se o produto cumpre os requisitos específicos;
- Verificar se o produto deve ser testado por um organismo certificado;
- Testar o produto;

- Apresentar uma ficha técnica;
- Aplicar a marcação CE e redigir uma declaração de conformidade.

Em relação aos requisitos específicos, cabe ao fabricante saber se o produto cumpre com todos os requisitos previstos na legislação europeia. Ou seja, se existirem normas europeias harmonizadas para o produto e, se estas forem respeitadas durante o processo de produção, o produto será automaticamente considerado conforme com as diretivas aplicáveis na União Europeia [18]. Por exemplo, a diretiva 2014/30/EU é responsável pela compatibilidade eletromagnética, a diretiva 2014/35/EU é responsável pelo material elétrico de baixa tensão e a diretiva 2014/32/EU é responsável pelos instrumentos de medição [19].

Geralmente, existem dois percursos possíveis para efetuar as etapas descritas anteriormente. Num primeiro percurso, a Enging poderá certificar a gateway submetendo-a a testes que permitam comprovar todos os requisitos para a obtenção da certificação CE. No entanto, uma vez que a Enging é a entidade responsável pela emissão do certificado do produto, por consequência, todas as responsabilidades relacionadas com o produto recaem sobre a Enging.

No outro percurso possível, a Enging poderá contratar uma empresa especializada em certificação de equipamentos. Nesse caso, é função dessa empresa comprovar todos os requisitos para a obtenção da certificação CE. Deste modo, todas as responsabilidades relacionadas com o produto reincidem sobre esta empresa, uma vez, que é ela a entidade certificadora.

5.3.3 Comercialização

A comercialização tem como objetivo vender um produto ou conceder as condições e os meios de distribuição necessários para a sua venda. A comercialização deste equipamento é incorporada na estratégia de comercialização da Enging, esta estratégia está de acordo com o diagrama representado na Figura 5.18.

Figura 5.18 – Estratégia de comercialização da Enging

Através do Marketing da Enging, que envolve principalmente publicidade e *Website* acerca de produtos e serviços, o cliente tem conhecimento sobre as soluções da Enging. Com a necessidade de resolução de um problema, o cliente solicita consultadoria à Enging, que por sua vez, recorrendo aos seus produtos e serviços apresenta uma solução. Se essa solução estiver em conformidade com as necessidades do cliente, então procede-se à sua venda.

Esta estratégia de comercialização incide na venda de uma solução, e não na venda de um produto. Por outro lado, esta solução pode ser apenas um produto, como também pode ser um conjunto de produtos onde cada um deles apresenta uma função específica para a solução global em causa.

6 Considerações Finais

Este capítulo está dividido em dois subcapítulos: Conclusão e Trabalho Futuro. No primeiro subcapítulo, a conclusão, tal como o nome indica, pretende-se apresentar uma abreve conclusão do trabalho desenvolvido. No segundo subcapítulo pretende-se apontar melhorias ao projeto desenvolvido, bem como novas funcionalidades evidenciadas durante o seu período de testes.

6.1 Conclusão

O projeto realizado durante o estágio resultou da contribuição de um conjunto de tarefas das quais se destacam:

- Recolha e análise de artigos, publicações, produtos existentes no mercado e outros tipos de informação relacionada com o tema em questão;
- Proposta do sistema a desenvolver e pesquisa do microcontrolador que o constitui;
- Desenvolvimento de *software* para testes de protocolos de comunicação;
- Desenvolvimento do algoritmo e do *software* onde foram implementadas as funcionalidades propostas;
- Realização de um ensaio experimental em ambiente de laboratório e de outro ensaio experimental em ambiente industrial com um transformador trifásico.

A tarefa de recolha e análise dos vários tipos de informação mencionada tornou-se bastante importante, sendo a base deste projeto, pois foi nesta fase que se tornou bastante evidente a implementação de soluções IoT no ambiente industrial, daí a evolução constante da popularidade da Indústria 4.0, bem como a monitorização de motores/transformadores elétricos. No entanto, concluiu-se que os equipamentos encontrados no mercado para estes fins, são bastante dispendiosos. Por outro lado, ao estudarem-se esses sistemas, foi possível obter uma perspetiva da complexidade de um sistema que junta soluções IoT com monitorização de sinais elétricos provenientes de motores/transformadores elétricos.

Na segunda tarefa e com base na anterior, foi apresentada uma proposta no sistema a desenvolver, onde é de referir que a topologia utilizada está funcional. Relativamente ao microcomputador que constitui o sistema, concluiu-se que implementa de forma eficiente e eficaz o funcionamento planeado, no entanto, em contrapartida, existem atualmente no mercado componentes mais sofisticados, no entanto, também mais dispendiosos.

A terceira tarefa correspondeu ao desenvolvimento de *software*, onde foram realizados testes experimentais com os protocolos de comunicação usados no sistema a desenvolver. Deste modo, foi possível confirmar a interação do microcomputador com vários circuitos integrados que usam os seguintes protocolos de comunicação: RS-232, RS-485 e SPI. Verificou-se ainda a implementação de entradas e saídas digitais.

A quarta tarefa correspondeu ao desenvolvimento do *software* do sistema. Esta tarefa foi constituída por três partes, sendo que houve uma versão inicial, uma versão final e uma

versão específica, e foi possível verificar que os algoritmos apresentados e descritos através de fluxogramas implementam as funcionalidades propostas.

A quinta tarefa correspondeu à realização de dois ensaios experimentais, um primeiro ensaio em ambiente laboratorial para validar as funcionalidades do sistema desenvolvido, que correspondeu à recolha e processamento de dados adquiridos de um transformador ligado à rede elétrica. O segundo ensaio experimental já foi realizado em ambiente industrial, onde foram recolhidos os sinais das correntes e tensões, de entrada e de saída, de um transformador trifásico. Para além da confirmação do funcionamento e do bom desempenho do sistema, verificaram-se também todos os algoritmos implementados a nível de *software*.

6.2 Trabalho Futuro

Como descrito ao longo deste relatório, todos os módulos do algoritmo descrito em fluxogramas foram verificados e testados tanto em ambiente laboratorial, bem como em ambiente industrial, permitindo validar as funcionalidades e a performance do sistema desenvolvido no ambiente onde deverá ser futuramente utilizado. Durante a fase de desenvolvimento foram detetadas várias possibilidades de melhorias, que foram implementadas. Para além destas, também foram detetadas outras possíveis melhorias, como o envio/gravação de dados analógicos e digitais ser efetuado no mesmo ficheiro no caso de gravação de dados, o que melhora bastante o tempo em que é realizada esta operação, bem como a utilização de recursos do processador, ou no caso do envio para o servidor/base de dados, os dois tipos de dados serem enviados na mesmas lista de *arrays*. Este processo torna-se muito vantajoso em termos temporais de comunicação, isto é, irá reduzir o tempo de todo o processo da gravação/envio de dados. Uma outra possível melhoria, seria atualizar o protocolo proprietário entre *gateway* e placas de aquisição de sinais analógicos e digitais, isto porque, o sistema está limitado a estes tipos de placas. Se futuramente for necessário implementar novas placas no sistema para aquisição de outras variáveis, como por exemplo, a temperatura e vibrações dos motores/transformadores elétricos, obrigatoriamente terá de se configurar um microcontrolador para controlar o sensor desejado (Ex: sensor de temperatura) e aplicar este protocolo proprietário, se este for atualizado e melhorado, pode possibilitar a adição, no barramento RS-485, de um sensor/placa para aquisição de novas variáveis.

Dependendo da experiência futura da utilização do sistema em aplicações concretas, é de esperar que se revelem necessárias ou desejáveis adaptações e melhorias no *software* que permitam uma adaptação às necessidades de recolha e processamento para os algoritmos de análise e para os serviços suportados pela *gateway*.

Referências Bibliográficas

[1] Akash Bhatia, Aviral Puri, and Akshay Behl, "Data Acquisition System," *IJIRT - International Journal of Innovative Research in Technology*, vol. 1, no. 12, 2015.

[2] National Instruments. Data Acquisition. [Online]. http://www.ni.com/data-acquisition/what-is/pt/ [Mar. 11, 2017]

[3] National Instruments. Measurement And Automation. [Online]. http://www.ni.com/pt-pt/shop.html [Mar. 11, 2017]

[4] DATAQ Instruments. Data Acquisition (DAQ). [Online]. http://www.dataq.com/data-acquisition/ [Mar. 11, 2017]

[5] Público. Indústria 4.0: uma oportunidade. [Online]. https://www.publico.pt/2017/05/07/economia/noticia/industria-40-uma-oportunidade-1771186/ [Mar. 25, 2017]

[6] TechTarget. Using an IoT *gateway* to connect "Things" to the cloud. [Online]. http://internetofthingsagenda.techtarget.com/feature/Using-an-IoT-*gateway*-to-connect-the-Things-to-the-cloud/ [Mar. 25, 2017]

[7] FAQInformática. O que é *Gateway* e qual é a sua função. [Online]. https://faqinformatica.com/*gateway*-o-que-e/ [Mar. 25, 2017]

[8] Arduino. ARDUINO LEONARDO ETH. [Online]. http://www.arduino.org/products/boards/arduino-leonardo-eth/ [Jan. 12, 2017]

[9] Raspberry Pi. STICKY: Getting Started with Raspberry Pi. https://www.raspberrypi.org/forums/viewtopic.php?t=4751/ [Jan. 12, 2017]

[10] BeagleBoard. BeagleBone Black. [Online]. https://beagleboard.org/black/ [Jan. 12, 2017]

[11] Adafruit. Raspberry Pi 2, Model B. [Online]. https://cdn-shop.adafruit.com/pdfs/raspbrrypi2modelb.pdf [Jan. 12, 2017]

[12] Ibex. RPi2 Model B IO Pins. [Online]. http://www.raspberry-projects.com/pi/pi-*hardware*/raspberry-pi-2-model-b/rpi2-model-b-io-pins/ [Jan. 12, 2017]

[13] BeagleBoard. BeagleBone: open-hardware expandable computer. [Online]. http://beagleboard.org/support/bone101 [Jan. 12, 2017]

[14] Elinux. Beagleboard: Cape Expansion Headers. [Online]. https://elinux.org/Beagleboard:Cape_Expansion_Headers [Fev. 23, 2017]

[15] Arcelect. RS232 Data Interface. [Online]. http://www.arcelect.com/rs232.htm [Fev. 23, 2017]

[16] Chipkin. RS485 – What is RS485, EIA-485. [Online]. http://www.chipkin.com/what-is-rs485-eia-485/ [Fev. 23, 2017]

[17] Sparkfun. Serial Peripheral Interface (SPI). [Online]. https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi [Fev. 23, 2017]

[18] Europa. (2017, Nov). Marcação CE. [Online]. http://europa.eu/youeurope/business/product/ce-mark/index_pt.htm [Dez. 28, 2017]

[19] IAPMEI – Agência para a Competitividade Inovação. (2016, Jun) Enterprise Europe Network. [Online]. http://www.enterpriseeuropenetwork.pt/info/mercadounico/Paginas/marcaoce.aspx [Dez. 28, 2017]

Anexos

Anexo A Especificações das possíveis placas de desenvolvimento

Anexo A.1 BeagleBone Black

	Feature	
Processor	Sitara AM3359AZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 606MHZ	
Onboard Flash	2GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 3 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

Anexo A.2 Raspberry Pi 2 Model B

Raspberry Pi

Raspberry Pi 2, Model B

Product Name	**Raspberry Pi 2, Model B**
Product Description	The Raspberry Pi 2 delivers 6 times the processing capacity of previous models. This second generation Raspberry Pi has an upgraded Broadcom BCM2836 processor, which is a powerful ARM Cortex-A7 based quad-core processor that runs at 900MHz. The board also features an increase in memory capacity to 1Gbyte.
Specifications	
Chip	Broadcom BCM2836 SoC
Core architecture	Quad-core ARM Cortex-A7
CPU	900 MHz
GPU	Dual Core VideoCore IV® Multimedia Co-Processor Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V, 2A
Connectors:	
Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4)
Audio Output	3.5mm jack, HDMI
USB	4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
JTAG	Not populated
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Micro SDIO

Anexo A.3 Arduino Leonardo Eth

Arduino Microprocessor

Processor	802.3 10/100 Mbit/s

Arduino Microcontroller

Microcontroller	ATmega32u4
Architecture	AVR
Operating Voltage	5V
Flash memory	32 KB of which 4 KB used by bootloader
SRAM	2.5Kb
Clock Speed	16 MHz
Analog I/O Pins	12
EEPROM	1 KB
DC Current per I/O Pins	40 mA on I/O Pins; 1A on 3.3 V Pin only when powered via external power supply

General

Input Voltage	7-12 V
Digital I/O Pins	36-57 V
Reserved Pins	4 used for SD card select; 10 used for W5500 select
Digital I/O Pins	20
PWM Output	7
Power Consumption	82 mA
PCB Size	53.34 x 68.58 mm
Card Reader	Micro SD card, with active voltage translators
Weight	28g
Product Code	A000108

Anexo B *Scripts* de testes iniciais

Anexo B.1 GPIO output

blink.py

```
#importar librarias
import Adafruit_BBIO.GPIO as GPIO
import time

#Configurar pino como saida digital
LED = "P9_12"
GPIO.setup(LED, GPIO.OUT)


#piscar LED 5 vezes de segundo em segundo
for i in range(0,5):

      GPIO.output(LED, GPIO.HIGH) # LED acende
      time.sleep(1) #espera 1 segundo
      GPIO.output(LED, GPIO.LOW) # LED apaga
      time.sleep(1) #espera 1 segundo

GPIO.cleanup()
```

Anexo B.2 GPIO input

input.py

```
#importar librarias
import Adafruit_BBIO.GPIO as GPIO
import time

#Configurar pino como entrada digital
interrupt = "P9_12"
GPIO.setup(interrupt, GPIO.IN)


#Sempre que o botao for pressionado, mostra uma mensagem
#Se for pressionado durante 2 segundos, o programa termina
```

```
while True:
      time.sleep(0.5) # delay adicionado, para nao ler constantemente
o estado da entrada digital
      if GPIO.input(interrupt) == 1:
            print ' >>> Botao Pressionado <<< '
            time.sleep(2)
            if GPIO.input(interrupt) == 1:
                  print ' Botao Pressionado durante 2 segundos, fechar
programa '
                  break
            else:
                  pass

GPIO.cleanup()
```

Anexo B.3 UART

tx.py

```
#importar librarias
import Adafruit_BBIO.UART as UART
import serial, time

#iniciar e configurar uart (port,baudrate)
UART.setup("UART1")
uart = serial.Serial('/dev/ttyO1',115200)

string_enviar = raw_input("Digite a string a enviar\n")
print ' Tamanho da string = ', len(string_enviar)
#enviar string0
uart.write(string_enviar)

while uart.inWaiting() == 0:
      #enquanto nao ha dados para ler ...
      pass

#variavel para ler quantos bytes estao no buffer
dados_buffer = uart.inWaiting()
print ' Numero de bytes no buffer ->', dados_buffer
```

```
#ler numero de bytes no buffer
data = uart.read(dados_buffer)
#funcao ord() converte string hexadecimal para decimal
print 'Recebi : ', ord(data),'=',hex(ord(data))

#fechar uart
uart.close()
```

rx.py

```
#importar librarias
import Adafruit_BBIO.UART as UART
import serial, time

# iniciar e configurar uart (port,baudrate)
UART.setup("UART4")
uart = serial.Serial('/dev/ttyO4',115200)

while uart.inWaiting() == 0:
      #enquanto nao existe dados para ler ...
      pass

#variavel para ler quantos bytes estao no buffer
dados_buffer = uart.inWaiting()
print ' Numero de bytes no buffer ->', dados_buffer

#ler numero de bytes no buffer
data = uart.read(dados_buffer)
print ' Dados recebidos - ', data

#enviar 0xFF(255)
uart.write('\xFF')

#fechar uart
uart.close()
```

Anexo B.4 RS-232

rs232.py

```
#importar librarias
import Adafruit_BBIO.UART as UART
import serial, time, struct

#iniciar e configurar uart
UART.setup("UART4")
uart = serial.Serial(port='/dev/ttyO4',baudrate=115200, parity="N",
stopbits=1, timeout = None)

def process(number):

      #iniciar variaveis
      array = []

      #criar array de valores de 1 a number(valor recebido)
      for x in range(1,number+1):
            array.append(x)

      #converter array para string hexadecimal -> "\x00\x01\x02..."
      string_hexa = struct.pack('B'*number, *array)

      return string_hexa

while True:

      data = uart.read(1) #ler apenas 1 byte
      data_number = ord(data) # converter string hexadecimal para
decimal
      print ' Recebeu -> ',hex(data_number),' = ', data_number
      string_hexa = process(data_number)

      print ' Enviar string hexadecimal de %d bytes ' % data_number,
len(string_hexa)
      uart.write(string_hexa)
      time.sleep(0.1)

      if data_number == 255:
            # se receber o valor 255(FF) acaba o programa
            print ' Fechar programa '
            uart.close()
            break
```

Anexo B.5 RS-485

bbb_1.py

```
#importar librarias
import Adafruit_BBIO.UART as UART
import Adafruit_BBIO.GPIO as GPIO
import serial, time, struct

#iniciar e configurar uart
UART.setup("UART4")
uart = serial.Serial(port='/dev/ttyO4',baudrate=115200, parity="N",
stopbits=1, timeout = None)

#configurar GPIOs
RE = 'P8_7'
GPIO.setup(RE,GPIO.OUT)
DE = 'P8_8'
GPIO.setup(DE,GPIO.OUT)


def rs_send():
      #RE e DE a 1 para enviar
      GPIO.output(RE, GPIO.HIGH)
      GPIO.output(DE, GPIO.HIGH)
def rs_receive():
      #RE e DE a 0 para receber
      GPIO.output(RE, GPIO.LOW)
      GPIO.output(DE, GPIO.LOW)

def process(number):

      #iniciar variaveis
      array = []

      #criar array de valores de 1 a number(valor recebido)
      for x in range(1,number+1):
            array.append(x)
```

```
        #converter array para string hexadecimal -> "\x00\x01\x02..."
        string_hexa = struct.pack('B'*number, *array)

        return string_hexa

while True:

        rs_receive()
        data = uart.read(1) #ler apenas 1 byte
        data_number = ord(data) # converter string hexa para decimal
        print ' Recebeu -> ',hex(data_number),' = ', data_number
        string_hexa = process(data_number)
        #time.sleep(0.1)
        print ' Enviar string hexadecimal de %d bytes' %
len(string_hexa)
        rs_send()
        uart.write(string_hexa)
        time.sleep(1)
        uart.flushInput()
        if data_number == 255:
                # se receber o valor 255(FF) acaba o programa
                print ' Fechar programa '
                uart.close()
                break
```

bbb_2.py

```
#importar librarias
import Adafruit_BBIO.UART as UART
import Adafruit_BBIO.GPIO as GPIO
import serial, time, struct

#iniciar e configurar uart
UART.setup("UART4")
uart = serial.Serial(port='/dev/ttyO4',baudrate=115200, parity="N",
stopbits=1, timeout = None)

#configurar GPIOs
RE = 'P8_7'
```

```
GPIO.setup(RE,GPIO.OUT)
DE = 'P8_8'
GPIO.setup(DE,GPIO.OUT)


def rs_send():
      #RE e DE a 1 para enviar
      GPIO.output(RE, GPIO.HIGH)
      GPIO.output(DE, GPIO.HIGH)
def rs_receive():
      #RE e DE a 0 para receber
      GPIO.output(RE, GPIO.LOW)
      GPIO.output(DE, GPIO.LOW)

def process(number,dados):

      #converter string hexadecimal para array decimal

      string_hexa = struct.unpack('B'*number, dados)

      return string_hexa



while True:


      data_send = input("Valor decimal a enviar(0-255)\n")
      rs_send()
      hexstring_data = struct.pack('B',data_send)#converter de decimal
para string hexadecimal
      uart.write(hexstring_data)
      uart.flushInput()
      time.sleep(0.001)#esperar o tempo de envio de um byte antes de
mudar o estado do RE e DE
      rs_receive()
      data = uart.read(data_send) #ler numero de bytes(valor enviado+1
(valor 0))


      print ' Recebeu -> ',len(data),' bytes '
```

```
      string_hexa = process(data_send,data)

      print ' Valores recebidos: ', string_hexa

      if data_send == 255:
            # se receber o valor 255(FF) acaba o programa
            print ' Fechar programa '
            uart.close()
            break
```

Anexo B.6 SPI

memory_debug.py

```
#importar librarias
import spidev
import time


#configurar SPI
spi = spidev.SpiDev()
spi.open(1, 0)
spi.max_speed_hz = 1000000 # SCLK 4 MHz
spi.mode = 0 #modo 0 - clock 0 e fase 0
spi.lsbfirst = False #msb primeiro
spi.threewire = False #modo 4 fios
spi.bits_per_word = 8 # 8 bits
spi.cshigh = False # CS 0

#variaveis

read_status = 0b00000101 #comando para ler status da memoria
write_status = 0b00000001 #comando para escrever status da memoria

read_byte = 0b00000011 #comando para ler da memoria
write_byte = 0b00000010 #comando para escrever na memoria

def r_status():
```

```
	#ler status do rtc
	rdsr = spi.xfer2([read_status, 0])

	if rdsr[1] == 0:
		print ' Status : Modo Byte '
		return 'b'
	if rdsr[1] == 64:
		print ' Status : Modo Sequencial '
		return 's'

def w_status(wdsr):

	byte_status = 0b00000000 #comando para alterar o status para em modo 'byte'
	sequencial_status = 0b01000000 #comando para alterar o status para modo 'sequencial'
	if wdsr == 'byte':
		spi.xfer2([write_status,byte_status])
		print ' Estamos agora em modo Byte '
		return 'b'
	if wdsr == 'sequencial':
		spi.xfer2([write_status,sequencial_status])
		print ' Estamos agora em modo sequencial '
		return 's'

def modo(modo_bs, no_change):
	if modo_bs == 'sim':
		modo_bs = raw_input("Qual modo?  (byte/sequencial) \n")
		modo = w_status(modo_bs)
		return modo
	if modo_bs == 'nao':
		return no_change

def w_byte(statu, addrr):

	'''
	Esta funcao tem como objetivo escrever dados na memória nos dois modos, byte e sequencial.
	A funcao tem como argumentos o status(statu) e endereco(addrr)
	'''
```

```
	wrr = raw_input("O que escrever?(Em modo sequencial, separar por espacos os caracteres)\n")
	#separar o endereco(addrr) de 16 bytes em duas varisveis de 8 bytes cada (addr_msb e addr_lsb)
	addr_msb = (addrr >> 8) & 0xff
	addr_lsb = addrr & 0xff
	if statu == 'b':
		#status em modo byte
		spi.xfer2([write_byte,addr_msb,addr_lsb,ord(wrr)])
	if statu == 's':
		#status em modo sequencial
		wrr2 = wrr.split()#separar a string de 5 carateres em 5 strings de 1 caracter
		#instrucao ord() converte string para decimal(ascii)

	spi.xfer2([write_byte,addr_msb,addr_lsb,ord(wrr2[0]),ord(wrr2[1]),ord(wrr2[2]),ord(wrr2[3]),ord(wrr2[4])])



def r_byte(statu, addrr):

	'''
	Esta funcao tem como objetivo ler dados na memória nos dois modos, byte e sequencial
	A funcao tem como argumentos o status(statu) e endereco(addrr)
	'''

	#separar o endereco(addrr) de 16 bytes em duas variáveis de 8 bytes cada (addr_msb e addr_lsb)
	addr_msb = (addrr >> 8) & 0xff
	addr_lsb = addrr & 0xff
	if statu == 'b':
		#status em modo byte
		leitura = spi.xfer2([read_byte,addr_msb,addr_lsb,0])
		print ' Valor na posicao %d -> %d = %s ' % (addrr, leitura[3], str(unichr(leitura[3])))
	if statu == 's':
		#status em modo sequencial
		lecture = []
		response = spi.xfer2([read_byte,addr_msb,addr_lsb,0,0,0,0,0])
		leitura = response[3:]
```

```
            for x in leitura:
                  caractere = str(unichr(x))
                  lecture.append(caractere)
            print ' Valor na posicao %d -> {} = {}
'.format(leitura,lecture) % addrr

if __name__ == "__main__":

      # Funcao Main

      # Comeca por ler o status
      print ' Reading Status '
      status = r_status()

      # Pergunta se quer mudar o status
      change_status = raw_input(' Mudar de modo?  (sim/nao) \n')
      aux_status = status
      # Escrever status se for necessario mudar
      status = modo(change_status, aux_status)

      # Escolher Ler ou Escrever na memoria
      rw = raw_input("Ler ou Escrever? (modo sequencial->5 bytes) \n")
      if rw == 'Ler':
            #Ler
            # Escolher endereço dentro da gama 0-65535
            addr = input("Endereco? (16 bits -> 0-65535)\n")
            r_byte(status,addr)
      if rw == 'Escrever':
            #Escrever
            # Escolher endereço dentro da gama 0-65535
            addr = input("Endereco? (16 bits -> 0-65535)\n")
            w_byte(status,addr)

      #fechar spi
      spi.close()
```

Anexo C Ficheiro de configurações e Logs

Anexo C.1 Ficheiro de configurações


```
{
"Analogicas":
      {
            "n_placas": 6,
            "frequencia": 20,
            "n_canais": 6,
            "n_amostras": 20,
            "ids":
                  [[0,1,2,100,170,249],
                  [0,1,2,110,170,250],
                  [0,1,2,120,170,251],
                  [0,1,2,130,170,252],
                  [0,1,2,140,170,253],
                  [0,1,2,150,170,254]]
      },

      "Digitais":
      {
            "n_placas": 1,
            "ids":[[0,1,2,100,110,120]]
      },

      "Calibration":
      {
            "placas": [4,5,6],
            "calibration_values": {
            "4":[1.014,1.0216,1.015,1.0058,1.0066,1.0054],
            "5":[1.0092,1.0056,1.0068,1.0106,1.0098,1.0096],
            "6":[1.0054,1.0068,1.0048,1.0044,1.0092,1.005]}
      },

      "RS485_0":
      {
            "port": "/dev/ttyO1",
            "baudrate": 115200,
            "parity": "N",
```

```
        "stopbits": 1,
        "timeout": 1
    },

    "RS485_1":
    {
        "port": "/dev/ttyO4",
        "baudrate": 115200,
        "parity": "N",
        "stopbits": 1,
        "timeout": 1
    },

    "RS232":
    {
        "port": "/dev/ttyO5",
        "baudrate": 115200,
        "parity": "N",
        "stopbits": 1,
        "timeout": 1
    },

    "IOs e Timer":
    {
        "trigger": "P9_30",
        "input_DI": "P8_18",
        "timer": 5
    },

    "Cliente":
    {
        "cliente": "Cliente",
        "local": "local",
        "tipo": "A_D"
    },

    "Servidor":
    {
        "ip": "10.0.0.1",
        "port": 1000
```

```
    }
}
```

Anexo C.2 Classes de Logs

```
                    ***** CLASS *****

1000 - Comunication Server-Gateway
1100 - Interrupts (DI, Timer)
1200 - Acquisition / Collecting Data (Analog)
    1250 - (Digital)
1300 - Process Data (Analog)
    1350 - (Digital)
1400 - Socket Comunication (Estoril)
    1450 - (Alcoitão)
1500 - Monitoring sdCard
1600 - Traceback Errors

                    ***** Sub-class *****

1000 - Comunication Server-Gateway
    1000 - Server Trigger
    1010 - Change Configs
    1020 - Send Analog Data
    1030 - Send Digital Data

1100 - Interrupts (DI, Timer)
    1100 - Time schedule
    1110 - DI
    1120 - Main Interrupt

1200 - Acquisition / Collecting Data (Analog)
    1200 - Broadcast
    1201 - Collecting Data / Debug
    1210 - Invalid Data
    1220 - Missing Bytes
```

```
      1230 - Get Data Again
      1240 - Send ID Again

1250 - Acquisition / Collecting Data (Digital)
      1250 - Collecting Data / Debug
      1260 - Invalid Data
      1270 - Missing Bytes
      1280 - Get Data Again
      1290 - Send ID again

1300 - Process Data (Analog)
      1300 - Process Invalid Data
      1310 - Debug
      1320 - Request/Joining Data ***
      1330 - Check/Get Files *

1350 - Process Data (Digital)
      1350 - Debug

1400 - Socket Comunication (Estoril)**
      1400 - Connection/Debug
      1410 - Data request
      1420 - Trigger from Server
      1430 - Trigger from DI
1450 - Socket Comunication (Alcoitao)*
      1450 - Connection/Debug
      1460 - Send Parcial Data
      1470 - Trigger from Server
      1480 - Trigger from DI
      1490 - Request Part File

1500 - Monitoring sdCard
      1500 - Debug
      1510 - File with parcial data***
      1520 - Socket communication for parcial data***
      1530 - Analog File
      1540 - Digital File
      1550 - Delete old files

1600 - Traceback Errors
```

```
      1600 - Start
      1610 - Main
      1620 - Socket*

                        ***** Logs *****

1000 - Comunication Server-Gateway
      1000 - Server Trigger
            -"1000:Trigger from server received: {}".format.(device)"
      1010 - Change Configs
            -"1010:Config received from server: {}".format(payload)"
            -"1011:Acquisition in progress, delay reboot"
            -"1012:System will reboot"
      1020 - Send Analog Data
            -"1020:Analog Reading successfully sent"
            -"1021:Failed sending message to server, save analog data
in txt"
      1030 - Send Digital Data
            -"1030:Digital Reading successfully sent to server"
            -"1031:Failed sending message to server, save digital data
in txt"

1100 - Interrupts (DI, Timer)
      1100 - Time schedule
            -"1100:First time schdule trigger at %s"
            -"1101:Trigger from time schedule at %s"
            -"1102:Detected event -> %s" ***
      1110 - DI
            -"1110:Trigger from Digital Board(DI) at %s"
            -"1111:Trigger DI before data acquisition, doing new
acquisition"
      1120 - Main Interrupt
            -"1120:>>>>>>>>>>     Main Interrupts active, aquisition
will start at %s"
            -"1121:>>>>>>>>>>     Main Interrupts reactive at %s"
            -"1122:>>>>>>>>>>     Main Interrupts finish at %s"

1200 - Acquisition / Collecting Data (Analog)
      1200 - Broadcast
            -"1200:Broadcast sent at %s (acquitision timestamp)"
      1201 - Collecting Data / Debug
```

```
            -"1201:Collecting data from Analog Boards(AI) will start at
%s"
            -"1202:Collecting from Analog Board(AI) Complete - %s "
            -"1203:Doesn't exist Analog Boards(AI)"
            -"1204:Collecting data from all Analog Boards(AI) in the
system Complete"
            -"1205:Doesn't exist Digital Boards(DI) in the system"
            -"1206:Doesn't exist Analog Boards(AI) to process"
            -"1207:Enter in Process Mode-Analog"
            -"1208:Process and Analog Data Record Complete at - %s"
      1210 - Invalid Data
            -"1210:Fail(1) on data received, invalid data from Analog
Boards(AI) - %s"
            -"1211:Fail(2) on data received, invalid data from Analog
Boards(AI) - %s"
            -"1212:Fail(3) on data received, invalid data from Analog
Boards(AI) - %s"
      1220 - Missing Bytes
            -"1221:Fail(3) on data received, missed bytes from Analog
Board(AI) - %s"
            -"1222:Fail(2) on data received, missed bytes from Analog
Board(AI) - %s"
            -"1223:Fail(1) on data received, missed bytes from Analog
Board(AI) - %s"
      1230 - Get Data Again
            -"1230:Send ID again: %s"
            -"1231:Fail on received data(2x), invalid data from Analog
Board(AI) - %s"
      1240 - Fail ID
            -"1240:Fail(1) to send ID to Analog Board(AI) - %s"
            -"1241:Fail(2) to send ID to Analog Board(AI) - %s"
            -"1242:Fail(3) to send ID to Analog Board(AI) - %s"
            -"1243:Send ID again(2)"
            -"1244:Send ID again(3)"
1250 - Acquisition / Collecting Data (Digital)
      1250 - Collecting Data / Debug
            -"1250:Collecting from Digital Board(DI) Complete - %s "
            -"1251:Collecting(2) from Digital Board(DI) Complete - %s "
            -"1252:Collecting(3) from Digital Board(DI) Complete - %s "
            -"1253:Collecting(2x) from Digital Board(DI) Complete - %s
"
            -"1256:Collecting data from all Dnalog Boards(DI) in the
system Complete"
            -"1257:Enter in Process Mode-Digital"
```

```
            -"1258:Process and Digital Data Record Complete at - %s"
      1260 - Invalid Data
      1270 - Missing Bytes
            -"1270:Fail(3) on data received, missed bytes from Digital
Board(DI) - %s"
            -"1271:Fail(2) on data received, missed bytes from Digital
Board(DI) - %s"
            -"1272:Fail(1) on data received, missed bytes from Digital
Board(DI) - %s"
      1280 - Get Data Again
            -"1280:Send ID again"
      1290 - Send ID again
            -"1290:Fail(1) to send ID to Digital Board(DI) - %s"
            -"1291:Fail(2) to send ID to Digital Board(DI) - %s"
            -"1292:Fail(3) to send ID to Digital Board(DI) - %s"

1300 - Process Data (Analog)
      1300 - Process Invalid Data
            -"1300:Process error data"
      1310 - Debug
            -"1310:Processing Analog Data"
            -"1311:Finish data process, waiting for request"
      1320 - Request/Joining Data
            -"1320:Request received" *
            -"1321:Din't receive request, saving data in txt file" *
            -"1322:Joining data together" **
            -"1323: Didn't receive any data, saving data in txt file"
**
      1330 - Check/Get Files
            -"1330:File name with timestamp %s exist" *
            -"1331:File name with timestamp %s doesn't exist" *
            -"1332:Opening file >> %s" *
            -"1333:File >> %s  removed" *
1350 - Process Data (Digital)
      1350 - Debug
            -"1350:Processing Digital Data"

1400 - Socket Comunication (Estoril) **
      1400 - Connection/Debug
            -"1400:Connection to: %s  establish"
            -"1401:No Route to Host, connection denied"
```

```
            -"1402:Closing socket"
            -"1403:Sending confirmation"
      1410 - Data request
            -"1410:Sending data request"
            -"1411:Receiving data"
            -"1412:Processing data string"
      1420 - Trigger from Server
            -"1420:Sending Trigger from Server"
      1430 - Trigger from DI
            -"1430:Sending trigger DI"

1450 - Socket Comunication (Alcoitão) *
      1450 - Connection/Debug
            -"1450:Waiting socket connection"
            -"1451:Connection from: %s"
            -"1452:Receive request: %s"
      1460 - Send Parcial Data
            -"1461:Sending Data"
            -"1462:Confirmation received: %s"
      1470 - Trigger from Server
            -"1470:Trigger received from Estoril(Server)"
      1480 - Trigger from DI
            -"1480:Trigger received from GW1(DI)"
      1490 - Request Part File
            -"1490:Request to a file part received"
            -"1491:Sending file part to GW1"
            -"1492:Informing GW1 that file part doesn't exist"

1500 - Monitoring sdCard
      1500 - Debug
            -"1500:Analizing sdCard, %s files detected"
      1510 - File with parcial data***
            -"1510:Parcial data detected with timestamp >> %s"
            -"1511:Opening file and joining data"
            -"1512:Joining data complete"
            -"1513:Analog joined data successfully sent"
            -"1514:Failed sending message to server"
            -"1515:Saving joined data with title - %s"
            -"1516:File >> %s  removed"
      1520 - Socket communication for parcial data***
```

```
            -"1520:Connection to: %s  establish"
            -"1521:No Route to Host, connection denied"
            -"1522:Sending data request with timestamp >> %s"
            -"1523:File exist in GW2(Alcoitao), receiving data"
            -"1524:File don't exist in GW2(Alcoitao)"
            -"1525:Closing Socket"
      1530 - Analog File
            -"1530:Analog file detected with timestamp >> %s"
            -"1531:Processing data of open file"
            -"1532:Analog Reading successfully sent"
            -"1533:Analog Reading failed sending message to server"
      1540 - Digital File
            -"1540:Digital file detected with timestamp >> %s"
            -"1541:Processing data of open file"
            -"1542:Digital Reading successfully sent"
            -"1543:Digital Reading failed sending message to server"
      1550 - Delete old files
            -"1550:Comparing timestamp of files in sdCard with >>> %s"
            -"1551:{} removed at %s"

1600 - Traceback Errors
      1600 - Start
            -"1600:ERROR IN START-THE TRACEBACK:\n {}"
      1610 - Main
            -"1610:ERROR IN MAIN-THE TRACEBACK:\n {}"
      1620 - Socket
            -"1620:ERROR IN SOCKET-THE TRACEBACK:\n {}"
```

*apenas aplicável na *Gateway* 1 no sistema da versão específica - GPS

**apenas aplicável na *Gateway* 2 no sistema da versão específica - GPS

***apenas aplicável na *Gateway* 1 e *Gateway* 2 no sistema da versão específica - GPS

Anexo C.3 Logs Genérico

```
DEBUG:Gateway:Logs Created
INFO:root:Attempting to connect websocket.
INFO:root:Connection established
```

```
INFO:Gateway:Start

INFO:Gateway:1100:First time schedule trigger at 2017-05-29 16:55:00

DEBUG:Gateway:1120:>>>>>>>>>>     Main Interrupts active, aquisition
will start at 2017-05-29 16:55:00

DEBUG:Gateway:1200:Broadcast sent at 2017-05-29 16:55:00(acquitision
timestamp)

INFO:Gateway:1201:Collecting data from Analog Boards(AI) will start at
2017-05-29 16:55:00

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(1,2,3,110,170,153)

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(1,2,3,110,170,154)

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(1,2,3,110,170,155)

DEBUG:Gateway:1204:Collecting data from all Analog Boards(AI) in the
system Complete

DEBUG:Gateway:1250:Collecting from Digital Board(AI) Complete -
(100,110,120,130,140,150)

DEBUG:Gateway:1256:Collecting data from all Digital Boards(AI) in the
system Complete

INFO:Gateway:1257:Enter in Process Mode-Digital

INFO:Gateway:1350:Processing Digital Data

INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS
connection (1): 10.0.0.2

DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings
HTTP/1.1" 200 82

DEBUG:Gateway:1020:Analog Reading successfully sent

INFO:Gateway:1207:Enter in Process Mode-Analog

INFO:Gateway:1310:Processing Analog Data

INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS
connection (1): 10.0.0.2

DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings
HTTP/1.1" 200 82

DEBUG:Gateway:1020:Analog Reading successfully sent

DEBUG:Gateway:1208:Process and Analog Data Record Complete at - 2017-
05-29 16:56:23

DEBUG:Gateway:1122:>>>>>>>>>>     Main Interrupts finish at 2017-05-29
16:56:23

DEBUG:SDcard:1500:Analyzing sdCard, 0 files detected

INFO:Gateway:1101:Trigger from time schedule at 2017-05-29 17:00:00

DEBUG:Gateway:1120:>>>>>>>>>>     Main Interrupts active, aquisition
will start at 2017-05-29 16:55:00

DEBUG:Gateway:1200:Broadcast sent at 2017-05-29 16:55:00(acquitision
timestamp)

INFO:Gateway:1201:Collecting data from Analog Boards(AI) will start at
2017-05-29 16:55:00
```

```
DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(1,2,3,110,170,153)

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(1,2,3,110,170,154)

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(1,2,3,110,170,155)

DEBUG:Gateway:1204:Collecting data from all Analog Boards(AI) in the
system Complete

DEBUG:Gateway:1250:Collecting from Digital Board(AI) Complete -
(100,110,120,130,140,150)

DEBUG:Gateway:1256:Collecting data from all Digital Boards(AI) in the
system Complete

INFO:Gateway:1257:Enter in Process Mode-Digital

INFO:Gateway:1350:Processing Digital Data

INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS
connection (1): 10.0.0.2

DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings
HTTP/1.1" 200 82

DEBUG:Gateway:1020:Analog Reading successfully sent

INFO:Gateway:1207:Enter in Process Mode-Analog

INFO:Gateway:1310:Processing Analog Data

INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS
connection (1): 10.0.0.2

DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings
HTTP/1.1" 200 82

DEBUG:Gateway:1020:Analog Reading successfully sent

DEBUG:Gateway:1208:Process and Analog Data Record Complete at - 2017-
05-29 17:01:23

DEBUG:Gateway:1122:>>>>>>>>>>     Main Interrupts finish at 2017-05-29
17:01:23

DEBUG:SDcard:1500:Analyzing sdCard, 0 files detected
```

Anexo C.4 Logs GPS

Logs *Gateway* 1

```
DEBUG:Gateway:Logs Created

INFO:root:Attempting to connect websocket.

INFO:root:Connection established

INFO:Gateway:Start

INFO:Gateway:1102:Detected event -> 2017-06-12 15:59:59

DEBUG:Gateway:1120:>>>>>>>>>>     Main Interrupts active, aquisition
will start

DEBUG:Gateway:1200:Broadcast sent at 2017-06-12 16:00:00(acquitision
timestamp)

INFO:Gateway:1201:Collecting data from Analog Boards(AI) will start at
2017-11-06 16:00:02
```

```
DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(100,101,102,103,104,105)

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(106,107,108,109,110,111)

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(112,113,114,115,116,117)

DEBUG:Gateway:1204:Collecting data from all Analog Boards(AI) in the
system Complete

DEBUG:Gateway:1250:Collecting from Digital Board(DI) Complete -
(200,201,202,203,204,205)

DEBUG:Gateway:1256:Collecting data from all Analog Boards(AI) in the
system Complete

INFO:Gateway:1257:Enter in Process Mode-Digital

INFO:Gateway:1350:Processing Digital Data

INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS
connection (1): 10.0.0.201

DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings
HTTP/1.1" 200 45

DEBUG:Gateway:1030:Digital Reading successfully sent to server

DEBUG:Gateway:1258:Process and Digital Data Record Complete at - 2017-
06-12 16:00:03

INFO:Gateway:1207:Enter in Process Mode-Analog

INFO:Gateway:1310:Processing Analog Data

DEBUG:Gateway:1400:Connection to: 10.0.0.134  establish

DEBUG:Gateway:1410:Sending data request

DEBUG:Gateway:1411:Receiving data

DEBUG:Gateway:1403:Sending confirmation

DEBUG:Gateway:1412:Processing data string

DEBUG:Gateway:1402:Closing socket

DEBUG:Gateway:1322:Joining data together

INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS
connection (1): 10.0.0.201

DEBUG:requests.packages.urllib3.connectionpool:"POST /api/readings
HTTP/1.1" 200 45

DEBUG:Gateway:1020:Analog Reading successfully sent

DEBUG:Gateway:1208:Process and Analog Data Record Complete at - 2017-
06-12 16:00:39

DEBUG:Gateway:1122:>>>>>>>>>>     Main Interrupts finish

DEBUG:SDcard:1500:Analyzing sdCard, 0 files detected
```

Logs *Gateway* 2

```
DEBUG:Gateway:Logs Created

INFO:root:Attempting to connect websocket.
```

```
INFO:root:Connection established

INFO:Gateway:Start

INFO:Gateway:1450:Waiting socket connection

INFO:Gateway:1102:Detected event -> 2017-06-12 15:59:59

DEBUG:Gateway:1120:>>>>>>>>>>     Main Interrupts active, aquisition
will start

DEBUG:Gateway:1200:Broadcast sent at 2017-06-12 16:00:00(acquitision
timestamp)

INFO:Gateway:1201:Collecting data from Analog Boards(AI) will start

DEBUG:Gateway:1202:Collecting from Analog Board(AI) Complete -
(60,61,62,63,64,65)

DEBUG:Gateway:1204:Collecting data from all Analog Boards(AI) in the
system Complete

INFO:Gateway:1205:Doesn't exist Digital Boards(DI) in the system

INFO:Gateway:1207:Enter in Process Mode-Analog

INFO:Gateway:1310:Processing Analog Data

DEBUG:Gateway:1311:Finish data process, waiting for request

DEBUG:Gateway:1451:Connection from: ('10.0.0.249', 54240)

DEBUG:Gateway:1452:Receive request: data

DEBUG:Gateway:1460:Data Request received

DEBUG:Gateway:1461:Sending Data

DEBUG:Gateway:1462:Confirmation received: OK

INFO:Gateway:1450:Waiting socket connection
```