Universidade do Minho
Escola de Engenharia

Paulo Jorge da Silva Santos

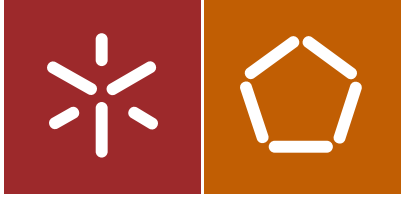
ARM Cortex M0+: Porting de Aplicações

UMinho | 2016

Paulo Jorge da Silva Santos

ARM Cortex M0+: Porting de Aplicações

outubro de 2016



Universidade do Minho
Escola de Engenharia

Paulo Jorge da Silva Santos

ARM Cortex M0+:
Porting de Aplicações

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de
Mestre em Engenharia de Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Jorge Cabral

outubro de 2016

DECLARAÇÃO

Nome:

Paulo Jorge da Silva Santos

Endereço eletrónico: paulo.jorge.silva.santos@gmail.com Telefone: 934845742

Cartão do Cidadão: 12844910 1 ZZ8

Título da dissertação:

ARM Cortex M0+: Porting de Aplicações

Orientador:

Professor Doutor Jorge Cabral

Ano de conclusão: 2016

Designação do Mestrado:

Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Universidade do Minho, _____/_____/_____

Assinatura: _____

Agradecimentos

Agradeço a todos os que, de alguma forma contribuíram para a realização deste trabalho.

Em especial ao meu orientador, professor Doutor Jorge Cabral, que propôs este tema e me deu todo apoio necessário para a concretização do mesmo.

A toda a minha família e em particular ao meu irmão Pedro Santos.

Aos meus amigos, Miguel Esteves, Rodrigo Marinho, Francine Oliveira, Vítor Fernandes, Marco Cruz, Vítor Pacheco, Romeu Gonçalves e Tiago Teixeira.

Resumo

A seleção da família de microprocessadores a utilizar numa aplicação de sistemas embebidos é determinada por um conjunto de fatores relacionados sobretudo com os periféricos necessários, a performance requerida (memória e capacidade de processamento), o consumo energético e o tempo de desenvolvimento, que estão diretamente relacionados com o custo total da aplicação. Com o aparecimento dos processadores da família *ARM* torna-se possível uniformizar uma metodologia de seleção do processador e especificar as regras de desenho a utilizar no desenvolvimento e implementação de aplicações minimalistas de sistemas embebidos.

A família *ARM Cortex* foi especificamente desenhada para aplicações de baixo consumo, baixo-custo e simplicidade de uso, tendo como aplicações alvo o controlo de motores, a automação industrial, áudio embebido e telecomunicações. A oferta de várias versões de processadores compatíveis, com diferentes dimensões de memória, periféricos e capacidade de processamento, torna possível garantir a expansão de um produto em termos de funcionalidades e prolongando assim o tempo de vida do produto. Atualmente os processadores da família *ARM Cortex M0+* são os que apresentam a maior eficiência energética de entre todos os processadores da família.

Nesta dissertação será efetuado o *porting* de uma aplicação de domótica existente desenvolvida para o controlo de janelas inteligentes, alimentadas por uma bateria e um painel fotovoltaico. Este sistema tem a capacidade de diminuir o consumo energético e de tentar manter a qualidade do ar e os níveis de luminosidade no melhor nível possível. Isto é conseguido através de um conjunto de válvulas, que controlam o fluxo de ar do exterior para o interior do edifício através da janela. E de persianas, que possibilitam o controlo da luminosidade solar que entra para o interior. Com este *porting*, pretende-se reduzir o consumo de energia, minimizar o esforço requerido no desenvolvimento de novas funcionalidades e garantir o suporte no desenvolvimento de *software* para futuras gerações do produto.

Adicionalmente, tirando partido das principais vantagens da família de processadores selecionados, desenvolveu-se um conjunto de *Applications Programming Interface standards* que permitem a interface uniformizada com os periféricos desta família de processadores. Pretende-se também caracterizar o sistema em termos de consumo energético e de performance.

Nesta dissertação demonstra-se o *porting* de uma aplicação desenvolvida para microcontroladores de oito *bits* para uma plataforma de trinta e dois *bits*, como é o *ARM Cortex M0+*, conseguindo-se diminuir o consumo energético de todo o sistema em vinte e cinco porcento.

Palavras-chave: *ARM*; *Low-Power*; *MBED*

Abstract

The selection of the family of microprocessors to use in a particular application is determined by a set of factors related mainly with the price and the development time. With the emergence of the ARM Cortex family processors it becomes possible to standardize the selection of processor and specify design rules to be used in the development and implementation of minimalist embedded systems applications.

The ARM Cortex family was specifically designed for low-power applications, low-cost and simplicity of use, targeting applications on engine control, industrial automation, embedded audio and telecommunications, among others.

The supply of multiple versions of compatible processors, with different memory sizes, peripheral sets and processing power, makes it possible to ensure the expansion of a product in terms of features. Currently the ARM Cortex M0+ processors are those with the highest energy efficiency between all the ARM processors.

The main goal is to port all the functionality of a dedicated embedded system to an ARM Cortex M0+ processor. In this dissertation, the porting will be made of an existing home automation application developed for the control of intelligent electric shutters, powered by a battery and a solar panel. This system has the ability to try to preserve the quality of air and light levels at the best possible value. This is achieved through a set of valves that control the inflow and outflow of the air in the building, and shutters which enable control of the amount of solar light that enters inside. With this porting, the intention is to reduce power consumption, to minimize the effort required to develop new features and secure the bracket in software development.

Additionally, taking advantage of the major benefits, it was developed a set of Applications Programming Interface standards that allow uniform interface with the peripherals of this family of processors. The intension is also to characterize the system in terms of energy consumption and performance.

This dissertation demonstrates the porting of an application developed for eight-bit micro-controllers to a 32-bit platform, such as ARM Cortex M0+, reducing the energy consumption in twenty-five percent.

Keywords: ARM, Low-Power, MBED

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ADC	*Analog to Digital Converter*
API	*Application Programming Interface*
CRC	*Cyclic Redundancy Check*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
FTDI	*Future Technology Devices International*
GPIO	*General Purpose Input/Output*
I2C	*Inter-integrated circuit*
IC	*Integrated Circuit*
ISP	*In-System Programmer*
LED	*Light Emitting Diode*
LSB	*Least Significant Byte*
MOSFET	*Metal Oxide Semiconductor Field Effect Transistor*
MSB	*Most Significant Byte*
NiMH	*Nickel-Metal Hydride*
PCB	*Printed Circuit Board*
RAM	*Random Access Memory*
RF	*Radio Frequency*
SCL	*Serial Clock Line*
SDA	*Serial Data Line*
UART	*Universal Asynchronous Receiver Transmitter*
UML	*Unified Modeling Language*
USART	*Universal Synchronous Asynchronous Receiver Transmitter*
USB	*Universal Serial Bus*

Capítulo I - Introdução

Cada vez mais sistemas embebidos de baixo consumo, baixo custo e com capacidade de processamento cada vez maior, designadamente sistemas alimentados por baterias e/ou energias renováveis, têm-se tornando uma exigência do desenvolvimento tecnológico. A *ARM,* para colmatar essa necessidade crescente, criou uma família de microcontroladores de trinta e dois *bits* com uma elevada eficiência energética, representando os melhores resultados, em termos de eficiência energética, alguma vez conseguidos pela *ARM*. Designada por *ARM Cortex M0+*, consome apenas nove *microwatts* e quatro décimos por *megahertz* [1], o que o torna ideal para qualquer sistema embebido que exija um compromisso entre *performance* e baixo consumo a um custo reduzido.

A família *ARM Cortex M0+* possui a vantagem de facilitar a migração de aplicações desenvolvidas em outras famílias de microcontroladores do segmento *ARM Cortex M*. Possui ainda uma grande variedade de microcontroladores com diferentes periféricos, o que possibilita a escolha do microcontrolador mais adequado às necessidades do sistema embebido.

O projeto desenvolvido no âmbito desta dissertação trata da migração de um sistema de domótica, mais concretamente um sistema de janelas inteligentes, *Climawin* [2], para a família *ARM Cortex M0+.*

O sistema de janelas inteligentes selecionado foi criado para, de forma automática, minimizar o consumo energético e tentar manter a qualidade do ar e os níveis de luminosidade no interior do edifício nos melhores valores possíveis. Isto é conseguido através de um conjunto de válvulas que controlam o fluxo do ar do exterior para o interior da habitação e de persianas que podem minimizar a exposição solar no interior. Desta forma o sistema possibilita o controlo da luminosidade solar que entra para interior do edifício e, auxiliado por um conjunto de sensores que monitorizam a temperatura, a humidade, a luminosidade e os níveis do dióxido de carbono, garantir sempre os valores ideais para o interior do edifício.

O sistema foi desenhado com o objetivo de aumentar o nível de eficiência energética de um edifício sendo todo ele alimentado através de uma bateria que é recarregada através de um painel solar fotovoltaico.

1.1. Objetivos

O principal objetivo deste projeto é fazer o *porting* de uma aplicação de domótica desenvolvida com dois processadores de oito bits, *ATTiny* da *Atmel* e *EnOcean*, para uma plataforma de baixo consumo de trinta e dois bits, *ARM Cortex M0+*.

Pretende-se ainda, para além do objetivo principal mencionado anteriormente, cumprir os seguintes objetivos:

- Diminuir o consumo de energia total do sistema;
- Criar um conjunto de *APIs* que permitam a interface uniformizada com os periféricos do *ARM Cortex M0+*;
- Criar um conjunto de *APIs* para estabelecer a interface com os vários sensores e atuadores;
- Caracterizar o sistema em termos de consumo energético e de *performance*.

1.2. Organização da Dissertação

No presente capítulo efetuou-se uma introdução e descrição dos objetivos do projeto proposto, justificando ainda a necessidade da sua realização.

No capítulo dois é feito um estudo do estado da arte sendo exibidos alguns sistemas atualmente disponíveis no mercado. É também descrita a versão do sistema na qual foi feito o *porting* para o *ARM Cortex M0+.*

O terceiro capítulo descreve a arquitetara existente e a nova arquitetura proposta.

No quarto capítulo é apresentada uma visão geral do sistema e dos seus módulos constituintes. É também descrita a forma de interação, ligações físicas e protocolos de comunicação, entre os diferentes módulos de *hardware*.

No capítulo cinco são descritos os algoritmos implementados no *ARM Cortex M0+* que controlam a persiana e as válvulas, adquirem os dados dos sensores, recebem as ordens via teclado ou rádio e gerem a energia da bateria de forma eficiente.

No capítulo seis são apresentados os resultados obtidos e uma discussão baseada na comparação entre o sistema antigo e o sistema desenvolvido em termos de consumo e de *performance* dando ênfase aos ganhos do sistema desenvolvido.

No último capítulo são apresentadas as principais conclusões, nomeadamente se todos os objetivos propostos foram atingidos e possível trabalho futuro.

Capítulo II - Análise de Mercado

Estando esta dissertação relacionada com o *porting* de uma aplicação de controlo de uma janela inteligente, irão ser abordados alguns produtos já existentes no mercado sobre janelas inteligentes. Ainda neste capítulo irá ser apresentado o sistema *Climawin*, sistema esse que está na base do projeto desenvolvido ao longo desta dissertação sendo o sistema de onde vai ser feito o *porting*.

2.1. MySmartBlinds

MySmartBlinds é um produto financiado pela empresa *Kickstarter* (empresa reconhecida por financiar projetos inovadores), que consiste num conjunto de persianas eletrónicas controladas localmente ou remotamente através de *Bluetooth*. Um dos principais focos deste produto como é ilustrado na Figura II-1 é a sua fácil instalação uma vez que é alimentado através de uma bateria, permitindo assim a sua acomodação sem ser necessário de passar cabos elétricos para alimentar a persiana.

Figura II-1: Instalação de uma *MySmartBlind.*

Para além das vantagens mencionadas anteriormente, o sistema apresenta um consumo relativamente baixo permitindo que uma carga de bateria dure entre seis a doze meses. Existe também a possibilidade de acrescentar um painel solar fotovoltaico, ver Figura II-2, para carregar a bateria eliminado a necessidade de a recarregar tornando assim o sistema mais comodo e autónomo.

Figura II-2: Painel solar da *MySmartWindow*.

O controlo local da persiana é conseguido através de uma vareta, como se pode observar na Figura II-3, que por cada puxão as abas rodam quarenta e cinco graus e se a vareta for puxada e mantida esticada o movimento é contínuo. Este método de controlo parecido com o tradicional facilita a instalação uma vez que não é necessário inserir ao lado da janela um conjunto de botões de controlo.

Figura II-3: Controlo local de uma *MySmartBlinds*.

O controlo remoto é realizado através de uma aplicação para *tablets*, *smartphones* ou *smartwatches*. Essa aplicação utiliza o *Bluetooth* do dispositivo para se conectar diretamente às persianas permitindo o controlo direto e o agendamento de tarefas. As tarefas agendadas são guardadas nas próprias persianas e executados conforme os horários e instruções introduzidos pelo utilizador. Também existe a possibilidade de mover a persiana através de um controlo remoto sem fios que pode ser colocado na parede. Na Figura II-4 apresentam-se as imagens dos diversos métodos de controlo remoto descritos.

Figura II-4: Controlo remoto de uma *MySmartBlinds* através de uma aplicação para *tablets*, *smartphones* ou *smartwatches* ou através de um controlo remoto sem fios que pode ser aplicado na parede.

Como ponto negativo estas persianas não podem ser controladas via internet estando o seu alcance limitado pelo alcance do *Bluetooth.* No entanto este ponto negativo pode-se tornar num ponto positivo pelo motivo de tronar impossível *hackear* o produto via *internet* [3].

2.2. FlipFlic

O *FlipFlic* é um dispositivo que pode ser aplicado diretamente nas persianas convencionais (sem motor e de controlo manual) sendo um método de fácil e rápida aplicação. O método consiste em retirar a vareta que controla a inclinação das abas e inserir o dispositivo no mesmo encaixe onde se encontrava a vareta. Na Figura II-5 vemos o *FlipFlic* aplicado numa persiana.

Figura II-5: *FlipFlic* aplicado numa persiana convencional.

O *FlipFlic* contém internamente um motor que faz girar o controlo da inclinação das abas, onde se encontrava vareta do controlo manual. O mecanismo de encaixe encontra-se na Figura II-6. Para além do sistema de atuação o mecanismo contém também sensores de luminosidade e de temperatura para um controlo autónomo.

Figura II-6: Mecanismo de encaixe do *FlipFlic* com a persiana.

O sistema apresenta uma aplicação para *smartphones* e *tablets* que possibilita o controlo da abertura das abas e o agendamento dos movimentos ao longo do dia. A conexão à aplicação é estabelecida através de um *hub Bluetooth* ou *ZigBee*, apresentando a capacidade de se conectar à internet e assim poder controlar as persianas em qualquer parte do mundo.

O dispositivo é também acompanhado de um painel solar fotovoltaico e um cabo *USB* para recarregar a bateria (Ver Figura II-7).

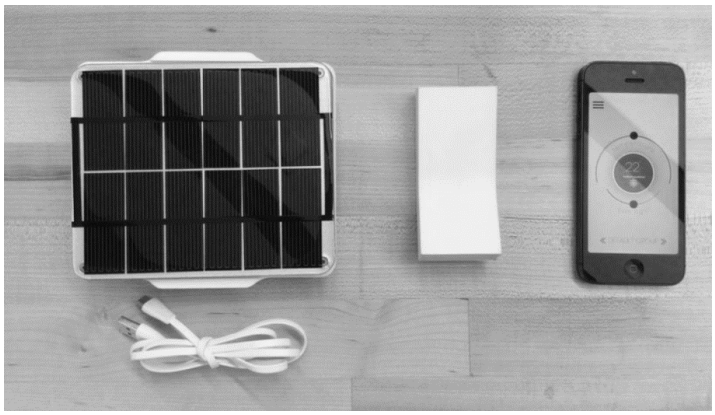


Figura II-7: Conteúdo do kit *FlipFlac.*

Como pontos negativos o dispositivo não disponibiliza um controlo na própria persiana sendo o utilizador obrigado a utilizar o *smartphone* ou o *tablet* de cada vez que pretenda mover a persiana. O dispositivo controla somente a inclinação das abas não permitindo subir nem descer a persiana [4].

2.3. Climawin

O *Climawin* vai além de apenas mover a persina, controlar a temperatura e luminosidade através da abertura das abas. Este movimenta ainda um conjunto de válvulas para controlarem o sentido da circulação do ar pela janela que em coordenação com os sensores de humidade e CO_2, melhoram a qualidade do ar no interior da habitação. Neste sistema a persiana está inserida entre dois vidros por onde circula o ar das válvulas

Figura II-8: Janela equipada com o sistema *Climawin*.

No total existem duas válvulas por janela, uma interna e outra externa. Na Figura II-9 estão representados os quatro cenários possíveis para a disposição das válvulas.

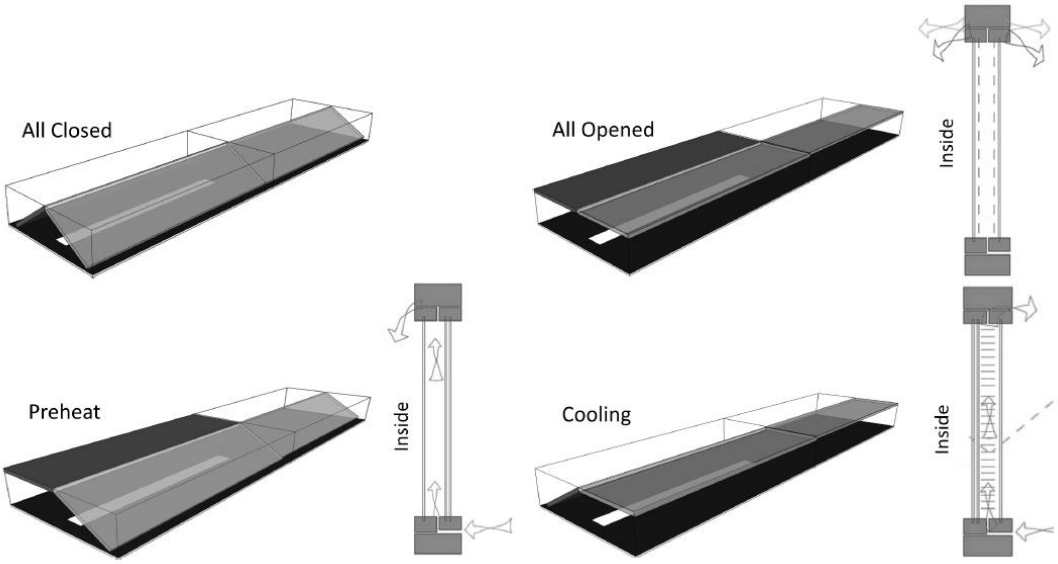


Figura II-9: Sentido da circulação do ar através da janela conforme o cenário ativo nas válvulas.

O cenário *All Closed* fecha todas as válvulas impedindo assim a circulação do ar pela janela. O cenário *All Opened* abre as duas válvulas para permitir a máxima circulação de ar. O *Preheat* mantém a válvula externa sempre fechada e fecha a interna o que obriga o ar exterior a entrar pela parte inferior da janela e a ficar retido entre os dois vidros para aquecer. Decorrido algum tempo a válvula interior abre para que o ar aquecido entre na habitação e o processo repete-se ciclicamente. Por último o cenário *Cooling* fecha a válvula interna e abre a externa para obrigar o ar exterior a entrar pela parte inferior da janela e a sair pela parte superior, mas novamente para o exterior. Esta circulação constante de ar entre os dois vidros da janela resulta no arrefecimento do ambiente dentro da habitação.

O *Climawin* dispõe também de aplicações para dispositivos móveis e uma página *web* que possibilitam o controlo da persiana e das válvulas para cada janela individualmente ou em conjunto. Este possibilita também a consulta dos valores atuais dos diferentes sensores, da carga da bateria e dos gráficos gerados com os valores da temperatura, humidade, CO_2 e luminosidade desde que o sistema foi montado até ao dia atual.

Na Figura II-10 está representada uma visão geral de todo o sistema *Climawin*.

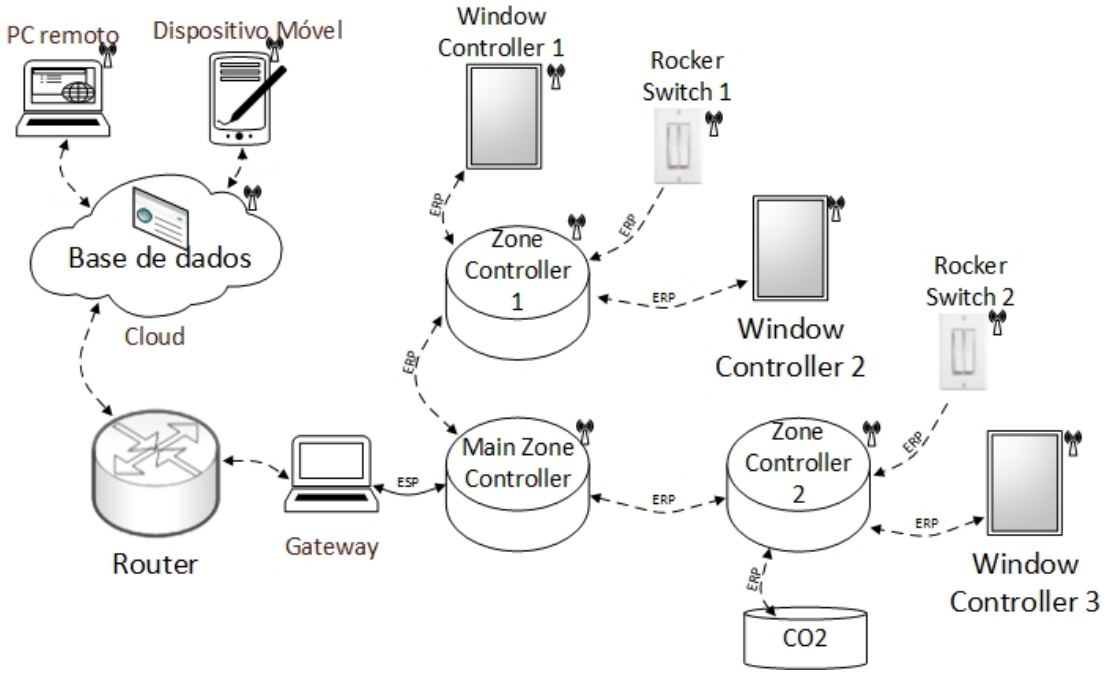


Figura II-10: Visão geral sobre todo o sistema *Climawin*.

Existem ainda interruptores sem fios designados por *Rocker Switch,* ilustrados no esquema da Figura II-10, que não necessitam de pilhas ou baterias e que são colados na parede para controlarem a posição da persiana e o ângulo de abertura das abas. Estes interruptores funcionam com a energia mecânica do movimento de pressionar [2] [5].

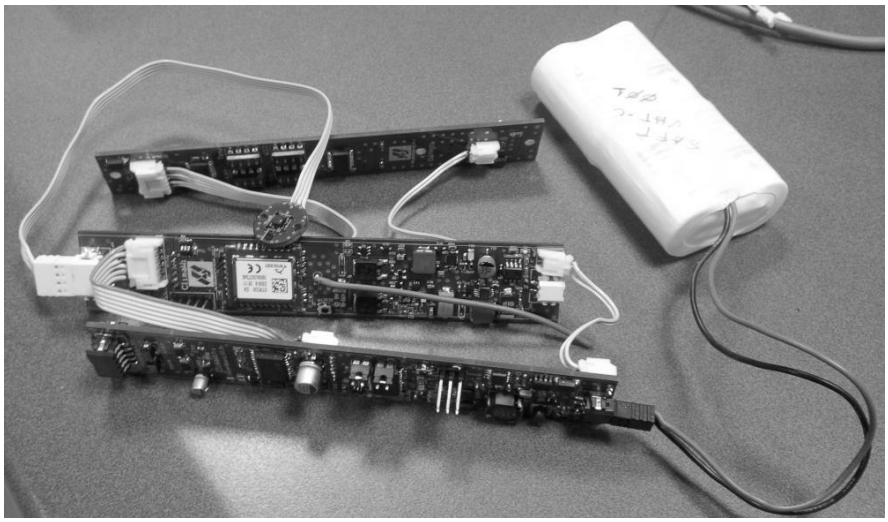


Figura II-11: *Hardware* do módulo da janela do *Climawin*.

Capítulo III - Arquitetura anterior ao *porting* e proposta

Ao longo deste capítulo será apresentada em maior detalhe a solução que o projeto desta dissertação intenta substituir. Vão ainda ser destacados os pontos em que o sistema desenvolvido supera o sistema anterior.

3.1. Arquitetura anterior ao *porting*

O sistema anterior caraterizava-se por se encontrar sobredimensionado, isto é, os dois controladores presentes na sua arquitetura estavam no máximo das suas capacidades. O módulo *EnOcean* [6] é um módulo disponível no mercado para transferir dados via rádio, sendo constituído por um microcontrolador da família 8051 e um *transceiver* de *RF*. Este módulo foi projetado para processar apenas os pacotes rádio, contudo, este microcontrolador está responsável por grande parte do controlo das janelas.

A Figura III-1 ilustra a arquitetura geral do sistema anterior ao *porting*. Pode-se observar a existência de dois blocos principais: o bloco responsável pelo controlo e pela comunicação (*EnOcean*) e o módulo responsável pela atuação das válvulas e da persiana (*ATTiny*). Genericamente não apresenta grandes diferenças estruturais em relação ao sistema desenvolvido sendo que o trabalho desenvolvido incidiu na escolha de um microcontrolador mais adequado e no desenvolvimento de um algoritmo de controlo mais eficiente.

A comunicação efetuada entre os dois módulos do sistema é realizada através do barramento *I2C*, em que o módulo *EnOcean* é o *master* e o *ATTiny* e os sensores os *slaves*.

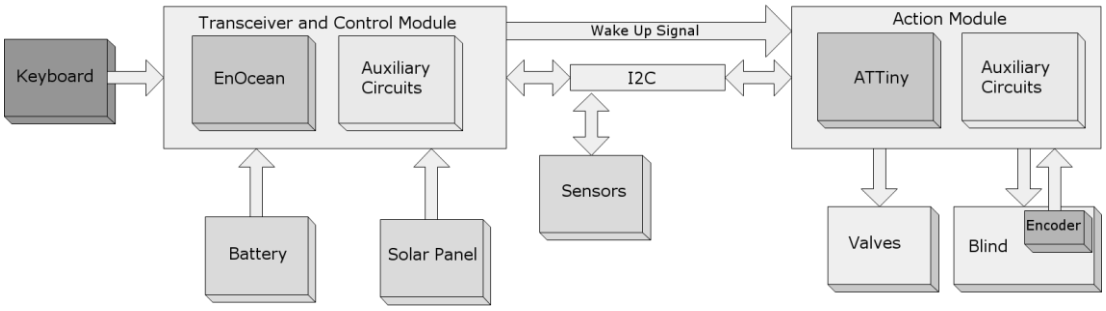


Figura III-1: Arquitetura geral anterior ao *porting*.

3.1.1 Tarefas desempenhadas pelo *EnOcean*

Como foi referido anteriormente o módulo *EnOcean* encontra-se no limite das suas capacidades de processamento uma vez desempenha as tarefas seguintes:

- Monitorizar a carga da bateria;
- Verificar as teclas pressionadas no teclado;
- Atuar os *LEDs*;
- Processar as comunicações rádio;
- Interpretar os pacotes recebidos via rádio;
- Adquirir os valores de luminosidade, temperatura e humidade relativa dos sensores;
- Controlar o módulo de atuação.

Esta sobrecarga de processamento no módulo *EnOcean* estava na base de falhas do sistema. Um exemplo destas falhas é o facto de que, se o algoritmo de controlo se encontrasse a atuar os *LEDs*, o sistema não conseguia dar resposta aos inputs do teclado. Face a esta situação era imperativo libertar este módulo de tarefas que não fossem exclusivamente de comunicação, uma vez que este foi desenhado para comunicações rádio.

3.1.2 Tarefas desempenhadas pelo *ATTiny*

O segundo microcontrolador, *ATTiny*, apesar de apenas atuar sobre as válvulas e sobre a persiana, também se encontrava no limite das suas capacidades, uma vez que o controlo da persiana exigia algum poder de processamento para garantir um movimento preciso. De forma genérica este microcontrolador apresenta as seguintes tarefas:

- Atuar as válvulas;
- Atuar sobre o motor da persiana;
- Controlo preciso da posição da persiana através da contagem dos pulsos do encoder.

3.2. Arquitetura proposta

Na Figura III-2 é apresentada a arquitetura do sistema desenvolvido neste projeto. Esta é semelhante à arquitetura apresentada anteriormente, pois também é constituída por dois módulos, no entanto o módulo *ATTiny* foi substituído pelo *ARM*.

As principais diferenças entre as duas arquiteturas são a inclusão do *ARM* no sistema e as tarefas realizadas pelos módulos do sistema. Note-se que a comunicação entre os módulos é realizada através do periférico *UART*.

O módulo *EnOcean* tem como função o processamento e interpretação da comunicação via rádio. No entanto, este também envia um sinal através de porto do *GPIO* para o *ARM* "acordar" e entrar em modo de receção. De seguida o *EnOcean* envia o comando via *UART* para o *ARM* que por sua vez trata da sua execução.

O módulo *ARM* tem como função o controlo e a atuação das válvulas e persiana.

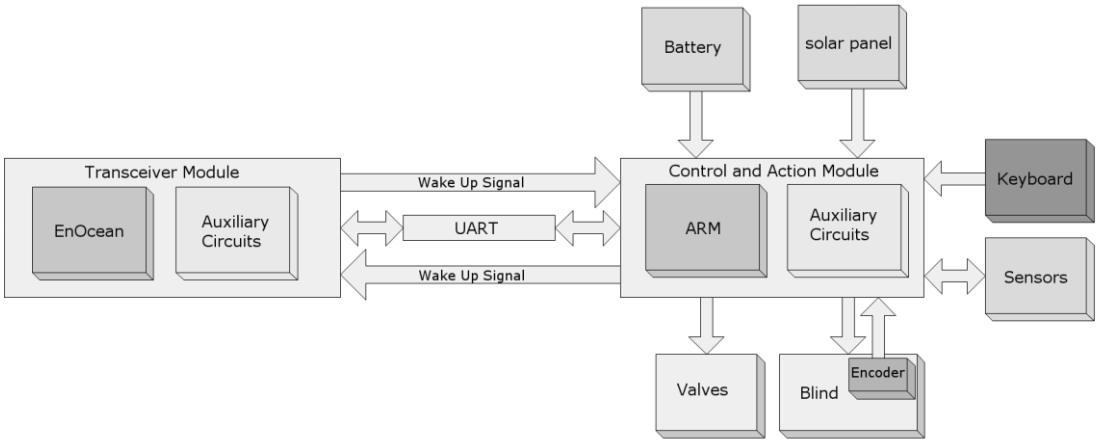


Figura III-2: Arquitetura proposta.

Tendo em conta todos os problemas apontados ao sistema anterior, decidiu-se que o microcontrolador *ATTiny* teria que ser substituído por outro que, para além de o substituir teria também de retirar ao controlador *EnOcean* todas as tarefas que não pertencessem a comunicações. Pela Figura III-2 é possível visualizar que na posição do *ATTiny* tem-se agora o microcontrolador da família *ARM*. Este microcontrolador representa uma grande melhoria para o sistema uma vez que apresenta uma capacidade de processamento muito superior, com um consumo muito inferior (nove *microwatts* e quatro décimos por *megahertz* [1]). Para essa elevada capacidade de processamento, contribui o facto de que o processador ser de trinta e dois *bits* e ter a capacidade de realizar multiplicações, divisões e cálculos com vírgula flutuante por *hardware*.

Outro aspeto importante deste microcontrolador são os seus periféricos internos, sendo os mais importantes para o projeto os timers e o *ADC*. Os *timers* têm a capacidade de se conectarem diretamente ao *encoder* e contar os impulsos por *hardware*. Desta forma diminui-se a complexidade do *software* e liberta-se tempo de processamento. O *ADC* tem a capacidade de funcionar como *Analog Window Watchdog* [7]. Assim consegue-se, através de *hardware*, monitorizar o consumo do motor da persiana. São estipulados valores de *threshold* máximo e mínimo e se algum destes valores for ultrapassado é desencadeada uma interrupção. O código de

resposta à rotina de interrupção, corta a alimentação ao motor, garantindo que este não consuma valores anormais.

3.2.1 Tarefas desempenhadas pelo *EnOcean*

Com a adição do microcontrolador *ARM* o módulo *EnOcean* viu reduzida a sua lista de tarefas estando agora encarregue do processamento e interpretação das comunicações rádio. De forma mais concreta o *EnOcean*:

- Lida com as comunicações rádio;
- Interpreta os pacotes recebidos via rádio;
- Comunica os comandos recebidos via rádio que para o *ARM* através da *UART*.
- Envia via rádio as informações enviadas pelo *ARM* via *UART*.

3.2.2 Tarefas desempenhadas pelo *ARM*

No sistema desenvolvido o microcontrolador *ARM* é considerado o cérebro de todo o sistema. É neste módulo que o controlo do sistema é realizado. Face às diferenças em relação ao sistema anterior, o *ARM* está responsável pela execução das tarefas apresentadas de seguida:

- Receber inputs do teclado e executar as ações correspondentes;
- Atuar os *LEDs*;
- Controlar o motor da persiana;
- Contar os pulsos do *encoder* para saber a posição da persiana;
- Controlar as válvulas;
- Ler os sensores;
- Monitorizar o estado da bateria.

Das características mais importantes que foram introduzidas no sistema desenvolvido destaca-se a diminuição de falhas relacionadas com a baixo poder de processamento dos microcontroladores face às exigências do *software*, bem como a implementação de um sistema de segurança que garante a proteção do motor no caso de sobrecargas. De uma forma mais direta destaca-se ainda a diminuição do consumo e um aumento do poder de processamento.

Capítulo IV - Implementação do *Hardware*

No capítulo seguinte são abordados dois temas: a arquitetura do sistema e a interação entre os componentes do sistema. Na arquitetura do sistema, é apresentada uma visão global do sistema desenvolvido, bem como os módulos de *hardware* envolvidos. Na interação dos componentes do sistema, são descritos os protocolos de comunicação entre os principais módulos.

4.1. Arquitetura do Sistema

Na Figura IV-1 encontra-se ilustrado o *overview* geral do sistema. Nesta pode-se observar os diversos sub-módulos desenvolvidos e as principais ligações entre si.

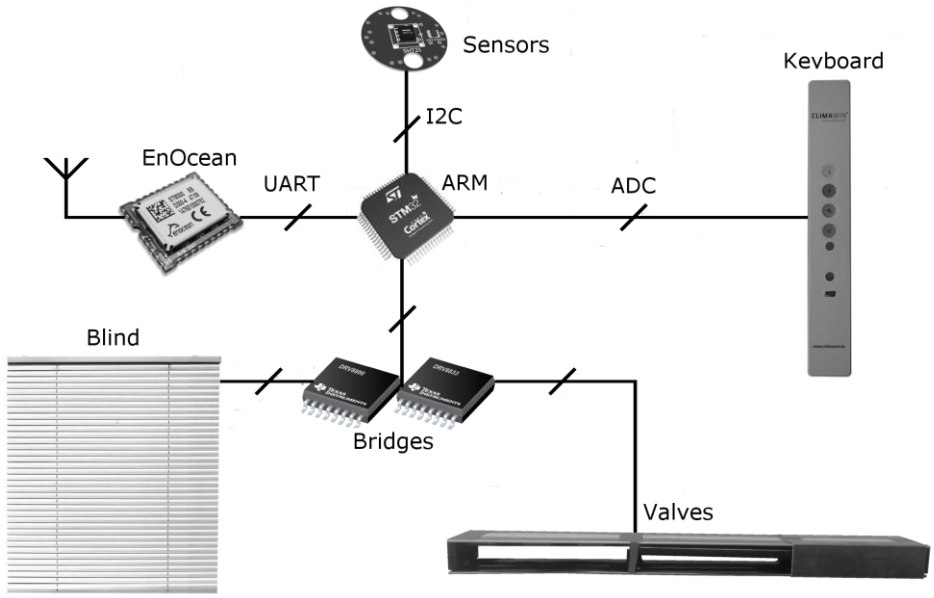


Figura IV-1: *Overview* geral do sistema.

O módulo *EnOcean* utiliza a designação de *SoC* (core *MCS-51* com *transceiver RF* integrado). por onde passam todas as comunicações de controlo da persiana e das válvulas, bem como todos os dados recolhidos pelos diversos sensores.

O módulo *Keyboard* é o teclado que permite ao utilizador controlar localmente a posição das persianas, a rotação das abas, trocar os cenários a serem aplicados às válvulas, entre outras funções. Para uma descrição mais detalhada de todas a funcionalidades acessíveis através do teclado, consultar o Anexo I. Neste módulo o reconhecimento das teclas pressionadas é feito

através de um divisor resistivo, sendo o valor da tensão resultante do divisor resistivo lido através do *ADC* do *ARM*.

O módulo *Bridges* são as pontes para a movimentação das persianas e das válvulas.

Os módulos *Blinds* e *Valves* representam as persianas e as válvulas, respetivamente.

O módulo *Power* é a fonte de alimentação de todo o sistema, sendo possível ligar e desligar individualmente a alimentação de dois grupos de sistemas, os de potência e os de controlo, permitindo assim aumentar ainda mais a autonomia da bateria.

O módulo *ARM* é considerado o "cérebro" de todo o sistema. Este módulo comunica com o módulo *EnOcean* através da *UART* e seguindo s respetivo protocolo de comunicação que é descrito numa fase posterior do capítulo.

Por fim, a comunicação para a configuração e aquisição dos dados das leituras dos sensores é realizada através de um único canal de comunicação *I2C*.

4.1.1 Atuação da persiana e das válvulas

A atuação das persianas e das válvulas é efetuada através das pontes eletrónicas *DRV8801PWP* [8] e *DRV8833PWPR* [9], respetivamente. Estas estão presentes no módulo *Bridges* da Figura IV-1, sendo controladas pelo *ARM.*

O esquemático da ponte da persina encontra-se na Figura IV-2 onde se pode observar a existência de duas linhas de controlo, *ENABLE* e *PHASE.*

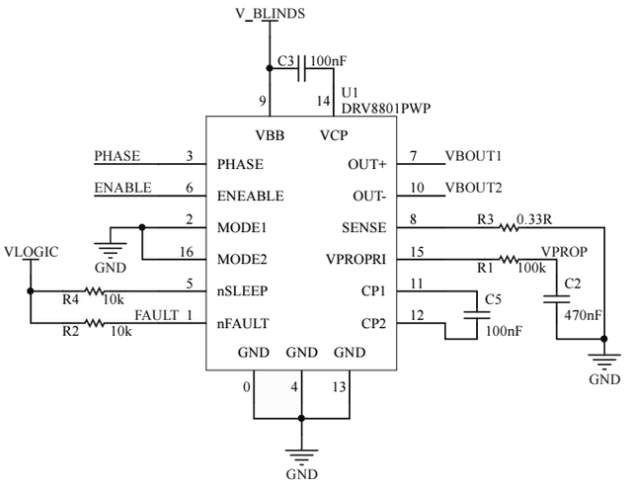


Figura IV-2: Esquemático do controlador da persiana.

Nesta figura pode-se observar uma linha designada por *VPROP* que é conectada a uma das entradas *ADC* do *ARM*. Essa linha possui uma tensão que é proporcional à corrente que circula nos enrolamentos do motor. Esta é obtida a partir da (IV-1 [8].

$$VPROPI = 5 \times I \times R3 \qquad \textbf{(IV-1)}$$

O *I* corresponde à corrente que circula nos enrolamentos do motor em amperes, *R3* ao valor da resistência *R3* no esquemático da Figura IV-2 em *ohms* e *VPROPI* é o valor da tensão obtida em *volts*. O propósito desta linha é informar o *ARM* da quantidade de corrente que circula pelo motor da persiana. Desta forma o *ARM* tem a capacidade de prevenir correntes excessivas nos enrolamentos do motor e detetar o fim de curso, uma vez que existe um interruptor de fim de curso que interrompe a circulação de corrente quando ativo.

A linha *ENABLE* quando habilitada, estado lógico "1", ativa a alimentação do motor da persiana com a polaridade definida pelo estado da linha *PHASE*. Se a linha *PHASE* estiver no estado lógico "1", a persiana move-se de forma ascendente, se estiver no estado lógico "0" a persiana move-se de forma descendente.

As linhas *VBOUT1* e *VBOUT2* que alimentam o motor da persiana estão ligados aos pinos um e dois do conector presente na Figura IV-3, que por sua vez conectam-se ao motor. Os pinos quatro e cinco estão ligados ao *encoder* da persiana, caso este exista. O pino cinco é a alimentação de cinco volts para o *encoder* e o pino quatro é o sinal de saída.

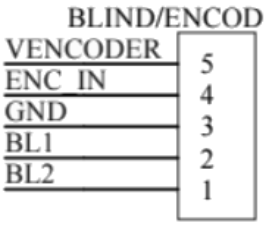


Figura IV-3: Conetor da persiana.

Na Figura IV-4 apresenta-se o circuito que conecta a saída do *encoder* a um pino do *ARM* configurado como entrada de *clock* para um dos seus *timers*. Com esta configuração consegue-se obter o máximo de aproveitamento da estrutura dos timers do *ARM*. Desta forma, a posição da persiana é constantemente atualizada por *hardware*, existindo somente interrupções, chamadas ao *software* quando a persiana atingir a posição pretendida ou caso atinja o limite máximo definido para parar o seu movimento.

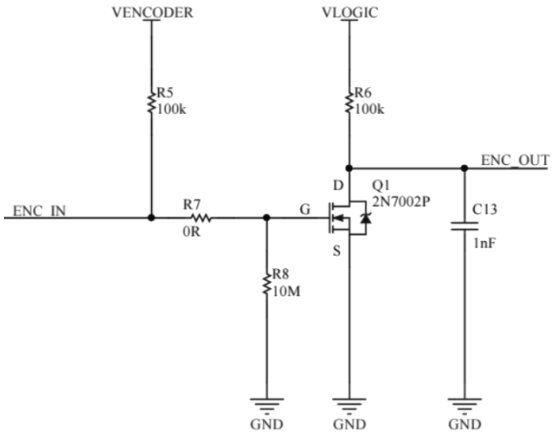


Figura IV-4: Circuito de ligação do encoder ao *ARM.*

Na Figura IV-5 apresenta-se o esquemático do controlador das válvulas. Neste pode-se observar duas linhas de controlo *S1_IN1* e *S1_IN2*. Estas linhas controlam as saídas *S1A* e *S1B* da ponte *H*. As saídas da ponte *H* conectam-se por sua vez, através de um conector às bobinas das válvulas [9].

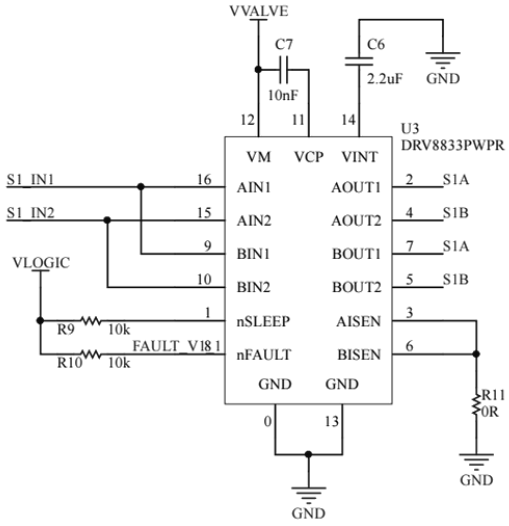


Figura IV-5: Esquemático do controlador da válvula.

Na Tabela IV-1 pode-se visualizar a lógica entre as entradas e as saídas com a sua respetiva função.

Tabela IV-1: Lógica da ponte H que atua sobre as válvulas

S1_IN1	S1_IN2	S1A	S1B	Função
L	L	Z	Z	Sem atuação
L	H	L	H	Fechar válvula
H	L	H	L	Abrir válvula
H	H	L	L	Sem atuação

Cada conjunto contém duas válvulas, uma interior e outra exterior. O esquemático do controlador da Figura IV-5 mais a lógica apresentada na Tabela IV-1 são para uma única válvula, sendo para duas válvulas o processo idêntico, mas duplicado.

4.1.2 Sensores de temperatura, humidade e luminosidade

O conjunto dos sensores é composto por dois *Integrated Circuits (IC)*, o *MAX44009* e o *SHT21*. Estes estão ligados ao *ARM* através de um único canal de comunicação *I2C*.

O *SHT21* é responsável pela medição da temperatura em graus centigrados e da humidade relativa em percentagem.

O *MAX44009* por sua vez está encarregue da medição dos valores de luminosidade em *LUX*.

Na Figura IV-6 apresenta-se o esquemático do circuito dos sensores *SHT21* e *MAX44009*.

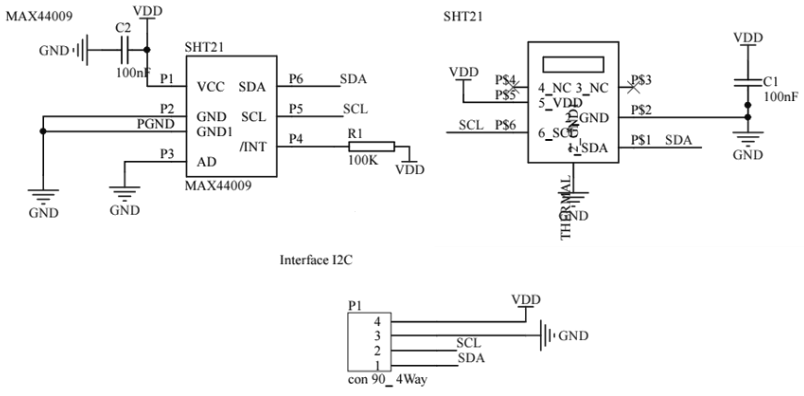


Figura IV-6: Esquemático do circuito dos sensores.

A comunicação com o *SHT21* segue o protocolo *I2C*. O primeiro *byte* transmitido corresponde ao cabeçalho *I2C* que é constituído por sete *bits* do endereço do periférico mais um *bit* de direção, onde zero indica escrita e um indica leitura. No caso do *STH21* o seu endereço é "1000000" em binário. Após ter sido enviado o cabeçalho o *byte* seguinte segundo o *datasheet* do componente [10] é o comando.

Na Tabela IV-2 são apresentados todos os comandos possíveis, nomeadamente, comandos para a aquisição da temperatura, comandos para a aquisição da humidade relativa, comandos para a leitura e escrita do registo do utilizador (registo onde constam as configurações do *STH21*) e um comando de *reset*.

Tabela IV-2: Lista de comandos do SHT21

Comando	Modo	Função
1110 0011	Segurar mestre	Leitura da temperatura
1110 0101	Segurar mestre	Leitura da humidade relativa
1111 0011	Não segurar mestre	Leitura da temperatura
1111 0101	Não segurar mestre	Leitura da humidade relativa
1110 0110		Escrever registo do utilizador
1110 0111		Ler registo do utilizador
1111 1110		Reset

Na Tabela IV-2 constata-se a existência de dois modos de aquisição da temperatura e da humidade relativa, sendo estes, "segurar mestre" e "não segurar mestre".

No primeiro modo, "segurar mestre", a linha *SCL* do *I2C* fica bloqueada (controlada pelo sensor) durante todo o processo de medição, sendo esta libertada quando a medição for enviada ao microcontrolador.

No modo "não segurar mestre" a linha *SCL* fica desbloqueada no processo de medição permitindo que outros periféricos comuniquem com o microcontrolador enquanto o *STH21* executa o processo de medição. Neste modo, o microcontrolador tem de periodicamente comunicar com o *STH21* para saber se o sensor já efetuou a medição e se esta está pronta a ser enviada. Se a medição já tiver sido efetuada o *STH21* envia-a na próxima comunicação com o microcontrolador.

Na Figura IV-7 e na Figura IV-8 pode-se visualizar a comunicação do microcontrolador com o *STH21* para a obtenção da humidade relativa, nos modos "segurar mestre" e "não segurar mestre" respetivamente.

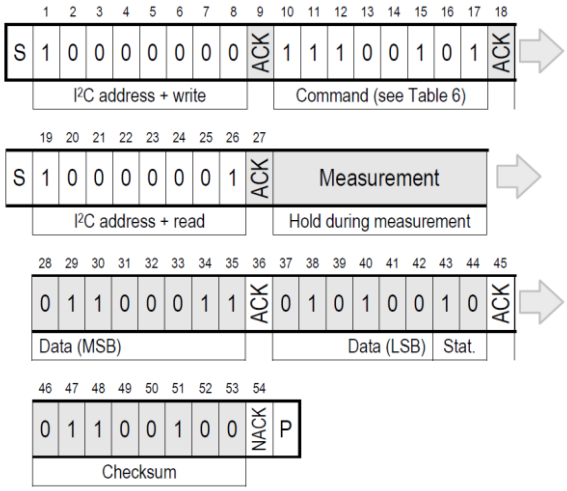


Figura IV-7: Comunicação *I2C* com o *SHT21* no modo "segurar mestre".

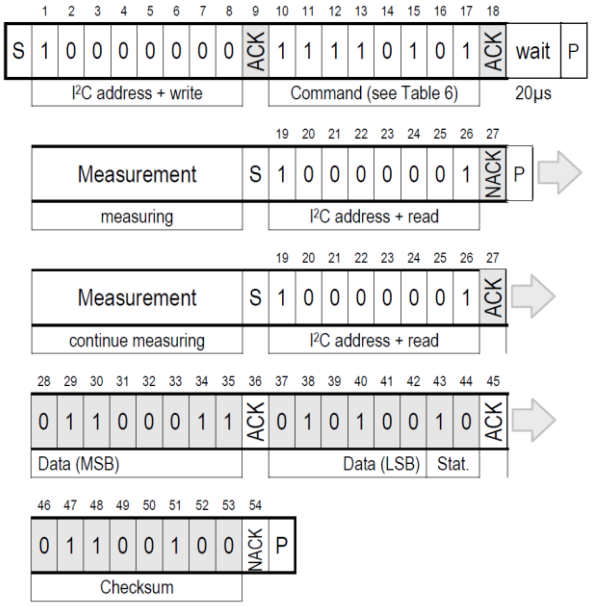


Figura IV-8: Comunicação *I2C* com o *SHT21* no modo "não segurar mestre".

Na Figura IV-9 tem-se uma comunicação *I2C* onde microcontrolador lê e escreve o registo do utilizador do *STH21*. Neste registo é possível, entre outras configurações, configurar as resoluções das medições, da temperatura e da humidade relativa. Quanto maior for a resolução maior é o tempo de medição.

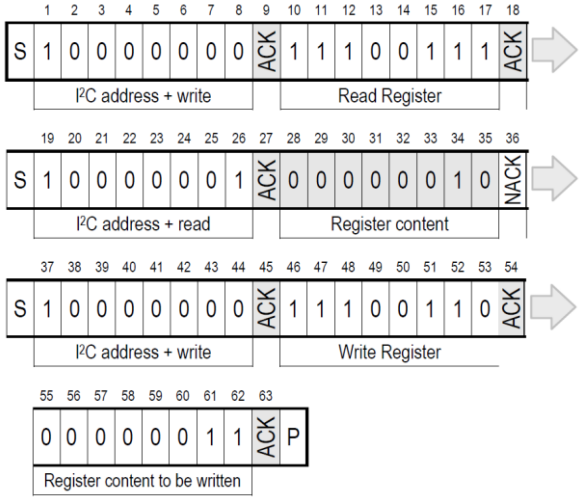


Figura IV-9: Comunicação *I2C* com o *STH21*. Leitura e escrita do registo do utilizador.

Na Figura IV-10 tem-se o comando de *reset* que reinicia o *STH21*.

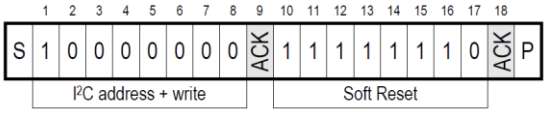


Figura IV-10: Comunicação *I2C* com o *STH21* para efetuar o *reset*.

Na Tabela IV-3 temos os tempos de medição máximos e médios para cada resolução.

Tabela IV-3: Tempos de medição do *STH21*.

Resolução	Humidade Típico	Humidade Máximo	Temperatura Típico	Temperatura Máximo	Unidades
14 bits			66	85	ms
13 bits			33	43	ms
12 bits	22	29	17	22	ms
11 bits	12	15	9	11	ms
10 bits	7	9			ms
8 bits	3	4			ms

Para a conversão dos catorze *bits*, adquiridos da medição, temos a (IV-2 para a temperatura em graus Celsius e a (IV-3 para a humidade relativa em percentagem. Onde R representa os catorze *bits* adquiridos da medição.

$$Temp = -46{,}85 + 175{,}72 \times \frac{R}{2^{14}} \quad \textbf{(IV-2)}$$

$$HR = -6 + 125 \times \frac{R}{2^{14}} \quad \textbf{(IV-3)}$$

O *MAX44009* é o sensor de luminosidade com um alcance que vai desde os quarenta e cinco milésimos de *Lux* aos cento e oitenta e oito mil *Lux*. A sua comunicação, tal como no *STH21*, é feita através do *I2C*.

Na Tabela IV-4 são apresentados os dois registos onde são guardadas as medições de luminosidade [11].

Tabela IV-4: Registos onde estão os valores de luminosidade medidos.

Registo	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Endereço
Lux Byte mais significativo	E3	E2	E1	E0	M7	M6	M5	M4	0x03
Lux Byte menos significativo					M3	M2	M1	M0	0x04

A (IV-4 converte os valores dos registos de luminosidade em *Lux*.

$$Lux = 2^{8\times E3+4\times E2+2\times E1+E0} \times (128 \times M7 + 64 \times M6 + 32 \times M5 + 16 \times M4 + 8 \times M3 + 4 \times M2 + 2 \times M1 + M0) \times 0{,}045 \quad \textbf{(IV-4)}$$

4.1.3 Teclado

A leitura do teclado é feita através de um *ADC* onde o valor da tensão representa a tecla ou teclas pressionadas.

Na Figura IV-11 encontra-se o esquemático do circuito que gera as tensões de entrada do *ADC* conforme as teclas pressionadas. Este trata-se de um simples divisor de tensão resistivo onde o valor da primeira resistência é fixo com o valor de vinte mil *ohms* e o valor da segunda é o resultado do paralelo de *R8*, *R9*, *R10* e *R12* que são inseridas ou retiradas do circuito conforme a tecla esteja pressionada ou não. A linha de interrupção *WAKE_INT* serve para "acordar" o *ARM* caso uma ou mais teclas tenha sido premida.

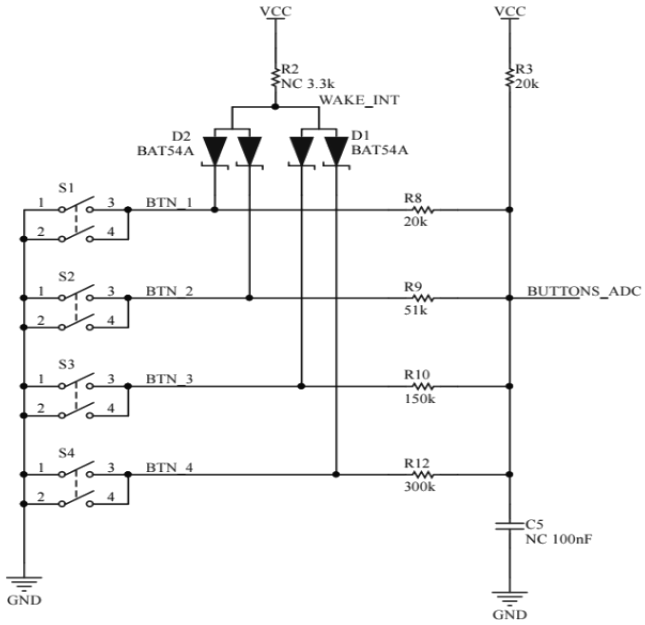


Figura IV-11: Esquemático do teclado.

Na (IV-5 apresenta-se o cálculo do valor da tensão de entrada no *ADC* onde R_P representa o valor resultante do paralelo.

$$BUTTONS\_ADC = 3{,}3 \times \frac{R_P}{20000 + R_P} \qquad \textbf{(IV-5)}$$

Na Tabela IV-5 estão os valores de tensão calculados e medidos para cada tecla ou combinação de teclas.

Tabela IV-5: Valores de tensões no ADC para cada combinação de teclas

Botão 4	Botão 3	Botão 2	Botão 1	Tensão ADC calculado	Tensão ADC medido
0	0	0	1	1648,4	1646,5
0	0	1	0	2367,0	2365
0	0	1	1	1378,4	1378
0	1	0	0	2906,6	2901,7
0	1	0	1	1545,4	1544,7
0	1	1	0	2160,4	2160,2
0	1	1	1	1304,6	1305,6
1	0	0	0	3088,0	3085,8
1	0	0	1	1595,2	1594,8
1	0	1	0	2259,0	2259,5
1	0	1	1	1339,9	1341,5
1	1	0	0	2745,4	2745,1
1	1	0	1	1497,3	1498,9
1	1	1	0	2067,5	2071,4
1	1	1	1	1270,1	1272,9

4.1.4 *Transceiver de rádio*

O módulo do *transceiver* [6] é apresentado na Figura IV-12 onde se pode observar o seu esquemático.

A comunicação com o *ARM* é efetuada através de um canal *USART* bidirecional. Existem também mais duas linhas, uma para o *EnOcean* "acordar" o *ARM* e outra para o *ARM* "acordar" o *EnOcean*. Estas duas linhas têm como função garantir que no momento em que a comunicação começa através do canal *UART,* os dois lados estejam acordados e prontos a comunicar.

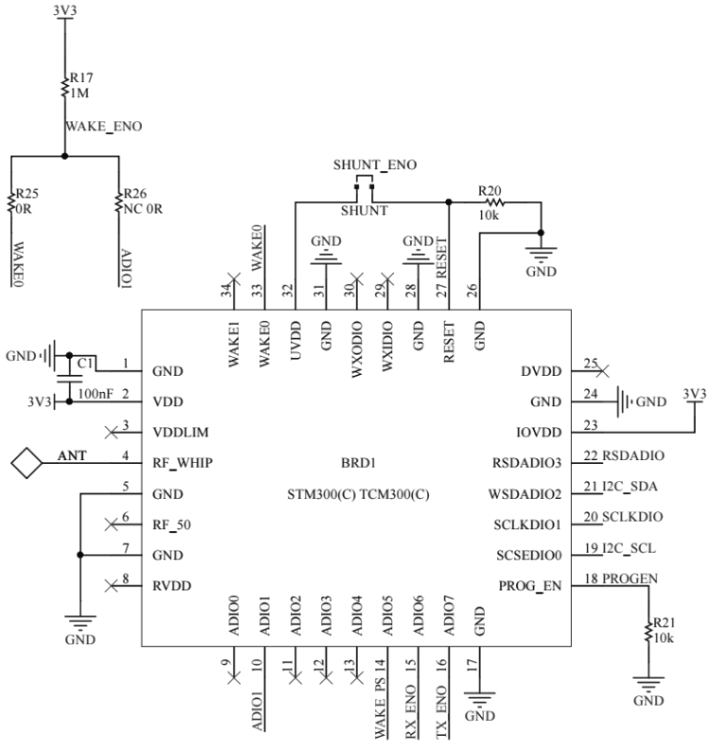


Figura IV-12: Esquemático do circuito do *transceiver*.

4.1.5 Fonte de alimentação

A fonte de alimentação, para além de fornecer a energia necessária para o bom funcionamento dos circuitos, também oferece a possibilidade do *ARM* poder desligar dois grandes conjuntos de circuitos individualmente para aumentar a autonomia da bateria. Estes correspondem aos circuitos lógicos e aos circuitos de atuação.

O esquemático dos circuitos que possibilitam ligar e desligar as alimentações dos circuitos lógicos de atuação individualmente encontram-se representados na Figura IV-13.

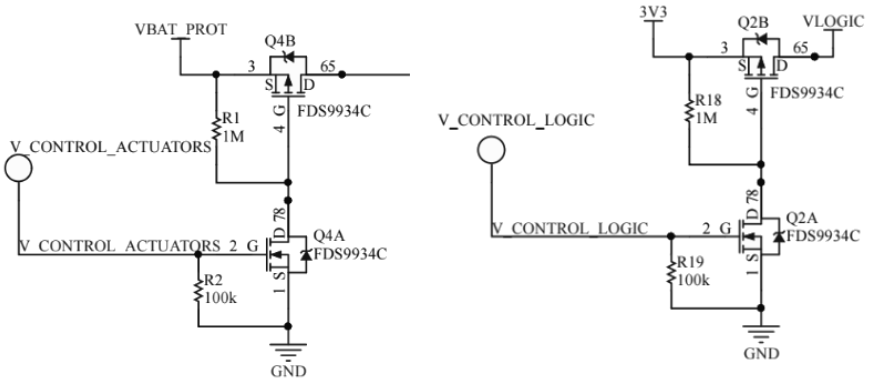


Figura IV-13: Circuitos de ativação e desativação das alimentações para os circuitos lógicos e para os circuitos de atuação.

Na Figura IV-14 encontra-se o esquemático do circuito utilizado para reduzir proporcionalmente o valor da tensão da bateria para níveis aceites pelo *ADC* do *ARM*. Devido à necessidade de aumentar a autonomia da bateria este circuito também possibilita ao *ARM* ligar este circuito (permitir a passagem de corrente palas resistências do divisor de tensão) somente quendo efetuar uma medição do nível da bateria através da ligação *BAT_EN*.

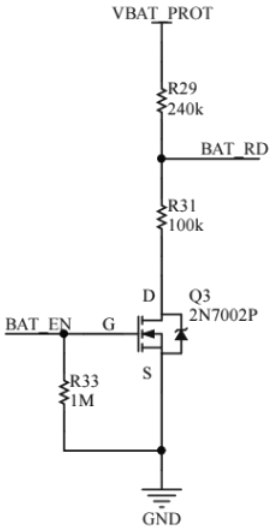


Figura IV-14: Circuito utilizado para a leitura do nível da tensão da bateria.

4.2. Interação entre os componentes do sistema

Neste subcapítulo é abordado o protocolo de comunicação desenvolvido para a comunicação entre o *ARM* e o *EnOcean*.

4.2.1 Protocolo de comunicação entre o *ARM* e o *EnOcean*

A comunicação *UART* entre o *ARM* e o *EnOcean* é estabelecida a partir do protocolo de comunicação descrito neste subcapítulo.

Este protocolo consiste num cabeçalho seguido, se existirem um ou dois *bytes* de dados. Existem três tipos de cabeçalhos, um para o *EnOcean* para pedir as configurações feitas no lado do *ARM*, designado por "*Request*" (Tabela 6), Outro para o *EnOcean* efetuar as configurações possíveis no *ARM*, designado por "*Configuration*" (Tabela 7) e por fim um para enviar comandos do *EnOcean* para o *ARM* e do *ARM* para o *EnOcean*, designado por "*Command*" (Tabela 8). Os dois primeiros cabeçalhos, "*Request*" e "*Configuration*" são seguidos de um ou dois *bytes* de dados, já o último *"Command"*, não é seguido por nenhum *byte* de dados sendo constituído unicamente pelo cabeçalho.

Tabela 6: Cabeçalho do pacote "*Request*".

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	x	ParamId4	ParamId3	ParamId2	ParamId1	ParamId0

Tabela 7: Cabeçalho do pacote "*Configuration*".

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	1	x	ParamId4	ParamId3	ParamId2	ParamId1	ParamId0

Tabela 8: Cabeçalho do pacote "*Command*".

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	0	x	x	CommandId3	CommandId2	CommandId1	CommandId0

Na Tabela 9 apresenta-se a lista de todos os parâmetros aceites pelo protocolo desenvolvido. Nesta pode-se observar os parâmetros de leitura e configuração, conforme se trate de um *"Request"* ou de um *"Configuration"*, dos níveis de temperatura, humidade relativa, luminosidade,

CO_2, posição relativa da persiana, posição relativa da abertura das abas, cenário das válvulas e nível da bateria.

Tabela 9: Descrição de todos os ID's de parâmetros aceites.

Valor	Descrição	Número de bytes de dados
0	Nível da temperatura interior	1
1	Nível do limite inferior da temperatura interior	1
2	Nível do limite superior da temperatura interior	1
3	Nível da temperatura exterior	1
4	Nível do limite inferior da temperatura exterior	1
5	Nível do limite superior da temperatura exterior	1
6	Nível da humidade relativa interior	1
7	Nível superior da humidade relativa interior	1
8	Nível da humidade relativa exterior	1
9	Nível superior da humidade relativa exterior	1
10	Nível da luminosidade exterior	2
11	Limite superior da luminosidade exterior	2
12	Nível do CO2 interior	1
13	Nível máximo do CO2 interior	1
14	Posição relativa da persiana	1
15	Posição relativa da abertura das abas	1
16	Posição da persiana mais a posição das abas	2
17	Tamanho da persiana	2
18	Modo de operação (automático manual)	1
19	Cenário das válvulas	1
20	Modo de operação mais cenário das válvulas	1
21	Estado da bateria (se a bateria esta a carregar ou não)	1
22	Nível da bateria	1
23	Estado e nível da bateria	1

Na Tabela 10 temos uma descrição de todos os comandos.

Tabela 10: Descrição de todos os ID's de comandos aceites.

Valor	Descrição	Número de bytes de dados
0	Iniciar a associação do dispositivo	0
1	Iniciar o envio do relatório periódico	0
2	Habilitar o transceiver	0
3	Desabilitar o transceiver	0
4	Ligação com a rede estabelecida	0
5	Ligação com a rede não estabelecida	0
6	Dispositivo associado	0
7	Dispositivo desassociado	0
8	Mover a persiana para cima	0
9	Mover a preciana para baixo	0
10	Mover as abas para cima	0
11	Mover as abas para baixo	0
12	Iniciar a sequencia de testes do transceiver	0
13	Resultado do sucesso da sequência de testes	0
14	Todos os LEDs piscam em simultâneo	0

4.3. Protótipo final

Como resultado final do sistema obteve-se um sistema totalmente funcional e testado em ambiente real. Neste subcapítulo são apresentadas as imagens relativas ao *hardware* do protótipo final. Na Figura IV-15, é apresentada *PCB* relativa ao teclado.

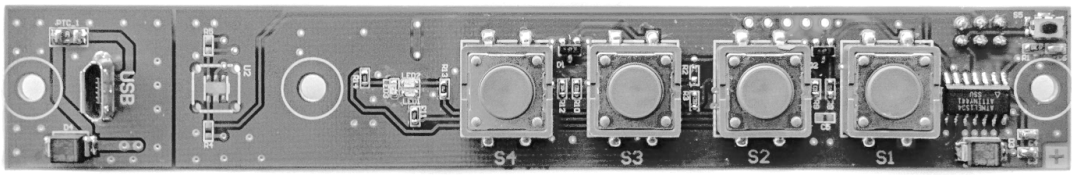


Figura IV-15: *Hardware* do Teclado.

Na Figura IV-16 é apresentada a *PCB* onde se encontra o *ARM* e o *EnOcean*.

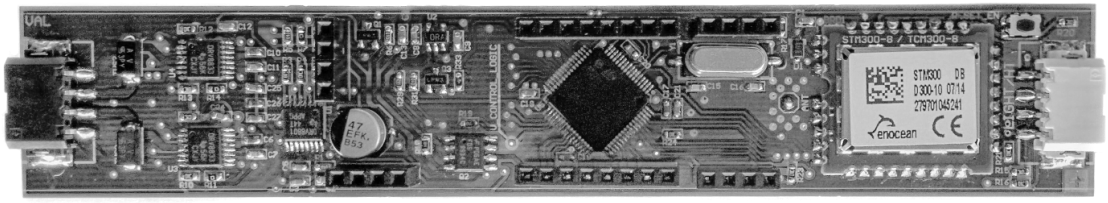


Figura IV-16: Placa principal do sistema.

Na Figura IV-17 é apresentada a fonte de alimentação. Nesta *PCB* encontram-se os circuitos de conversão da tensão de alimentação da bateria para os vinte e quatro *volts* da persiana e das válvulas e os três *volts* e três décimos do *ARM*.

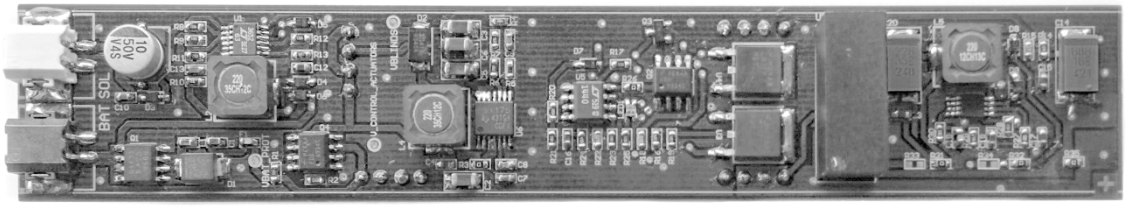


Figura IV-17: Fonte de alimentação.

Na Figura IV-18 é apresentada a ligação entre o *hardware* desenvolvido.

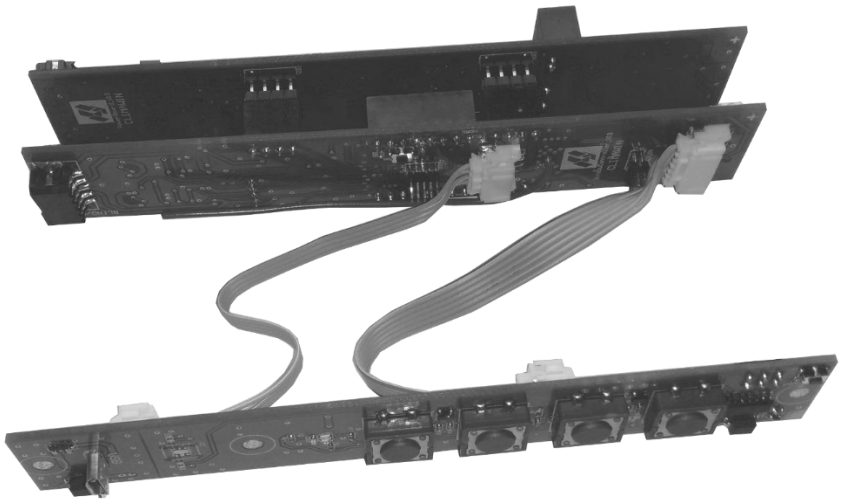


Figura IV-18: *Hardware* do sistema.

Capítulo V - Implementação *Software*

No presente capítulo é feita a apresentação da implementação do *software* cuja a finalidade é o controlo da persiana, das válvulas, dos atuadores, da aquisição dos dados dos sensores de temperatura, humidade e luminosidade, bem como, responder e, ou atuar face aos diversos comandos recebidos via rádio. É de salientar que todas estas ações de controlo, aquisição e comunicação são executadas em simultâneo.

É também descrito o funcionamento de cada um dos módulos constituintes deste *software*, nomeadamente, a interface e a ligação com os diversos módulos. A descrição dos módulos é efetuada através de diagramas de classes *UML*.

5.1. Requisitos do Sistema

O *software* implementado tem por base o microcontrolador da família *ARM Cortex M0+*, *STM32L053R8* e o *hardware* desenvolvido para o projeto. Deve-se tirar o máximo partido da nova arquitetura do microcontrolador e do novo *hardware*, de modo a que seja possível atingir um consumo menor e uma *performance* melhor quando comparado à versão antiga. Deve também incluir todas as funcionalidades do sistema desenvolvido anteriormente.

O *software* deve permitir ao utilizador controlar de forma simples a persiana a partir do teclado, mover a posição da persiana e alterar a posição das abas. Deve também fornecer ao utilizador as opções de seleção do cenário, do modo de atuação automático ou manual das válvulas, da configuração do tamanho da persiana, da associação/desassociação (*Lear In/Out*) da janela numa zona e forçar o envio de um relatório completo do estado da janela, através de um esquema de menus *user friendly*.

Para além do controlo efetuado através do teclado, o *software* também deve também possibilitar o controlo via rádio. Neste é permitido mover a persiana para uma determinada posição, mover as abas, alterar o cenário, o modo de atuação das válvulas, configurar o tamanho da persiana, adquirir os dados dos sensores de temperatura, humidade e luminosidade, ler o estado da bateria e pedir um relatório completo do estado da janela.

O relatório deve conter a posição da persiana, a posição das abas, o cenário das válvulas, o modo automático ou manual de atuação das válvulas, a temperatura exterior, a humidade relativa exterior, luminosidade exterior e o estado da bateria.

Na Figura V-1 pode-se observar o diagrama de casos de uso das funcionalidades descritas nos parágrafos anteriores deste subcapítulo.

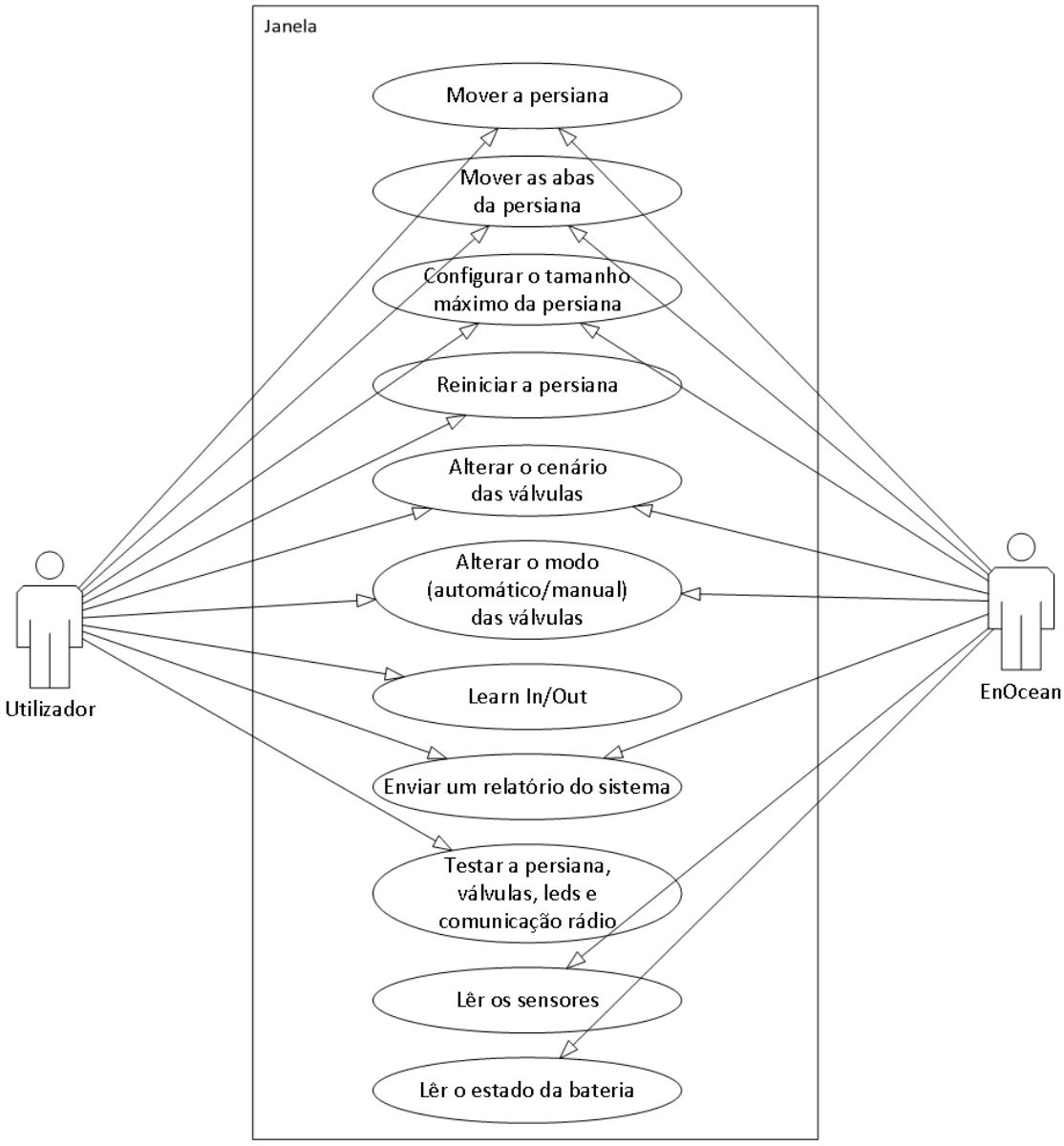


Figura V-1: Diagrama de casos do uso da janela.

5.2. Implementação do *Software*

Neste subcapítulo é descrita a estrutura geral do *software* e de cada um dos seus módulos constituintes.

Para além dos requisitos propostos o *software* irá colocar o microcontrolador na maior parte do tempo no modo de "dormir", estando apenas ativo um timer de baixo consumo e um oscilador de baixa frequência que serve de relógio para o *timer*. O microcontrolador apenas "acorda" quando ocorrer uma interrupção do timer de baixo consumo por *overflow*, uma interrupção por mudança descendente do estado do pino que indica que uma tecla do teclado foi pressionada e uma

interrupção igualmente por mudança de estado do pino ascendente que indica que o *EnOcean* irá inicializar uma comunicação via *UART* com o *ARM*.

5.2.1 Estrutura geral do *software*

O *software* do *ARM* é composto pelos seguintes módulos:

- *Blind:* este módulo é responsável pelo controlo da posição da persiana e respetivas abas;
- *Valve*: módulo que controla e aplica os cenários nas válvulas;
- *Sensors:* configura e adquire os dados dos diferentes sensores (temperatura, humidade e luminosidade);
- *Keyboard:* indica a tecla ou teclas pressionadas e o tempo de duração;
- *KeyboardMenus:* controla a navegação entre os menus e executa as respetivas ações;
- *EnOcean:* é responsável por controlar e comunicar através de um protocolo pré-estabelecido via UART com o *EnOcean*;
- *Leds:* permite piscar os *LEDs*, individualmente ou em conjunto, até quatro piscas num segundo ou regularmente em cada dois segundos;
- *Battery:* mede, através de um canal do *ADC*, e calcula a percentagem de carga da bateria. Executa também as ações necessárias quando a bateria está abaixo de vinte porcento, está a ser carregada e a carga está completa;
- *Power:* possibilita a ativação e a desativação da alimentação de dois grupos de circuitos, os de potência e os de lógica;
- *Adc:* configura, adquire e controla o acesso dos diferentes módulos ao *ADC*;
- *Eeprom:* possibilita a escrita e a leitura na memória *EEPROM* do microcontrolador;
- *Scheduler:* possibilita o agendamento de ações periódicas e executa-as no momento certo;
- *WindowController:* é o módulo principal.

Na Figura V-2 encontra-se o diagrama *UML* minimalista do *software* desenvolvido para o *ARM*, métodos e membros são apresentados ao longo das secções. Neste observa-se de forma gráfica as relações entre as diversas classes. A ligação com o losango negro numa das

extremidades indica associação. A classe do lado sem o losango negro contém objetos do tipo da classe do lado com o losango negro.

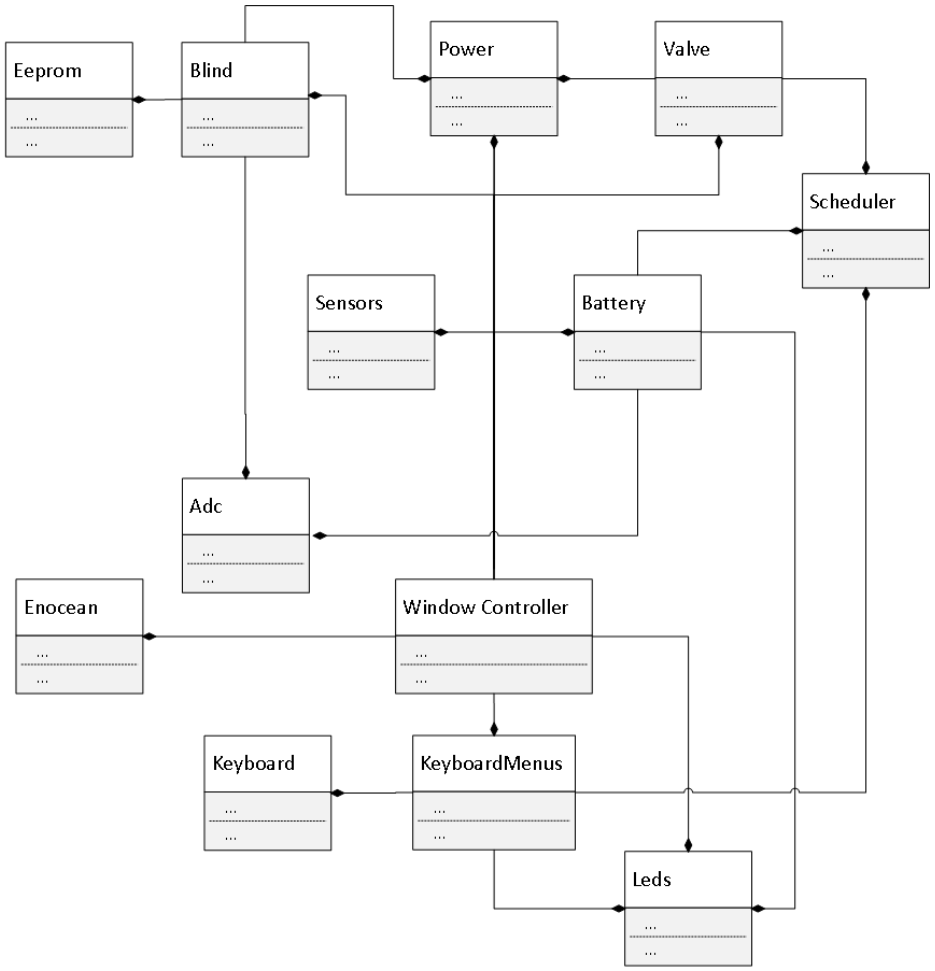


Figura V-2: Diagrama *UML* minimalista (métodos e membros são apresentados ao longo das secções).

5.2.2 Módulo *Scheduler*

O módulo *Scheduler* é responsável pelo agendamento de tarefas que necessitam de ser executadas periodicamente. Este é utilizado pelos módulos *Valve*, *Battery* e *KeyboardMenus*.

O módulo *Valve* é responsável por abrir e fechar periodicamente a válvula interior nos cenários de *preheat33* e *preheat66*.

O módulo *Battery* é responsável por fazer a leitura do estado da bateria a cada dez minutos.

Por último, o módulo *KeyboardMenus* tem como função bloquear o acesso aos menus após ter decorrido trinta minutos desde que foi desbloqueado.

Para implementar este módulo recorreu-se à utilização do periférico *LPTIM* do *STM32L053R8*. Este periférico *LPTIM* é um timer de baixo consumo de dezasseis *bits* com *auto-reload* e *compare-match*. Na Figura V-3 é possível observar o seu diagrama de blocos.

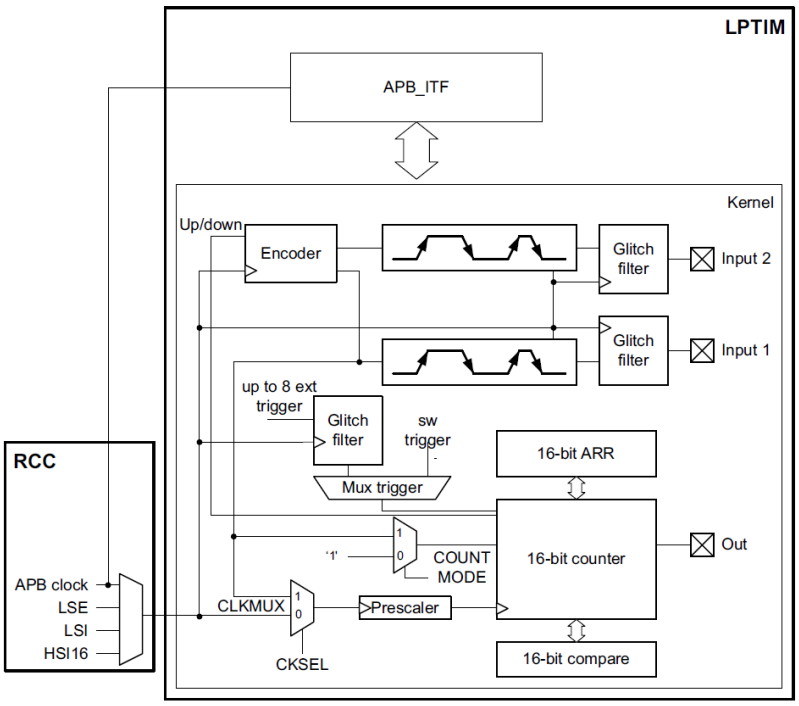


Figura V-3: Diagrama de blocos do periférico *LPTIM* do microcontrolador *STM32L053R8*.

No *software* desenvolvido foi configurado o *LPTIM* para utilizar o oscilador *LSI* que é um oscilador *RC* de trinta e sete kilohertz. Este encontra-se sempre ativo mesmo quando o microcontrolador entra no estado de "dormir". O *prescaler é* configurado para dividir a frequência do relógio em cento e vinte e oito vezes.

Devido ao facto do oscilador *LSI* não ser preciso por ser um oscilador RC, é necessário existir uma compensação. Essa compensação é feita através do auxílio de um segundo *timer* de trinta e dois megahertz, onde o seu relógio é um oscilador de cristal. Para realizar a calibração é feita uma contagem do número de ciclos do oscilador de trinta e dois megahertz que ocorreram durante oito ciclos do *LSI*. Esse valor corresponde à variável *pulse_time* utilizada na (V-1. Desta obtém-se o valor a colocar no registo *ARR*, ou seja, o valor de recarga do *timer*.

$$ARR = \frac{200 \times 32000000}{16 \times pulse\_time} \qquad \textbf{(V-1)}$$

Depois de efetuada a calibração configura-se o *LPTIM* para desencadear uma interrupção por *overflow* a cada duzentos segundos.

Na Figura V-4 é apresentado através de três fluxogramas o funcionamento do módulo *scheduler*. Neste observam-se três linhas de execução diferentes, a primeira, a do lado esquerdo, está esquematizada a inicialização e configuração do *LPTIM* e das variáveis necessárias para o seu funcionamento. No lado direito está o fluxograma que é executado quando ocorre uma interrupção. O fluxograma do centro representa a execução em ciclo infinito dentro da função *main* do *software*. Este verifica se em algum agendamento a variável *counter* atinge o valor zero, se sim, é executada a função apontada pela variável *FunctionCallback* e recarregada novamente a variável *counter* com o valor da variável *reload*.

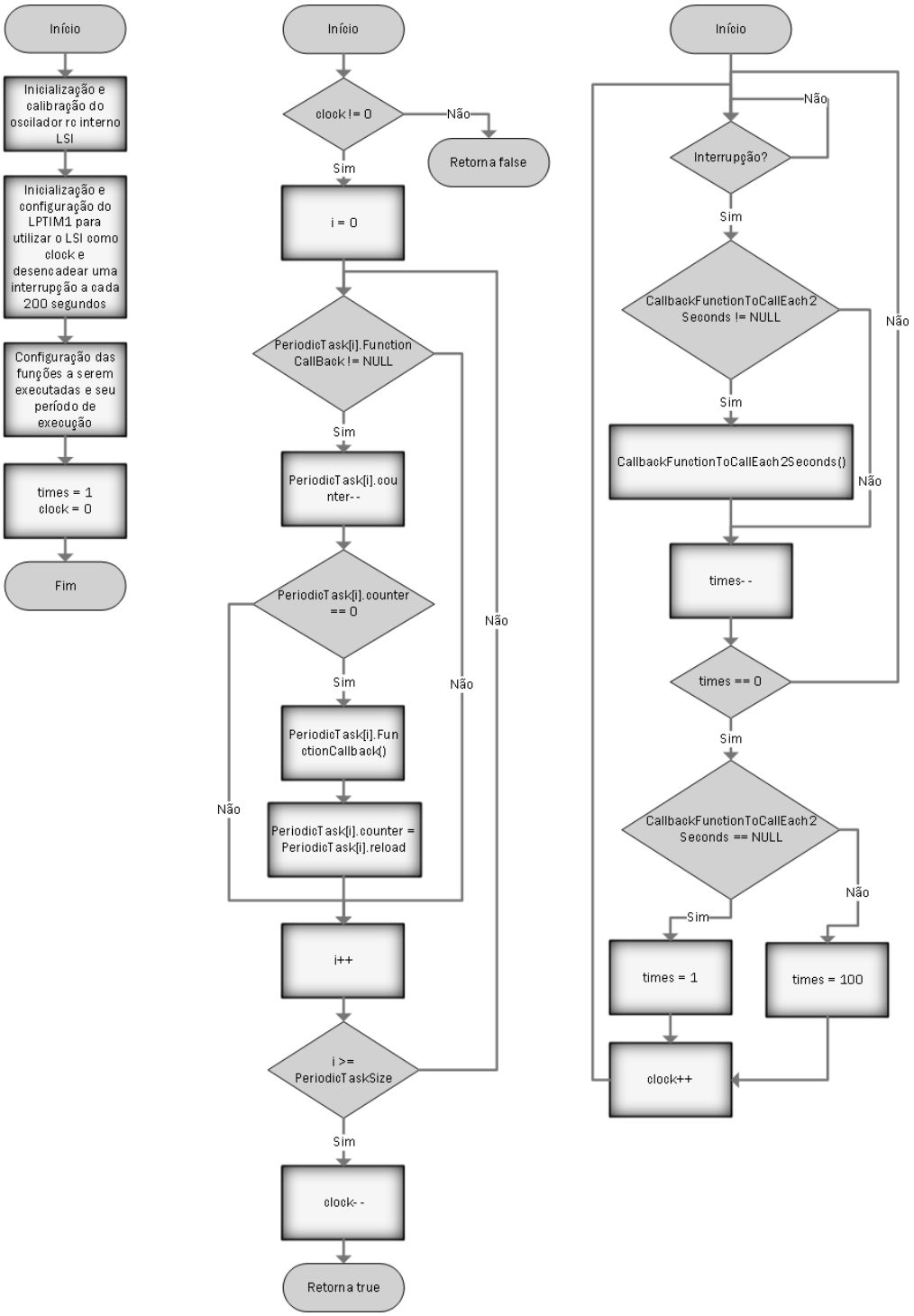


Figura V-4: Fluxograma do módulo *scheduler*.

Note-se que quando ocorre uma interrupção é decrementada uma unidade na variável *counter,* que existe dentro da estrutura representada na Figura V-5. Esta é decrementada em todos os agendamentos periódicos ativos.

```
typedef struct{
  /** @brief Callback function to execute. */
  void (*FunctionCallback)(void);

  /** @brief Time to execute callback function periodically (Each unit represent 200 seconds).  */
  unsigned char reload;

  /** @brief Time remaining to execute callback function again (Each unit represent 200 seconds). */
  unsigned char counter;
}SchedulerAgenda;
```

Figura V-5: Estrutura utilizada para o agendamento periódico da execução de uma função.

Na Figura V-6 pode-se observar o módulo *Scheduler* numa representação sob a forma de classe em *UML*.

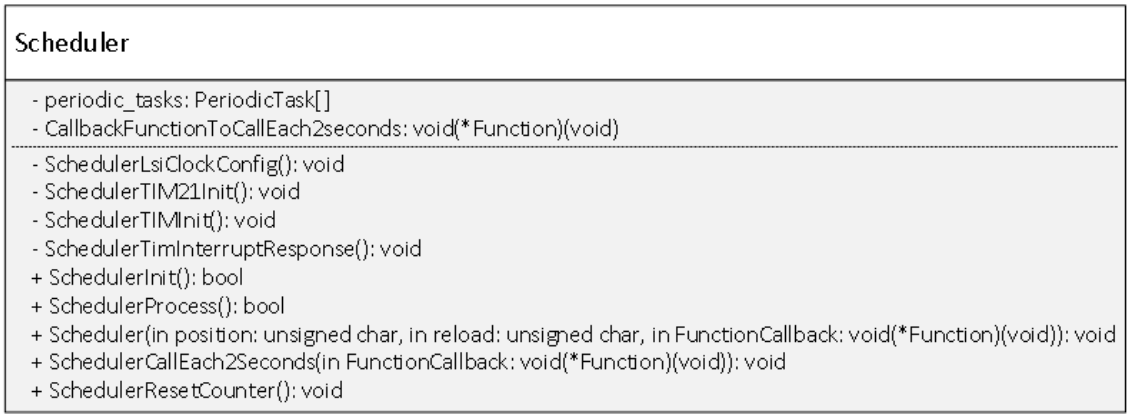


Figura V-6: *UML Scheduler class.*

A classe *Scheduler* contem quatro métodos privados e cinco métodos públicos.

Os métodos privados são:

- *SchedulerLsiClockConfig*: este método é responsável pela ativação e configuração do oscilador RC *LSI* de trinta e sete kilohertz;
- *SchedulerTIM21Init*: configura o *TIM21* para ser usado como calibrador do oscilador *LSI*;
- *SchedulerTIMInit*: ativa e configura o LPTIM para desencadear uma interrupção a cada duzentos segundos;
- *SchedulerTimInterruptResponse*: este método é chamado na resposta à interrupção por *overflow* do timer *LPTIM*.

Os métodos públicos são:

- *SchedulerInit*: este método chama os métodos privados S*chedulerLsiClockConfig, SchedulerTIM21Init* e *SchedulerTIMInit* para configurarem os periféricos do microcontrolador necessários para o funcionamento do módulo;

- *SchedulerProcess:* executa todas as funções agendadas no tempo certo;
- *Scheduler:* define a função a ser executada periodicamente através da passagem do apontador para a função e o tempo do período de execução;
- *SchedulerCallEach2Seconds:* define a função a ser executada a cada dois segundos através da passagem do apontador para a função como parâmetro;
- *SchedulerResetCounter:* este método reinicia o contador *LPTIM.*

5.2.3 Módulo *ADC*

O microcontrolador *STM32L053R8* contém um único periférico *ADC* com desaseis canais de doze *bits* de resolução máxima. Neste projeto estão a ser utilizados dois canais *ADC*, o primeiro para a tensão proporcional à corrente consumida pelo motor da persiana e o segundo para a tensão da bateria. O *ADC* foi configurado para adquirir amostras de doze *bits* de resolução.

Na Figura V-7 podemos observar o módulo *ADC* numa representação sob a forma de classe em *UML*.

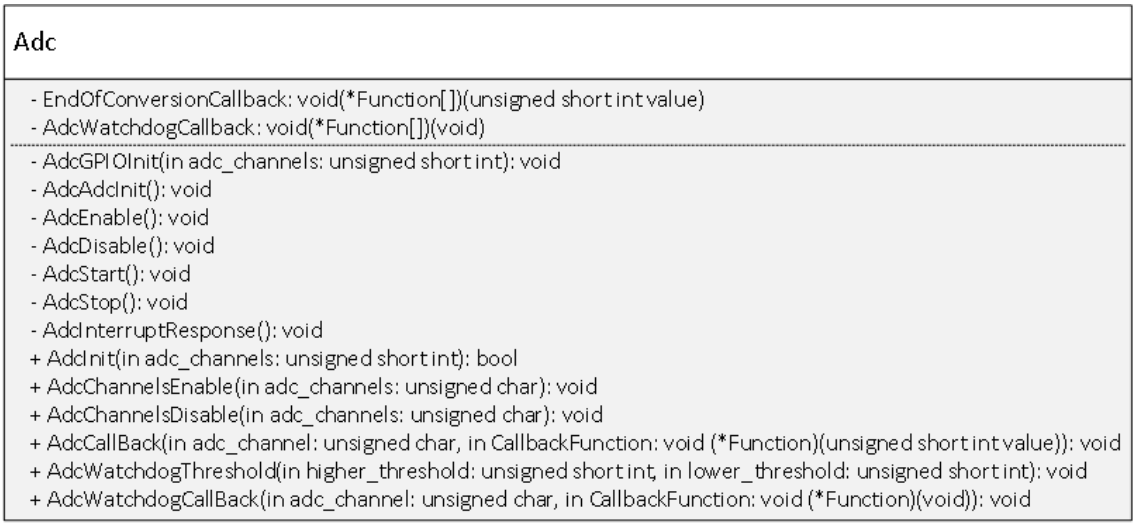


Figura V-7: *UML Adc class.*

A classe *ADC* contêm sete métodos privados e seis métodos públicos.

Os métodos privados são:

- *AdcGPIOInit*: configura os pinos do *GPIO* dos respetivos portos para serem usados como entradas de canais *ADC*;

- *AdcAdcInit:* inicializa e configura o *ADC* para adquirir uma amostra;
- *AdcEnable:* habilita o *ADC*;
- *AdcDisable:* desabilita o *ADC*;
- AdcStart: inicializa a aquisição de amostras;
- *AdcStop:* termina a aquisição de amostras;
- *AdcInterruptResponse:* resposta à interrupção relativa à aquisição de uma amostra.

Os métodos públicos são:

- *AdcInit:* configura o *ADC* e os respetivos pinos utilizados como entradas analógicas;
- *AdcChannelsEnable:* ativar os canais utilizados*;*
- *AdcChannelsDisable:* desabilita os canais*;*
- *AdcCallBack:* esta recebe como parâmetro o apontador da função e executa a função correspondente ao canal de aquisição da amostra;
- *AdcWatchdogThreshold:* configura os níveis de *threshold* do *adc watchdog*;
- *AdcWatchdogCallBack:* esta recebe como parâmetro o apontador da função e executa a função correspondente quando as leituras de um dos canais do *ADC* ultrapassarem os limites do *threshold*.

5.2.4 Módulo *Power*

O módulo *Power* através de dois pinos do *GPIO* habilita e desabilita as tensões que alimentam os circuitos lógicos e os circuitos de potência.

Na Figura V-8 pode-se observar o módulo *Power* numa representação sob a forma de classe em *UML*.

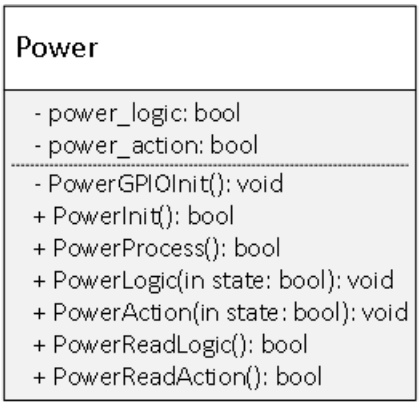


Figura V-8*: UML Power class.*

A classe *Power* contém um método privado e seis métodos públicos.

O método privado é:

- *PowerGPIOInit:* configura os pinos do *GPIO*.

Os métodos públicos são:

- *PowerInit*: invoca a função *PowerGPIOInit*;
- *PowerProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *PowerLogic:* habilita e desabilita os circuitos lógicos;
- *PowerAction:* habilita e desabilita os circuitos de potência;
- *PowerReadLogic:* verifica se os circuitos lógicos estão ligados ou desligados;
- *PowerReadAction:* verifica se os circuitos de potência estão ligados ou desligados.

5.2.5 Módulo *Eeprom*

Este módulo é uma interface simples de usar que facilita a escrita e a leitura na memória interna *eeprom* de dois *kilobytes* no microcontrolador. Assim é simples armazenar e adquirir valores de oito, dezasseis ou trinta e dois *bits* em qualquer endereço da memória *eeprom*.

Na Figura V-9 pode-se observar o módulo *Eeprom* numa representação sob a forma de classe em *UML*.

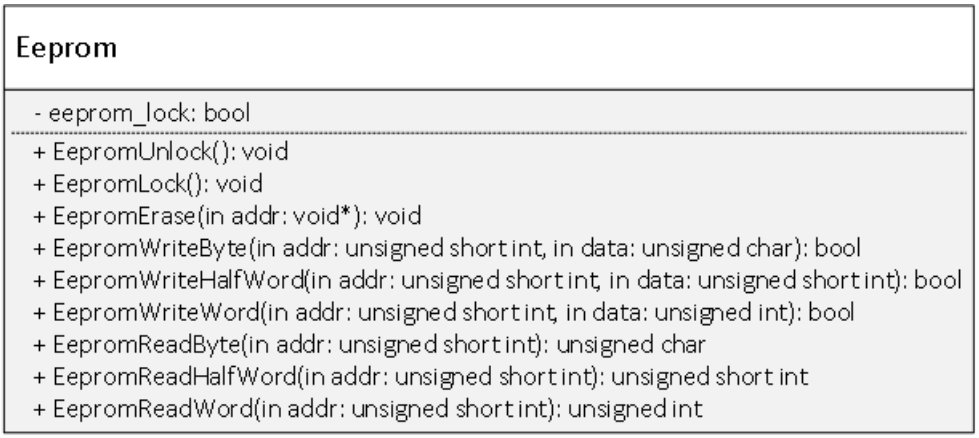


Figura V-9: *UML Eeprom class.*

A classe *Eeprom* contém nove métodos públicos.

Os métodos públicos são:

- *EepromUnlock:* desbloqueia a memória *eeprom* para escrita;
- *EepromLock:* bloqueia a memória *eeprom* para impedir a escrita;
- *EepromErase:* apaga a memória*;*
- *EepromWriteByte:* escreve um *byte* num determinado endereço*;*
- *EepromWriteHalfWord:* escreve dois *bytes* num determinado endereço;
- *EepromWriteWord:* escreve quatro *bytes* num determinado endereço;
- *EepromReadByte:* lê um *byte* num determinado endereço;
- *EepromReadHalfWord:* lê dois *bytes* num determinado endereço;
- *EepromReadWord:* lê quatro *bytes* num determinado endereço.

5.2.6 Módulo *Blind*

O módulo *Blind* é responsável pela movimentação da persiana para a posição pretendida e pelo ajuste da posição das abas.

Na Figura V-10 apresenta-se o fluxograma deste módulo.

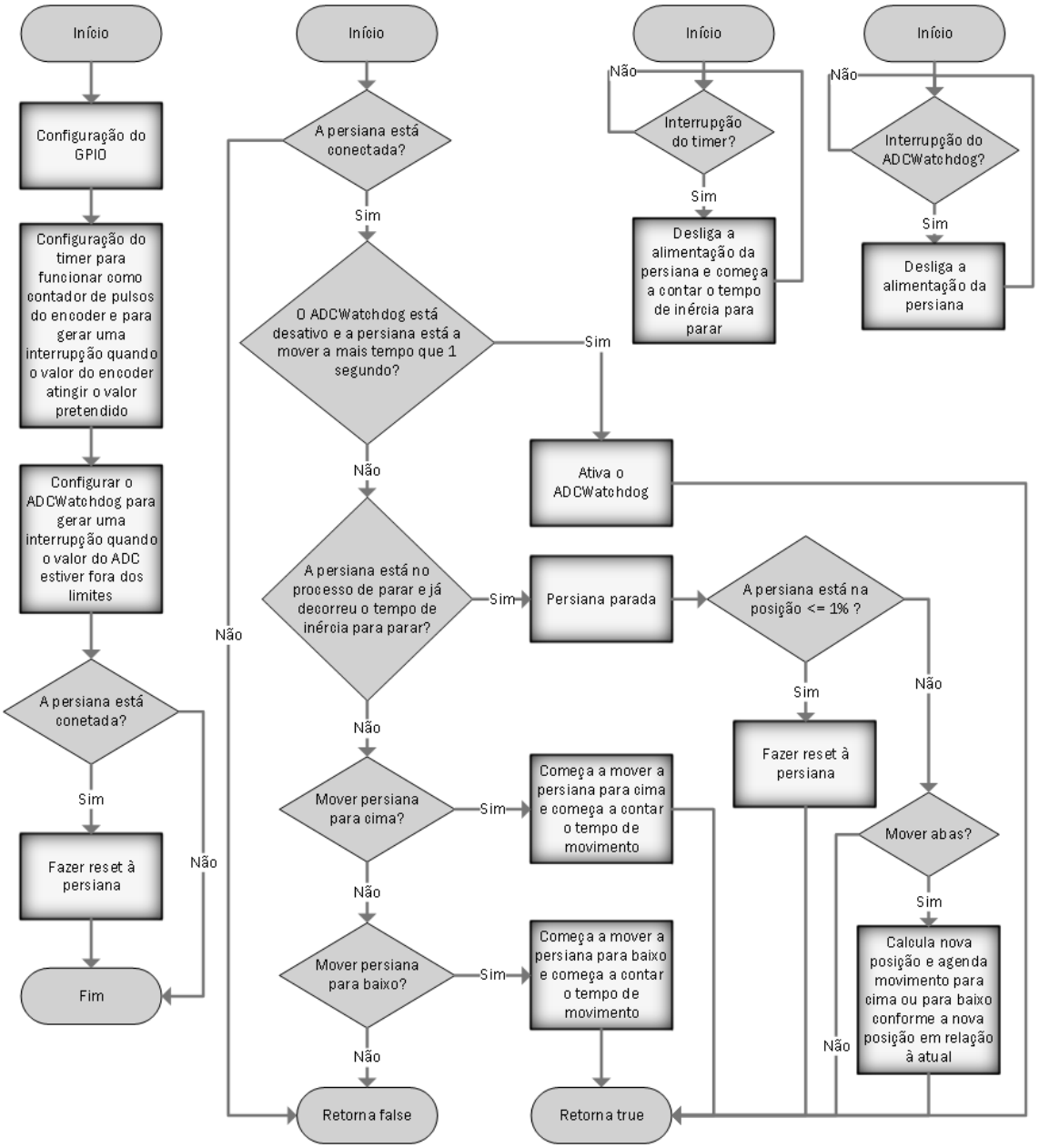


Figura V-10: Fluxograma do módulo Blind.

O fluxograma mais à esquerda corresponde à inicialização e configuração dos periféricos e variáveis necessárias para o funcionamento do módulo.

O fluxograma do centro representa a execução em ciclo infinito dentro da função *main* do *software*.

Os dois fluxogramas mais pequenos à esquerda são as respostas às interrupções do *timer* e do *ADCWatchdog*.

O controlo da posição da persiana é efetuado através do *encoder* embutido na persiana. Para contar os pulsos configurou-se o *timer TIM22*. Este funciona como contador de transições no

pino conectado ao *encoder*. Quando o valor no registo do contador coincidir com o valor armazenado no registo de comparação é desencadeada uma interrupção através dos comparadores do *timer*. Esta ocorre quando é detetado que a persiana chegou à posição pretendida.

Sempre que a persiana necessita de ser alterada para uma nova posição é calculado o novo valor que o contador do *timer* deve atingir e copiado para o registo de comparação do *timer*. De seguida, movimenta-se a persiana na direção desejada e espera-se que ocorra a interrupção do comparador do *timer*. Quando a interrupção do comparador do *timer* correr, o movimento da persiana é terminado uma vez que esta já chegou à posição desejada. Depois da persina estar na posição correta é ajustada a inclinação das abas. Este é efetuado através do calculo do novo valor da posição a ser copiado para o registo de comparação do timer e repetindo o mesmo método descrito em cima.

Na Figura V-11 pode-se observar o módulo *Blind* numa representação sob a forma de classe em *UML*.

Figura V-11: *UML Blind class.*

A classe *Blind* contém dez métodos privados e quinze métodos públicos.

Os métodos privados são:

- *BlindGPIOInit:* inicializa e configura os pinos necessários para controlar a ponte que alimenta o motor da persiana e o pino utilizado para conectar o sinal do *encoder* ao *timer TIM22*;
- *BlindTimInit:* inicializa e configura o *timer TIM22* para funcionar como contador e comparador;
- *BlindNoEncoderInit:* esta função configura o *timer TIM22* para ser usado como temporizador caso a persina não disponha de um *encoder*. Desta forma, a posição da persina é controlada através do tempo de movimento;

- *BlindMoveUp:* movimenta a persiana de forma ascendente;
- *BlindMoveDown:* movimenta a persina de forma descendente;
- *BlindMoveStop:* termina o movimento da persina;
- *BlindMoveFlap:* ajusta a inclinação das abas após o movimento da posição da persiana;
- *BlindReadPosition:* devolve a posição da persiana em percentagem;
- *BlindAdcWatchdogCallback:* esta função é a resposta à interrupção do *ADCWatchdog* que ocorre quando o motor da persiana consome mais corrente que o normal. Quando o consumo é fora do normal esta função cessa de imediato a alimentação do motor;
- *BlindEncoderInterruptResponse:* é a resposta à interrupção do *timer TIM22* que ocorre quando o valor do registo de comparação coincide com o valor do registo do contador.

Os métodos públicos são:

- *BlindInit:* inicializa o módulo através da chamada das funções privadas *BlindGPIOInit* e *BlindTimInit* e inicializa as variáveis necessárias ao funcionamento do módulo;
- *BlindProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *BlindReset:* movimenta a persiana toda para cima e inicializa o contador do *timer* a zero;
- *BlindTest:* executa uma rotina de testes pré-programada que consiste em movimentar a persiana e as abas num padrão conhecido para uma inspeção visual;
- *BlindSetSize:* configura o comprimento máximo da persiana;
- *BlindManualSetSize:* configura o comprimento máximo da persiana através do teclado;
- *BlindMove:* movimenta a persiana para uma determinada posição;
- *BlindStop:* cessa o movimento da persiana;
- *BlindFlapMove:* ajusta o ângulo de inclinação das abas;
- *BlindGetSize:* devolve o tamanho da persiana;
- *BlindGetPosition:* devolve a posição atual da persiana;
- *BlindGetFlapsPosition:* devolve a inclinação atual das abas;

- *BlindIsMoving:* verifica se a persiana está em movimento;
- *BlindConnected:* verifica se a persiana esta conectada;
- *BlindEncoderEnabled:* verifica se a persiana dispõe do *encoder*.

5.2.7 Módulo *Valve*

O módulo *Valve* controla a posição da válvula interior e da exterior, de acordo com o cenário selecionado. No total existem quatro cenários distintos, *All Opened*, *All Closed*, *Cooling* e *Preheat*. O cenário *Preheat* está dividido em *Preheat33*, *Preheat66* e *Preheat100*. O cenário *All Opened* abre as duas válvulas, o *All Closed* fecha as duas válvulas e o *Cooling* fecha a válvula interior e abre a exterior. O *Preheat* fecha as duas válvulas a cada período de trinta minutos e de acordo com o cenário ativo abre a válvula interior para depois a fechar novamente no inicio do próximo período. O *Preheat33* abre a válvula interior aos dez minutos, o *Preheat66* abre a válvula interior aos vinte minutos e o *Preheat100* está sempre com a válvula interior aberta.

Existe ainda dois modos de execução, o automático e o manual. No modo automático são alterados os cenários aplicados às válvulas de acordo com os comandos recebidos via rádio provenientes do algoritmo de controlo automático. No modo manual os comandos via rádio do algoritmo de controlo automático são ignorados aceitando apenas os comandos manuais do utilizador.

Na Figura V-12 pode-se observar o módulo *Valve* numa representação sob a forma de classe em *UML*.

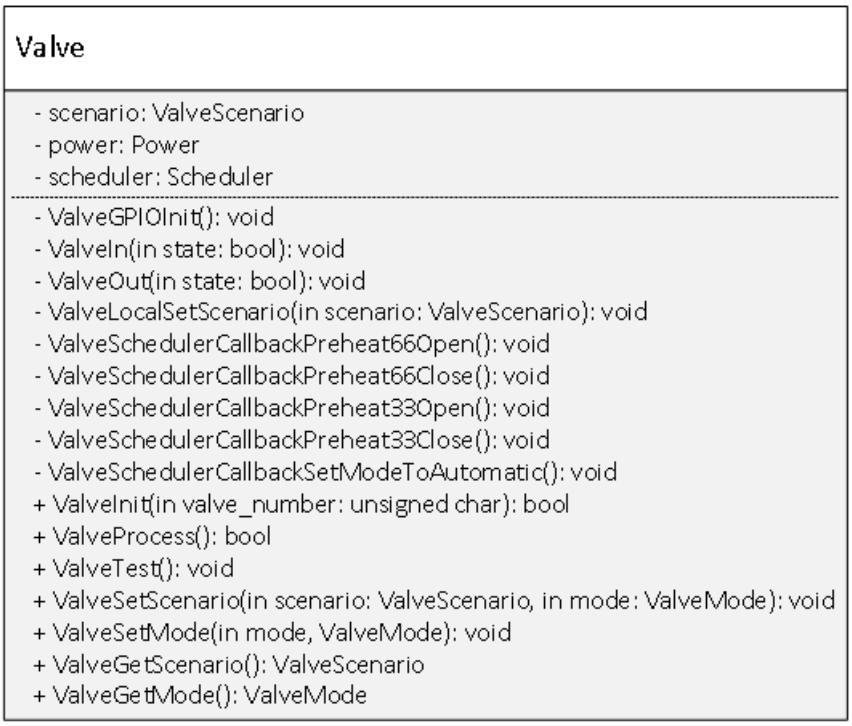


Figura V-12: *UML Valve class.*

A classe *Valve* contém nove métodos privados e sete métodos públicos.

Os métodos privados são:

- *ValveGPIOInit:* inicializa e configura os pinos necessários para controlar as pontes que alimentam as válvulas;
- *ValveIn:* abre ou fecha a válvula interior;
- *ValveOut:* abre ou fecha a válvula exterior;
- *ValveLocalSetScenario:* aplica o cenário selecionada às válvulas;
- *ValveSchedulerCallbackPreheat66Open:* abre a válvula interior após ter decorrido vinte minutos;
- *ValveSchedulerCallbackPreheat66Close:* fecha a válvula interior após ter decorrido dez minutos;
- *ValveSchedulerCallbackPreheat33Open:* abre a válvula interior após ter decorrido dez minutos;
- *ValveSchedulerCallbackPreheat33Close:* fecha a válvula interior após ter decorrido vinte minutos;
- *ValveSchedulerCallbackSetModeToAutomatic:* muda para o modo automático após ter decorrido uma hora no modo manual.

Os métodos públicos são:

- *ValveInit:* inicializa as variáveis internas do módulo e chama da função privada *ValveGPIOInit*;
- *ValveProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *ValveTest:* executa uma rotina pré-programada que movimenta as válvulas num padrão conhecido para uma inspeção visual do seu bom funcionamento;
- *ValveSetScenario:* altera o cenário atual das válvulas para o especificado no parâmetro de entrada da função;
- *ValveSetMode:* altera o modo de execução para o modo definido no parâmetro de entrada da função;
- *ValveGetScenario:* devolve o cenário atual aplicado nas válvulas;
- *ValveGetMode:* devolve o modo de execução atual.

5.2.8 Módulo *Sensors*

O módulo *Sensors* está encarregue por todas as configurações e aquisições de dados dos sensores utilizados. Neste módulo utiliza-se o periférico *I2C* para comunicar com os sensores *STH21* e o *MAX44009*, sendo *ARM* o *master* e os dois sensores os *slaves*.

O *MAX44009* e o *SHT21* são os sensores descritos no capitulo anterior, sendo o primeiro responsável pela aquisição da luminosidade e o segundo pela temperatura e humidade.

Na Figura V-13 visualiza-se o módulo *Sensors* numa representação sob a forma de classe em *UML*.

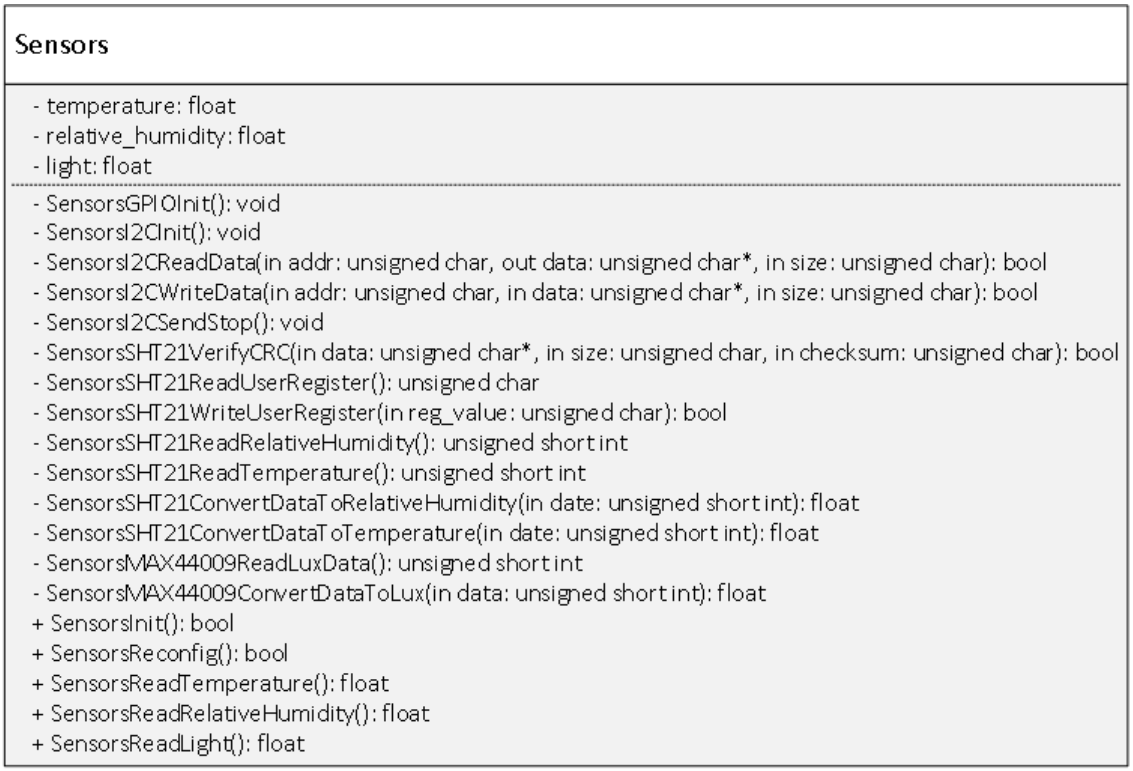


Figura V-13: *UML Sensors class.*

A classe *Sensors* é constituída por catorze métodos privados e cinco métodos públicos. Os métodos privados são apresentados de seguida.

- *SensorsGPIOInit:* inicializa e configura os pinos *SCL* e *SDA* do barramento *I2C* ao *GPIO* do *ARM*;
- *SensorsI2CInit:* inicializa e configura o periférico *I2C*;
- *SensorsI2CReadData:* lê uma determinada quantidade de *bytes* no canal *I2C* de um *slave*;
- *SensorsI2CWriteData:* escreve uma determinada quantidade de *bytes* no canal *I2C* para um *slave*;
- *SensorsI2CSendStop:* envia o sinal de stop no canal *I2C*;
- *SensorsSHT21VerifyCRC:* verifica se os dados foram recebidos sem erros;
- *SensorsSHT21ReadUserRegister:* lê o conteúdo do registo *user* do sensor *STH21*;
- *SensorsSHT21WriteUserRegister:* escreve no registo *user* do sensor *STH21*;
- *SensorsSHT21ReadRelativeHumidity:* lê dois *bytes* de dados com o valor da humidade relativa;
- *SensorsSHT21ReadTemperature:* lê dois *bytes* de dados com o valor da temperatura;

- *SensorsSHT21ConvertDataToRelativeHumidity:* converte os dois *bytes* de dados num *float* com o valor da humidade relativa em percentagem;
- *SensorsSHT21ConvertDataToTemperature:* converte os dois *bytes* de dados num *float* com o valor da temperatura em graus centigrados;
- *SensorsMAX44009ReadLuxData:* lê dois *bytes* de dados com o valor da luminosidade;
- *SensorsMAX44009ConvertDataToLux:* converte os dois *bytes* de dados num *float* com o valor da luminosidade em *LUX's*.

Os métodos públicos são:

- *SensorsInit:* chama as funções privadas *SensorsGPIOInit e SensorsI2CInit* para configurar e ativar o periférico *I2C*;
- *SensorsReconfig:* reconfigura os registos internos dos sensores quando o *ARM* acorda porque a alimentação dos sensores é desligada quando o *ARM* entra no modo de "dormir";
- *SensorsReadTemperature:* retorna o valor atual da temperatura em graus centigrados;
- *SensorsReadRelativeHumidity:* retorna o valor da humidade relativa em percentagem;
- *SensorsReadLight:* retorna o valor da luminosidade em *LUX*.

5.2.9 Módulo *Battery*

O módulo *Battery* como o próprio nome indica está encarregue pela monitorização do estado da bateria do sistema. Este através de *LEDs* informa o utilizador sobre o estado atual da bateria, mais concretamente indica quando é que bateria está com a carga completa, se está a ser carregada e por último quando atinge vinte porcento de carga.

Na Figura V-14 pode-se observar o módulo *Battery* numa representação sob a forma de classe em *UML*.

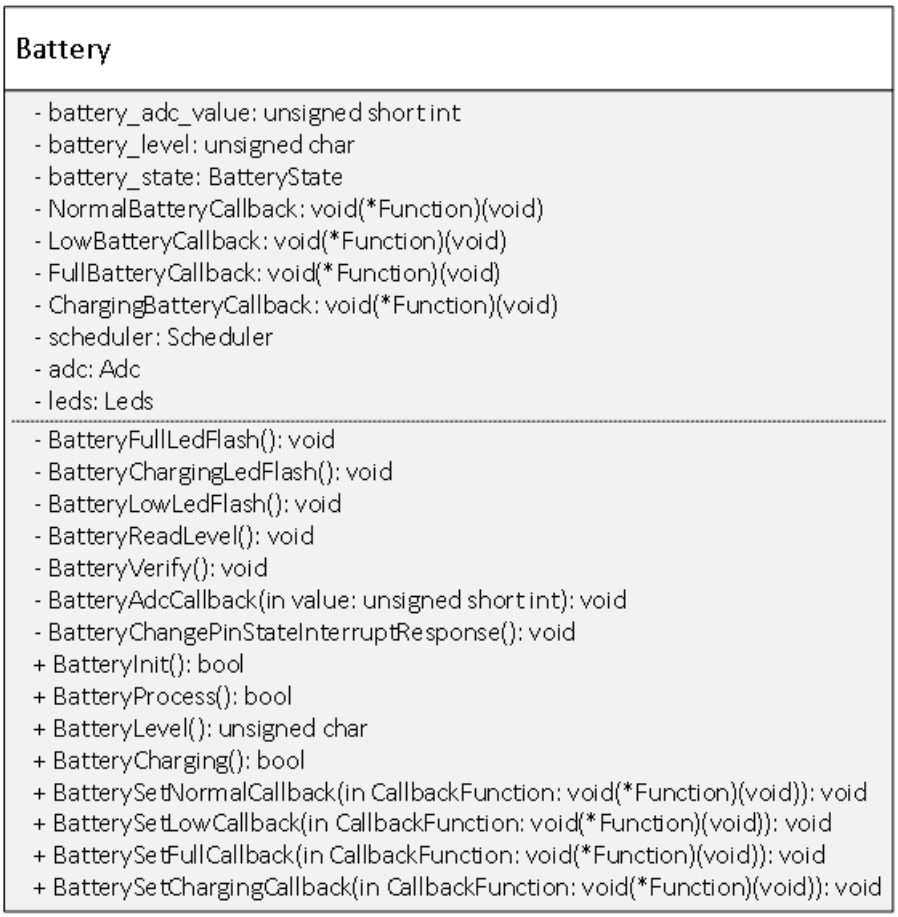


Figura V-14: *UML Battery class.*

A classe *Battery* contêm sete métodos privados e oito métodos públicos, sendo estes apresentados de seguida.

Os métodos privados são:

- *BatteryFullLedFlash:* faz piscar o *LED* verde a cada dois segundos para indicar que a bateria está com a carga no máximo;
- *BatteryChargingLedFlash:* faz piscar os *LEDs* vermelho e verde a cada dois segundos para indicar que a bateria está a ser carregada;
- *BatteryLowLedFlash:* faz piscar o *LED* vermelho a cada dois segundos para indicar que o nível da bateria está abaixo de vinte porcento;
- *BatteryReadLevel:* manda o *ADC* executar uma leitura do nível da bateria;
- *BatteryVerify:* atualiza o estado da bateria;
- *BatteryAdcCallback:* resposta à interrupção do *ADC*;
- *BatteryChangePinStateInterruptResponse:* resposta à mudança de estado do pino que indica se o *USB* está ou não conectado.

Os métodos públicos são:

- *BatteryInit:* inicializa as variáveis internas do módulo;
- *BatteryProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *BatteryLevel:* retorna o nível da bateria em percentagem;
- *BatteryCharging:* indica se a bateria está a ser carregada;
- *BatterySetNormalCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente quando o utilizador desconectar o carregador e o nível da bateria estiver acima de 20%;
- *BatterySetLowCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente quando a bateria atingir vinte porcento;
- *BatterySetFullCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente quando a bateria atingir cem porcento;
- *BatterySetChargingCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente quando for conectado o carregador *USB*.

5.2.10 Módulo *Keyboard*

O módulo *Keyboard* é responsável por identificar a tecla ou as teclas premidas e o tempo de duração do pressionamento. O reconhecimento das teclas pressionadas é feito através de um divisor resistivo, sendo o valor da tensão resultante do divisor resistivo lido através do *ADC* do *ARM*.

O *ARM* é acordado por *hardware* sempre que um botão é pressionado. Este depois de acordado faz uma leitura do valor analógico através do ADC. Com o valor analógico adquirido identifica a tecla ou as teclas premidas e conta o tempo de duração do pressionamento.

Na Figura V-15 pode-se observar o módulo *Keyboard* numa representação sob a forma de classe em *UML*.

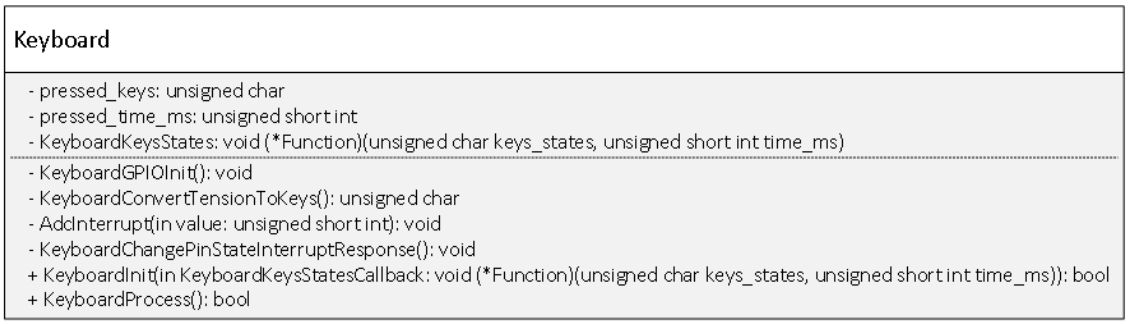


Figura V-15: *UML Keyboard class.*

A classe *Keyboard* contém quatro métodos privados e dois métodos públicos.

Os métodos privados são:

- *KeyboardGPIOInit:* configura dois pinos do *GPIO*, um para ser a entrada analógica de um dos canais do *ADC* e outro para desencadear uma interrupção por mudança de estado do pino;
- *KeyboardConvertTensionToKeys:* identifica a tecla ou as teclas premidas através do valor adquirido pelo *ADC*;
- *AdcInterrupt:* resposta à interrupção desencadeada pelo *ADC* quando a aquisição e conversão do valor analógico terminou;
- *KeyboardChangePinStateInterruptResponse:* resposta interrupção da mudança do estado do pino que indica que uma tecla do teclado foi pressionada ou largada;

Os métodos públicos são:

- *KeyboardInit:* inicialização do *GPIO* e configuração de um dos canais do *ADC*;
- *KeyboardProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*

5.2.11 Módulo *Leds*

O módulo *Leds* proporciona o controlo dos três *LEDs* (vermelho, verde e azul) que estão ligados ao microcontrolador através de três pinos do *GPIO*. Este módulo facilita o controlo dos *LEDs* através de uma interface simples que possibilita fazer piscar o *LEDs*, individualmente ou em conjunto, num determinado número de vezes por cada dois segundos. Possibilita também piscar os *LEDs* ciclicamente ou uma única vez.

Na Figura V-16 pode-se observar o módulo *Leds* numa representação sob a forma de classe em *UML*.

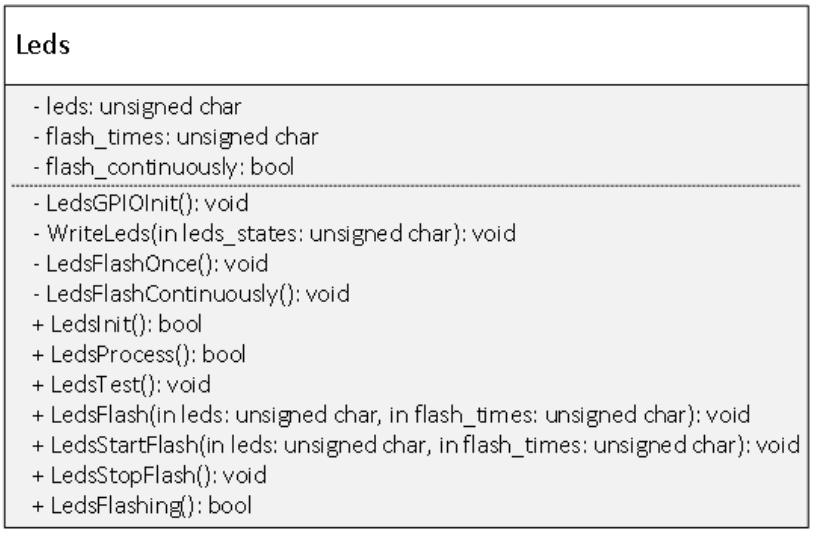


Figura V-16: *UML Leds class.*

A classe *Leds* contêm quatro métodos privados e sete métodos públicos.

Os métodos privados são:

- *LedsGPIOInit*: inicializa e configura os tês pinos conectados aos *LEDs*;
- *WriteLeds*: aplica o novo estado aos LEDs;
- *LedsFlashOnce:* faz o *LED* ou *LEDs* piscarem um determinado número de vezes, apenas uma única vez;
- *LedsFlashContinuously:* faz o *LED* ou *LEDs* piscarem um determinado número de vezes a cada dois segundos de forma ciclica.

Os métodos públicos são:

- *LedsInit:* inicializa e configura o GPIO através da chamada do método privado *LedsGPIOInit.* E inicializa as variáveis internas do módulo;
- *LedsProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *LedsTest:* executa uma rotina de teste para que de forma visual se possa comprovar o bom funcionamento dos *LEDs*;
- *LedsFlash:* pisca o *LED* ou *LEDs* um determinado número de vezes;
- *LedsStartFlash:* começa a piscar o *LED* ou *LEDs* um determinado número de vezes a cada dois segundos ciclicamente;

- *LedsStopFlash:* para o *LED* ou *LEDs* de piscar;
- *LedsFlashing:* indica se os *LEDs* estão a piscar.

5.2.12 Módulo *KeyboardMenus*

O módulo *KeyboardMenus* implementa a interface de utilizador com o sistema desenvolvido. O esquema de menus está presente no apêndice I.

Na Figura V-17 é possível observar o módulo *KeyboardMenus* numa representação sob a forma de classe em *UML*.

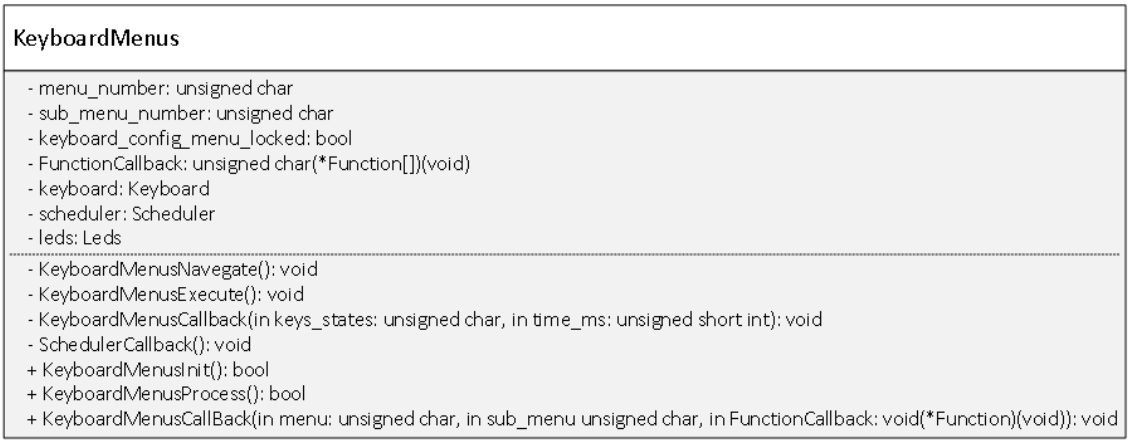


Figura V-17: *UML KeyboardMenus class.*

A classe *KeyboardMenus* contêm quatro métodos privados e três métodos públicos.

Os métodos privados são:

- *KeyboardMenusNavegate:* atualiza a variável através da interação do utilizador com o teclado, tendo como feedback o *LED* que indica o menu atual;
- *KeyboardMenusExecute:* executa a função da tecla pressionado no contexto do menu;
- *SchedulerCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente após o bloqueio dos menus de configuração.

Os métodos públicos são:

- *KeyboardMenusInit:* inicializa as variáveis internas do módulo;

- *KeyboardMenusProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *KeyboardMenusCallBack:* esta recebe como parâmetro o apontador da função e executa a função correspondente de acordo com o menu e tecla pressionada.

5.2.13 Módulo *EnOcean*

O módulo *EnOcean* estabelece a comunicação com o microcontrolador *EnOcean*. De uma forma geral, este módulo está encarregue por enviar toda a informação que o microcontrolador *EnOcean* solicita, por exemplo envio da temperatura, da posição persiana, do cenário das válvulas entre outros. Note-se que o microcontrolador *EnOcean* também pode mandar o *ARM* atuar a persiana, as válvulas, entre outras.

Na Figura V-18 pode-se observar o módulo *EnOcean* numa representação sob a forma de classe em *UML*.

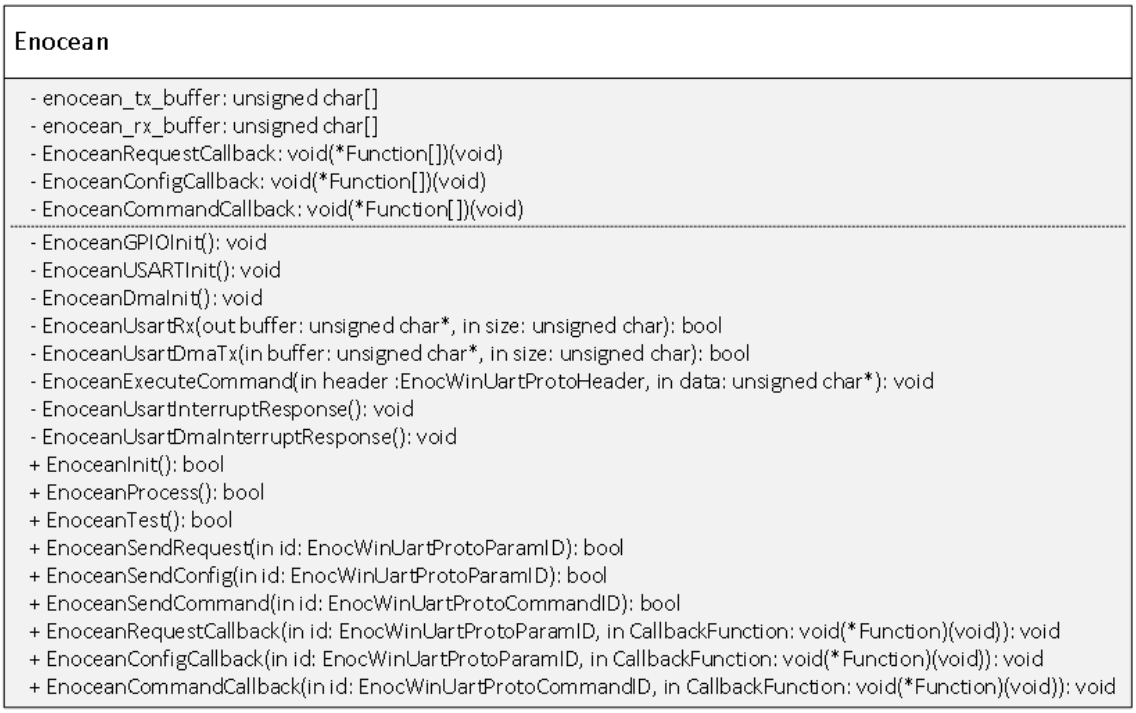


Figura V-18: *UML EnOcean class.*

A classe *EnOcean* contêm oito métodos privados e nove métodos públicos.

Os métodos privados são:

- *EnOceanGPIOInit:* configura dois pinos do *GPIO* para se conectar ao periférico *USART*, configura um terceiro para desencadear uma interrupção por mudança de estado do pino e outro para "acordar" o *EnOcean*;
- *EnOceanUSARTInit:* configura e inicializa o periférico *USART*;
- *EnOceanDmaInit:* configura e inicializa o periférico DMA que é utilizado em conjunto com a *USART*;
- *EnOceanUsartRx:* preenche um *buffer* com os *bytes* recebidos via *USART*;
- *EnOceanUsartDmaTx:* copia a serem enviados para o buffer de envio;
- *EnOceanExecuteCommand:* executa o comando enviado pelo microcontrolador *EnOcean*;
- *EnOceanUsartInterruptResponse:* resposta à interrupção quando o buffer *RX* tiver dados;
- *EnOceanUsartDmaInterruptResponse:* resposta à interrupção quando o buffer do DMA estiver vazio.

Os métodos públicos são:

- *EnOceanInit:* inicializa as variáveis internas do módulo;
- *EnOceanProcess:* esta função é executada em ciclo infinito na função *main* do módulo principal *WindowController;*
- *EnOceanTest:* executa uma rotina de teste para comprovar o bom funcionamento do *transceiver*;
- *EnOceanSendRequest:* solicita ao *EnOcean* os valores do CO_2;
- *EnOceanSendCommand:* tem como função habilitar e desabilitar o módulo EnOcean, efetuar o teste ao *transceiver*, entre outros.
- *EnOceanRequestCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente dependendo do tipo de parâmetros solicitados;
- *EnOceanConfigCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente, por exemplo atuar a persiana, as válvulas entre outras;
- *EnOceanCommandCallback:* esta recebe como parâmetro o apontador da função e executa a função correspondente, por exemplo a movimentação ascendente e descendente da persiana, da rotação das abas por incremento, entre outras.

5.2.14 Módulo *WindowController*

O módulo *WindowController* corresponde ao módulo *main* do sistema. É neste módulo que as funções por *callback* são invocadas para desempenharem a sua respetiva função.

Note-se que neste módulo apenas existem métodos privados, uma vez que as funções funcionam por *callback*, ou seja, apenas é enviado o apontador da função a executar. Desta forma, garante-se total independência entre módulos e um código genérico.

Na Figura V-19 observa-se o módulo *WindowController* numa representação sob a forma de classe em *UML*. Ao contrário dos capítulos anteriores, neste não são descritas as funções dos métodos apresentados na figura, uma vez que estes já foram abordados nos subcapítulos anteriores.

Figura V-19: *UML WindowController class.*

Para explicar o funcionamento deste módulo optou-se por elaborar o fluxograma apresentado na Figura V-20. Neste são descritas todas as etapas de execução do sistema.

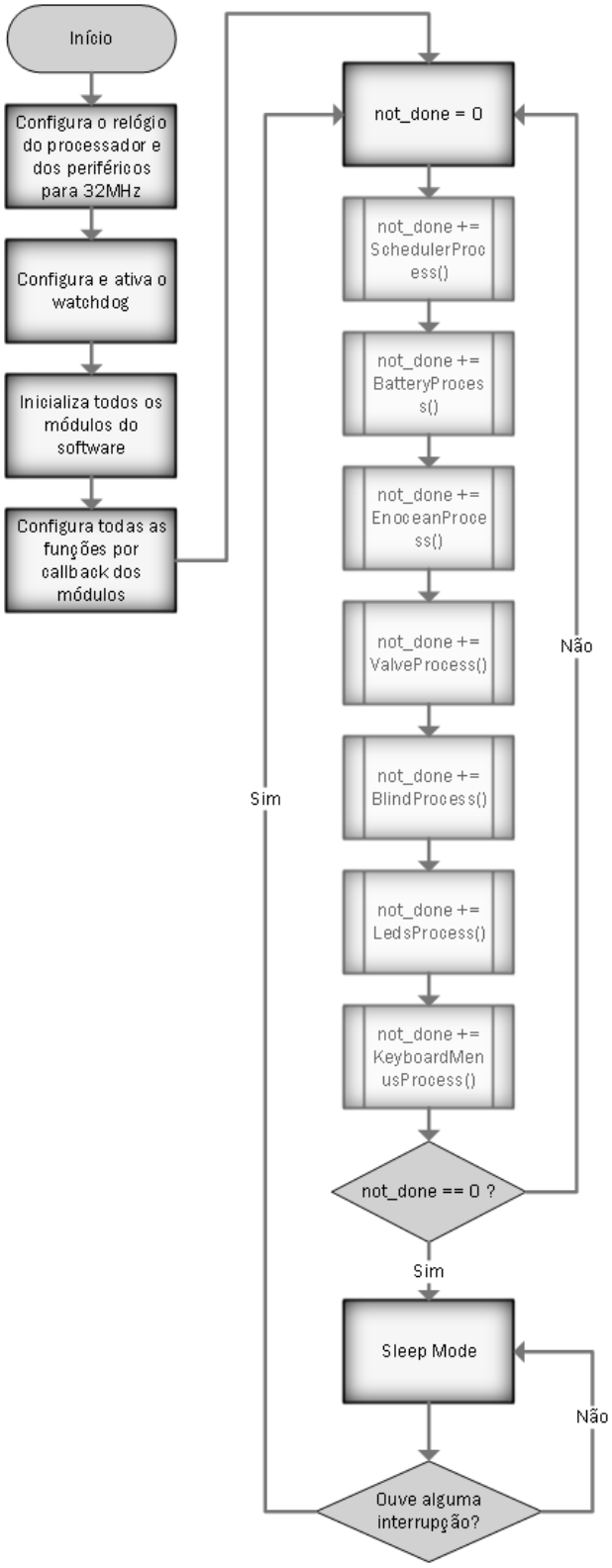


Figura V-20: Fluxograma do módulo *WindowController.*

Capítulo VI - Resultados

No presente capitulo são apresentados os resultados principais do *porting* da aplicação de domótica para o *Arm Cortex M0+.* O capítulo está dividido em dois subcapítulos, resultados individuais e resultados de integração onde são apresentados os consumos do sistema em distintas fases de execução, comparações de consumo com o sistema antigo e tempos de execução nas diversas fases.

6.1. Resultados Individuais

Para os resultados individuais considerou-se importante registar o consumo do movimento ascendente e descente da persiana e para o movimento de abertura e fecho das válvulas. De igual forma registou-se o consumo e o tempo de execução do *EnOcean* e do *ARM.* Quando estes acordam, executam as tarefas e voltam a "dormir" numa das suas rotinas periódicas de dois em dois segundos. Por fim é apresentado o resultado do teste da autonomia da bateria.

6.1.1 Atuação no motor da persiana da janela

Na Figura VI-1 está representada a tensão aplicada ao motor da persiana ao longo de um teste de vinte segundos.

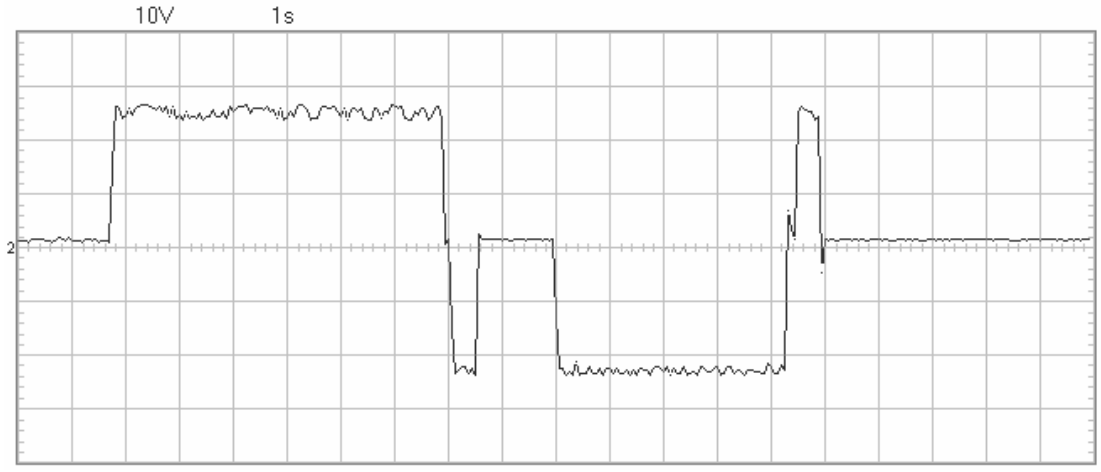


Figura VI-1: Gráfico de tensão à saída da ponte do movimento da persiana.

Na fase inicial a persiana encontra-se em repouso durante aproximadamente um segundo e sete décimas. A partir de um segundo e sete décimas até ao oitavo segundos é aplicada uma

tensão positiva de vinte e quatro *volts* e a persiana movimenta-se ascendentemente. Dos oito segundos até aos oito segundos e meio a persiana desce para ajustar a inclinação das abas sendo aplicada uma tensão negativa de vinte e quatro *volts*. A partir dos oito segundos e meio até aos dez segundos a persiana fica em repouso. Dos dez segundos até aos catorze segundos e três décimas a persiana movimenta-se no sentido descendente. Aos catorze segundos e três décimas a persiana chega à posição pretendida e ajusta a inclinação das abas com um movimento ascendente até aos quinze segundos. A partir dos quinze segundos até ao fim a persiana fica em repouso.

Na Figura VI-2 e na Figura VI-3 estão representadas as formas de onda à saída do *encoder* quando a persiana sobe (Figura VI-2) e quando a persiana desce (Figura VI-3).

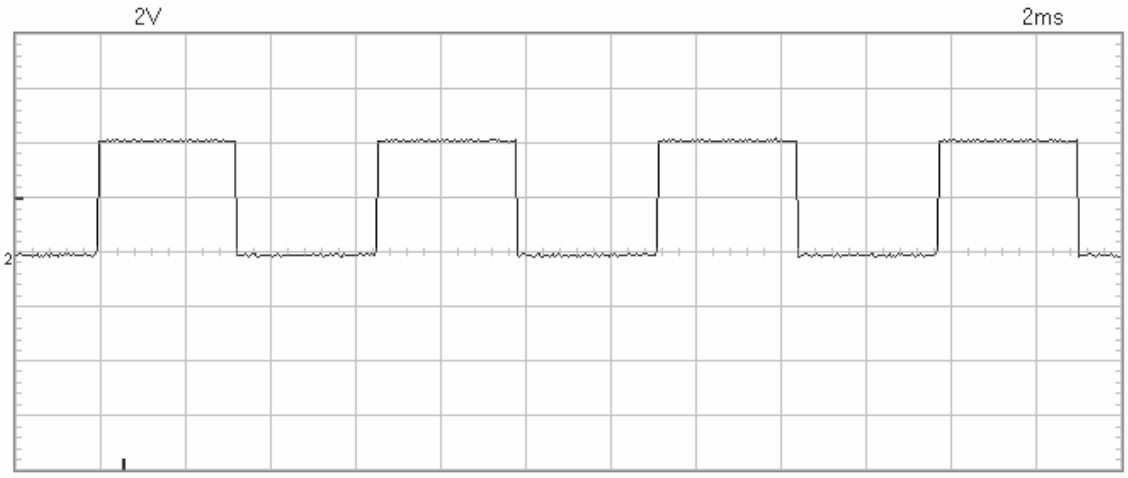


Figura VI-2: Forma de onda do sinal de saída do *encoder* no movimento ascendente.

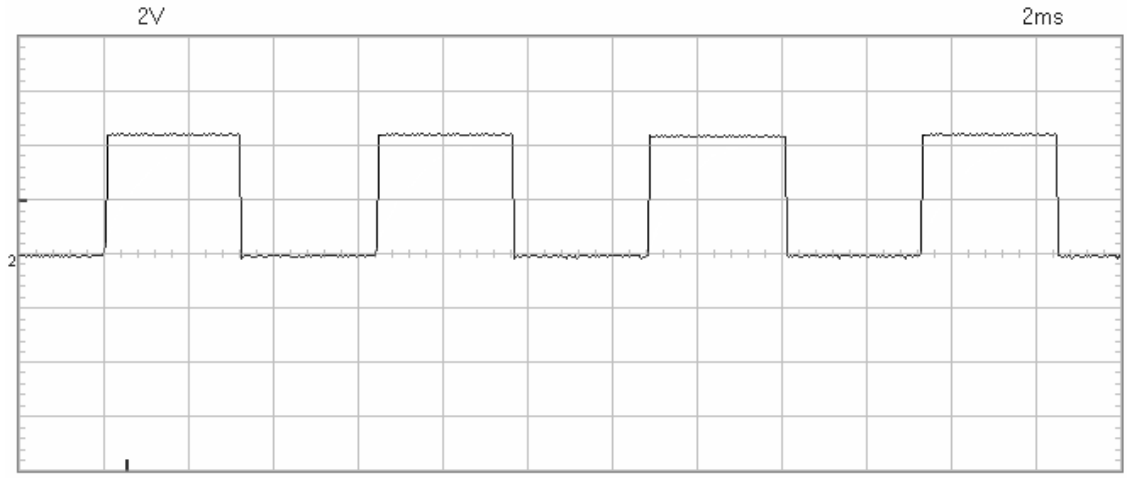


Figura VI-3: Forma de onda do sinal de saída do *encoder* no movimento descendente.

A frequência do sinal do *encoder* é diferente na subida e na descida. Na subida a frequência é de cento e cinquenta e cinco hertz e na descida é de cento e sessenta hertz. Esta diferença deve-se à força gravítica, quando a persiana desce a força gravítica faz o motor rodar mais rápido e quando sobe o motor tem um esforço maior o que provoca uma velocidade inferior.

Na Figura VI-4 e na Figura VI-5 pode-se visualizar a forma de onda do consumo quando a persiana sobe e desce respetivamente.

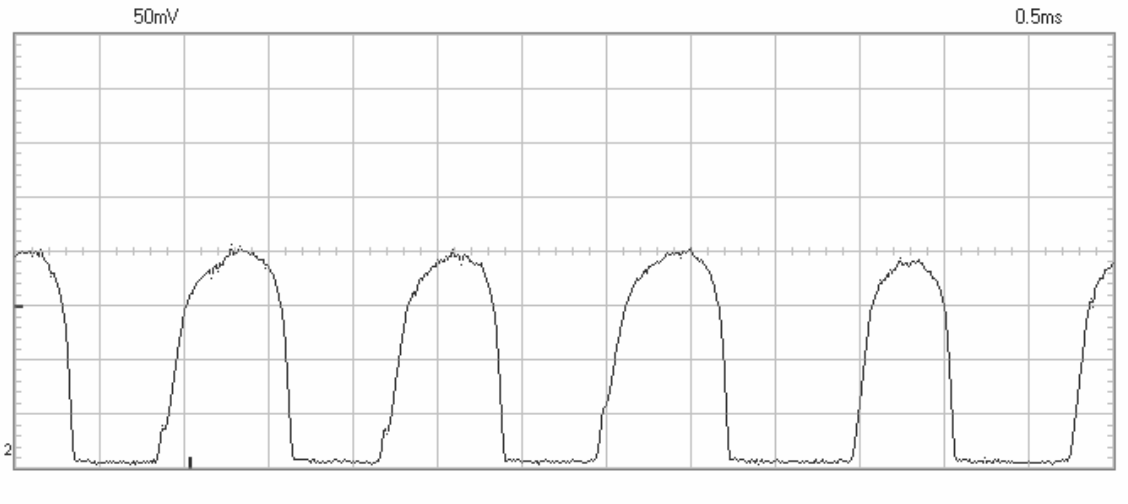


Figura VI-4: Forma de onda, de consumo de corrente com uma resistência em série de 0,05Ω, com a persiana em movimento ascendente.

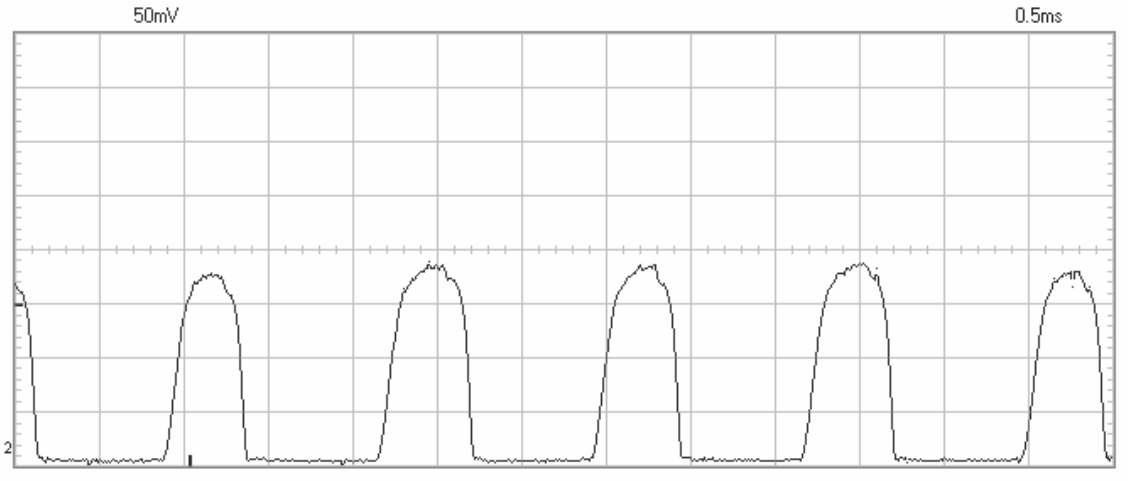


Figura VI-5: Forma de onda, de consumo de corrente com uma resistência em série de 0,05Ω, com a persiana em movimento descendente.

Dos gráficos apresentados infere-se que a unidade de medida está em *volts*, uma vez que o osciloscópio só mede tensões. Para medir correntes com o osciloscópio foi necessário

acrescentar em série com a bateria uma resistência de cinco centésimas de *ohm*. Ligou-se a ponta de prova aos terminais da resistência de forma a medir a queda de tensão na mesma. Para se obter o valor da corrente aplicou-se a Lei de *Ohm* uma vez que a queda de tensão e o valor da resistência são valores conhecidos. Aplicando a Lei de *Ohm* a fórmula de conversão fica forma da (VI-1.

$$i(t) = \frac{u(t)}{R} \qquad \textbf{(VI-1)}$$

Na equação *t* corresponde ao tempo em segundos, *i(t)* à curva da corrente ao longo do tempo em *amperes*, *u(t)* à curva da queda da tensão na resistência ao longo do tempo e *R* ao valor da resistência colocada em série.

A Corrente média consumida pela persiana quando sobe é de um *ampere* e quarenta e três centésimas e quando desce é de um *ampere* e vinte e três centésimas. Esta diferença deveu-se particularmente à força gravítica que auxilia na descida e dificulta na subida.

6.1.2 Atuação nas válvulas da janela

Para medir o consumo da atuação das válvulas aplicou-se o mesmo método de medição do consumo da persiana utilizando uma resistência em série com a bateria. Nas Figuras, Figura VI-6, Figura VI-7, Figura VI-8 e Figura VI-9 estão representadas as formas de onda do consumo das válvulas na atuação de cada cenário.

Na Figura VI-6 temos a atuação do cenário *All Opened* onde as duas válvulas abrem.

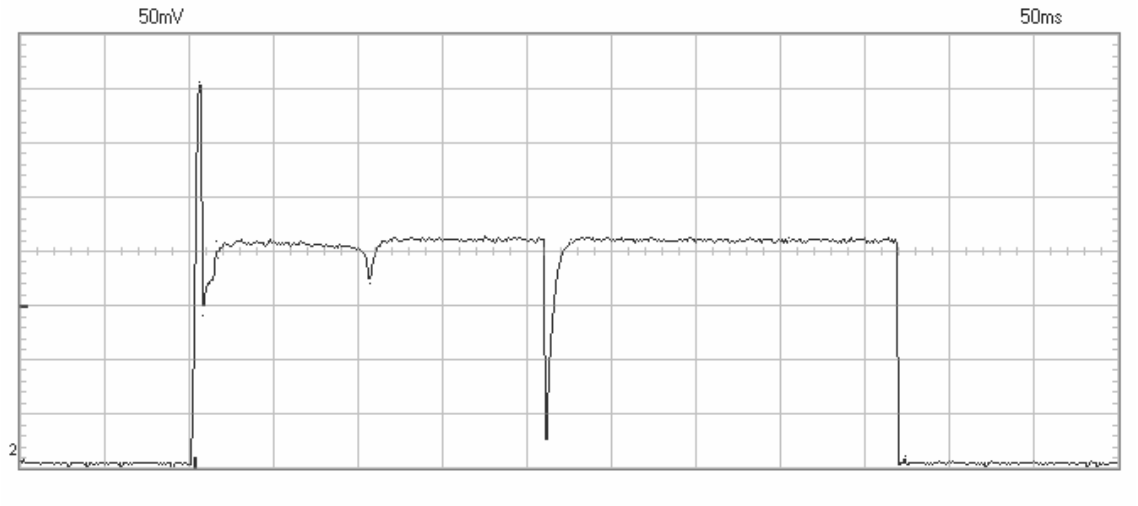


Figura VI-6: Forma de onda, de consumo de corrente com uma resistência em série de 0.05Ω, com as válvulas interior e exterior a abrirem.

No gráfico da figura a atuação começa aos cem milissegundos. Dos cem milissegundos até aproximadamente aos trezentos e dez milissegundos atua a válvula interior e dos trezentos e dez milissegundos até aos quinhentos milissegundos atua a válvula exterior. O consumo médio desta atuação é de um *ampere* e noventa e um centésimos durante cerca de quatrocentos e vinte milissegundos.

Na Figura VI-7 apresenta-se a atuação do cenário *All Closed* onde as duas válvulas fecham.

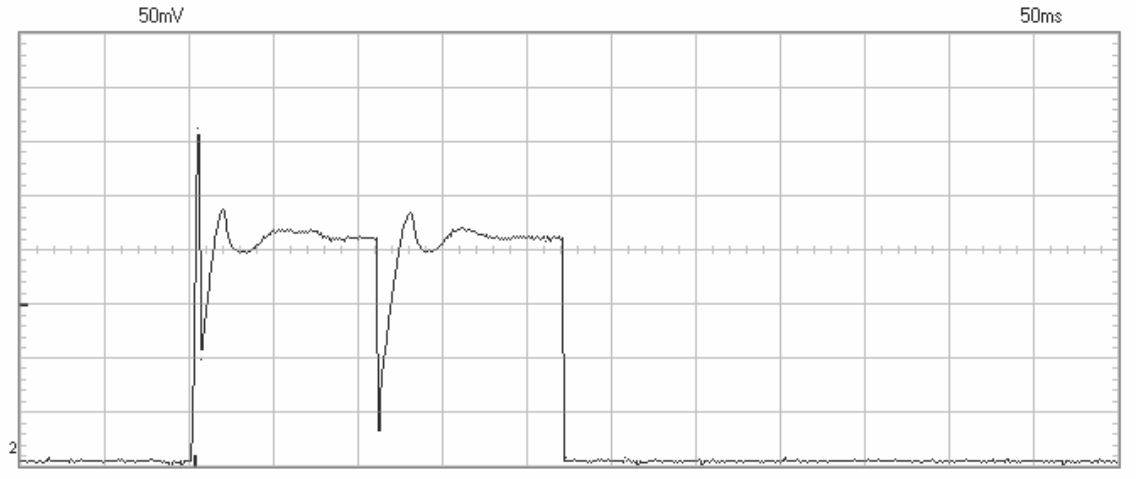


Figura VI-7: Forma de onda, de consumo de corrente com uma resistência em série de 0.05Ω, com as válvulas interior e exterior a fecharem.

No gráfico da figura a atuação começa aos cem milissegundos. Dos cem milissegundos até aproximadamente aos duzentos e dez milissegundos atua a válvula interior e dos duzentos e dez

milissegundos até aos trezentos e vinte milissegundos atua a válvula exterior. O consumo médio desta atuação é de um *ampere* e noventa e um centésimos durante cerca de duzentos e vinte milissegundos. No gráfico observa-se que o tempo de atuação é mais curto ao fechar a válvula do que a abrir. Essa situação deve-se ao facto de no fecho da válvula a força gravítica ajudar no movimento. Assim, ao encurtar o tempo de ação, o consumo diminui. O mesmo já não se pode fazer no ato de abrir a válvula já que, nesse caso, a força gravítica dificulta o movimento.

Na Figura VI-8 apresenta-se a atuação do cenário *Preheat* onde a válvula interior abre e a exterior fecha.

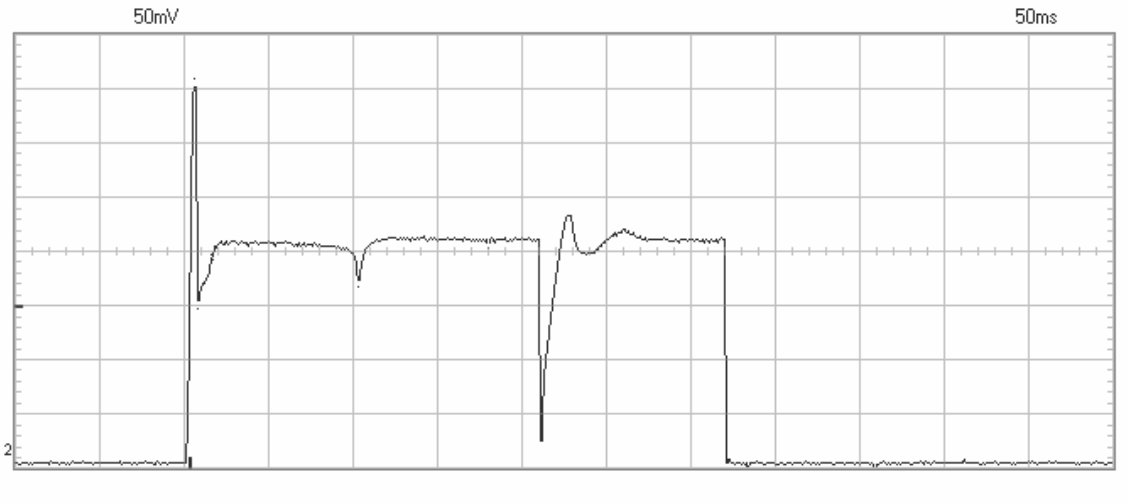


Figura VI-8: Forma de onda, de consumo de corrente com uma resistência em série de 0.05Ω, com as válvulas interior a abrir e exterior a fechar.

No gráfico da figura a atuação começa aos cem milissegundos. Dos cem milissegundos até aproximadamente aos trezentos e dez milissegundos atua a válvula interior e dos trezentos e dez milissegundos até aos quatrocentos e vinte milissegundos atua a válvula exterior. O consumo médio desta atuação é de um *ampere* e noventa e um centésimos durante cerca de trezentos e vinte milissegundos.

Na Figura VI-9 apresenta-se a atuação do cenário *Cooling* onde a válvula interior fecha e a exterior abre.

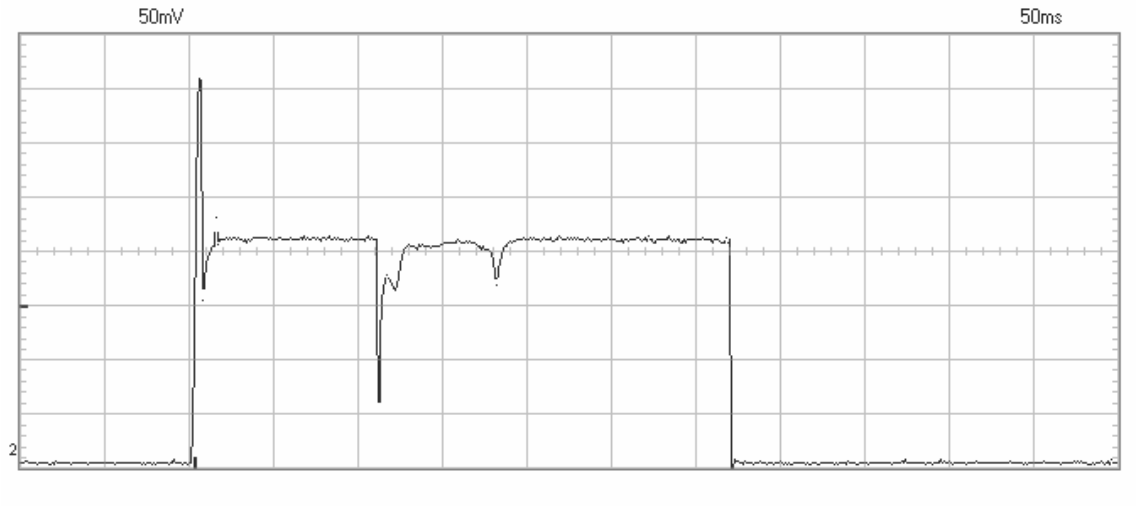


Figura VI-9: Forma de onda, de consumo de corrente com uma resistência em série de 0.05Ω, com as válvulas interior a fechar e exterior a abrir.

No gráfico da figura a atuação começa aos cem milissegundos. Dos cem milissegundos até aproximadamente aos duzentos e dez milissegundos atua a válvula interior e dos duzentos e dez milissegundos até aos quatrocentos e vinte milissegundos atua a válvula exterior. O consumo médio desta atuação é de um *ampere* e noventa e um centésimos durante cerca de trezentos e vinte milissegundos.

6.1.3 Consumo do *EnOcean*

Para medir o consumo do *EnOcean* cada vez que ele acorda periodicamente utilizou-se uma resistência de dez *ohms* em série com a bateria. A resistência teve de ser mais elevada em relação à utilizada para medir o consumo da persiana e das válvulas porque o consumo do *EnOcean* fica na casa de algumas unidades de *miliampere*. Por esse motivo a substituição foi feita para provocar uma queda de tensão que fosse suficientemente alta para ser medida pelo osciloscópio.

Na Figura VI-10 está representada a forma de onda do consumo do *EnOcean* desde o momento em que acorda até ao momento que volta a entrar no modo de "dormir" num dos seus ciclos periódicos.

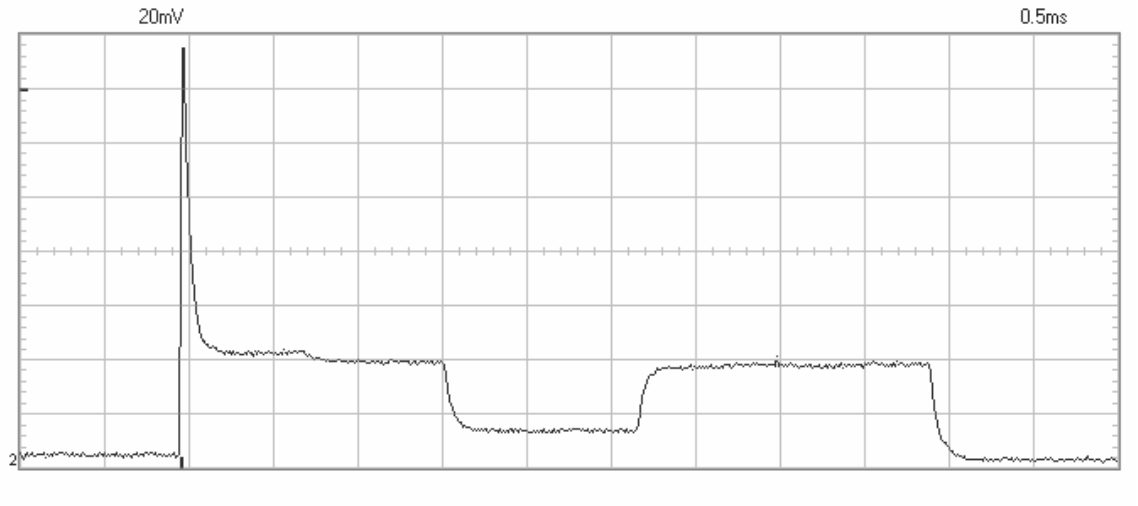


Figura VI-10: Forma de onda, de consumo de corrente com uma resistência em série de 10Ω, com o *EnOcean* a "acordar", a processar e a voltar a "dormir", num dos seus ciclos de 5 em 5 segundos.

Como forma de comparação na Figura VI-11 está representada a forma de onda do consumo do *EnOcean* na versão anterior ao *porting*. Pode-se observar pela comparação dos dois gráficos que houve uma diminuição considerável do consumo do *EnOcean* devido ao facto de a maior parte do seu código foi portado para o ARM. É de salientar que o tempo de cada ciclo do *EnOcean* aumentou de dois segundos para cinco segundos, ou seja, enquanto que na versão antiga o *EnOcean* acordava trinta vezes por minuto agora apenas acorda doze vezes.

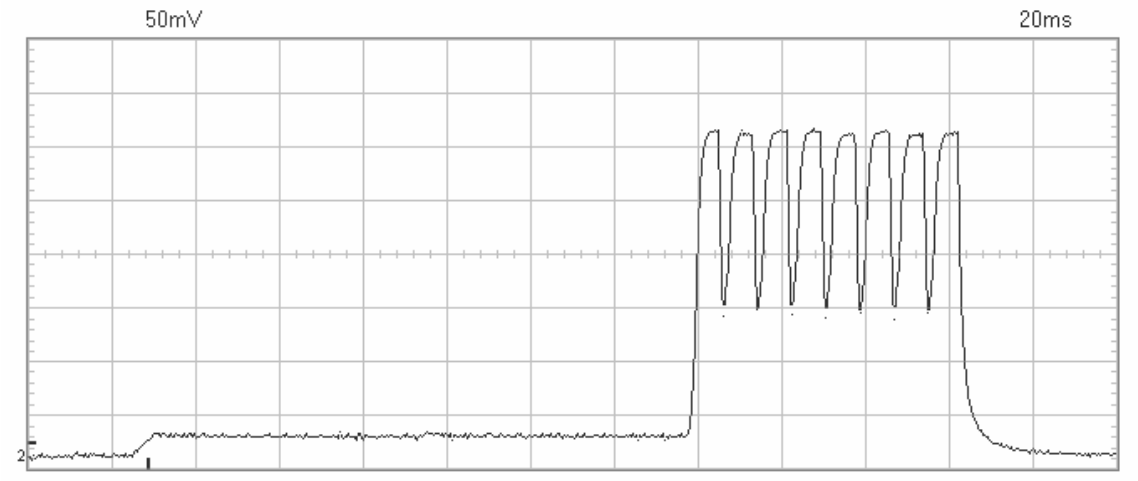


Figura VI-11: Forma de onda, de consumo de corrente com uma resistência em série de 10Ω, com o *EnOcean* na versão antiga a "acordar", a processar e a voltar a "dormir", num dos seus ciclos de 2 em 2 segundos.

6.1.4 Consumo do *ARM*

Tal como na medição do consumo do *EnOcean* na medição do consumo do *ARM* utilizou-se uma resistência em série com a bateria de dez *ohms*.

Na Figura VI-12 está representada a forma de onda do consumo do *ARM* num dos seus ciclos periódicos de duzentos em duzentos segundos.

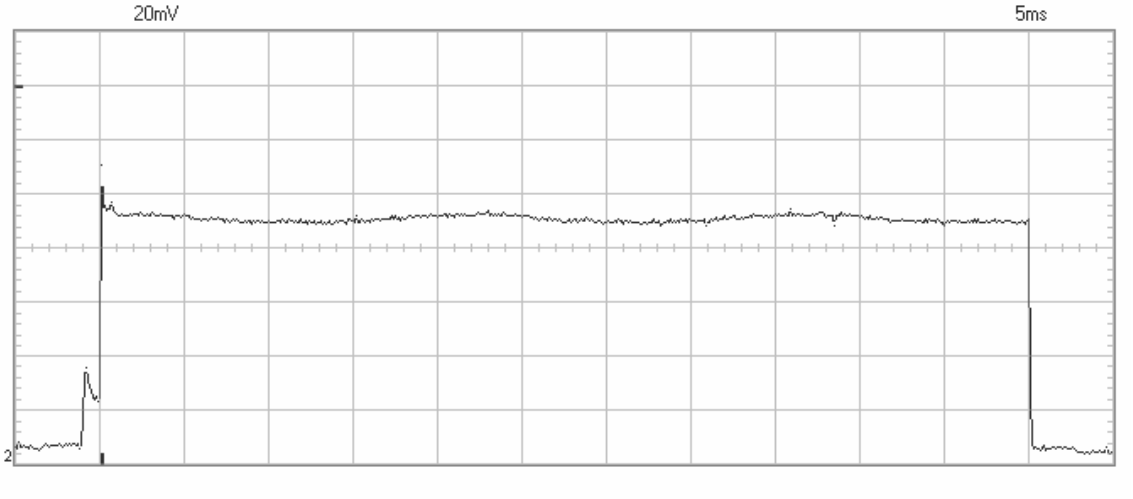


Figura VI-12: Forma de onda, de consumo de corrente com uma resistência em série de 10Ω, com o *ARM* a "acordar", a processar e a voltar a "dormir", num dos seus ciclos de 200 em 200 segundos.

Pela observação do gráfico conclui-se que o consumo do *ARM* é de cerca de nove *miliamperes* durante cinquenta e cinco milissegundos.

Através da observação dos gráficos infere-se que na versão posterior ao *porting* mesmo somando o consumo do ARM com o do *EnOcean obtém-se* consumo médio inferior à versão anterior.

6.2. Resultados de Integração

Nos resultados de integração é registado o consumo de todo o sistema com os microcontroladores a "dormirem" e com as alimentações desativadas dos circuitos de lógica e de potência do *hardware*.

No último subcapítulo deste capitulo é apresentado o resultado do teste à autonomia da bateria.

6.2.1 Consumo com os microcontroladores no modo de *sleep*

Para medir o consumo global do sistema quando os microcontroladores estão no modo de "dormir" foi necessário alterar novamente a resistência colocada em série com a bateria. Dado que o consumo do sistema está em torno de um *miliampere* foi necessário utilizar uma resistência de cem *ohms*.

Na Figura VI-13 e na Figura VI-14 estão representadas as formas de onda do consumo do sistema quando os microcontroladores estão no modo de "dormir" da versão posterior ao *porting* e da versão anterior respetivamente.

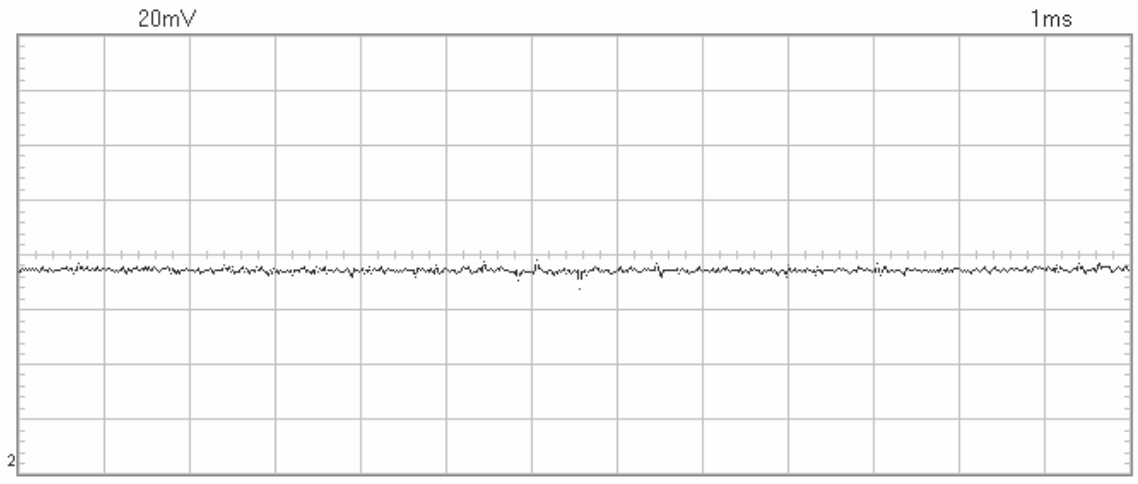


Figura VI-13: Forma de onda, de consumo de corrente com uma resistência em série de 100Ω, com os microcontroladores no modo de *sleep*.

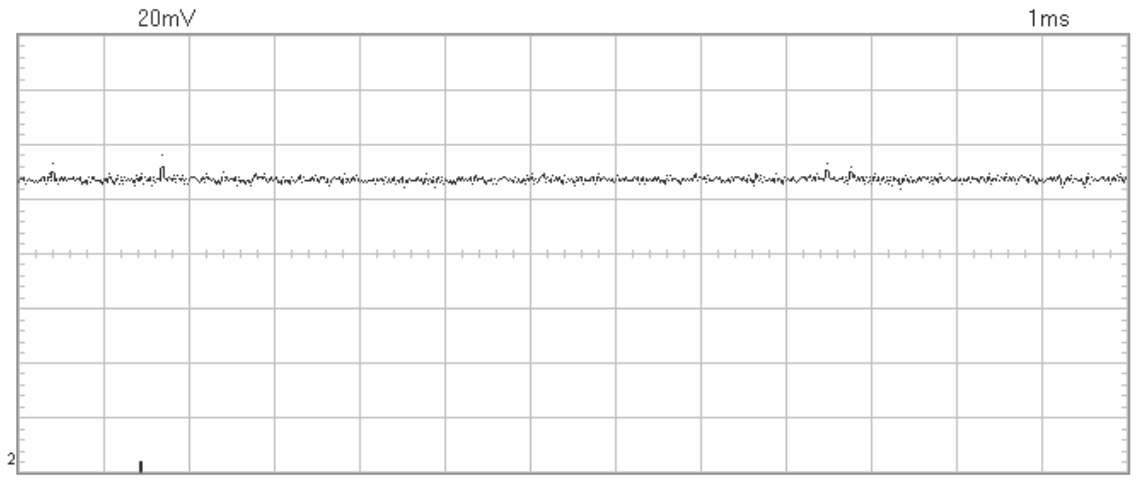


Figura VI-14: Forma de onda, de consumo de corrente com uma resistência em série de 100Ω, com os microcontroladores na versão antiga no modo de *sleep*.

Pela observação dos gráficos o consumo médio do sistema posterior ao *porting* é de setecentos e cinquenta *microamperes*. No sistema anterior ao *porting* o consumo é de um *miliampere* e cinco centésimos, logo existe uma melhoria do sistema posterior ao *porting* de cerca de vinte e oito porcento em relação ao sistema anterior no consumo com os microcontroladores no modo de "dormir".

6.2.2 Autonomia da bateria

Por último foi efetuado um teste à autonomia da bateria. O teste consistiu em movimentar a persiana cinquenta centímetros, alternar o cenário das válvulas entre *All Opened* e *All Closed* e enviar as informações relativas à temperatura, humidade relativa, luminosidade, posição relativa da persiana, posição das abas e cenário das válvulas. Note-se que as instruções descritas são realizadas a cada trinta minutos. Desta forma, obteve-se a autonomia estimada do sistema com todos os elementos ativos periodicamente.

Na Figura VI 15 encontra-se o gráfico do teste onde o eixo das ordenadas representa a tensão nos terminais da bateria e o eixo das abcissas o tempo em dias.

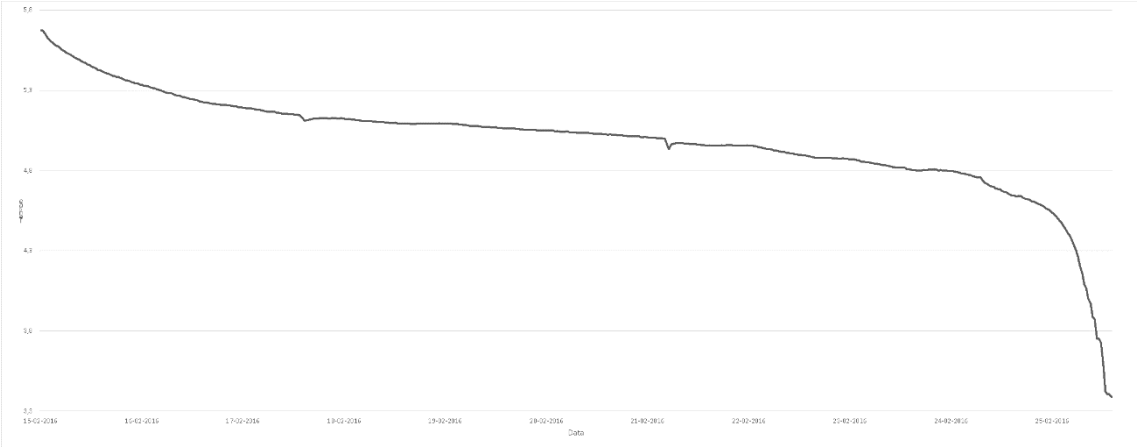


Figura VI-15: Gráfico de descarga da bateria ao longo do tempo com a persiana a movimentar-se 50cm de 30 em 30 minutos.

Com este teste, a bateria com a carga no máximo durou cerca de dez dias e meio.

Capítulo VII - Conclusões e Trabalho Futuro

7.1. Conclusões

No final deste trabalho de dissertação os resultados representam o sucesso do projeto desenvolvido. Os objetivos propostos foram todos cumpridos.

O *porting* foi feito com sucesso e o consumo de energia desceu significativamente em relação à versão original.

O novo código apresenta uma construção modular com *API's* de fácil utilização. Os módulos podem facilmente ser reutilizados num novo *software*. O *software* desenvolvido pode ser portado sem nenhuma dificuldade para qualquer plataforma da família *ARM Cortex M0+*. É possível ainda adicionar novas funcionalidades com alguma facilidade devido à organização do *software*.

Foi ainda acrescentada uma caraterística que não existia na versão anterior: a possibilidade de poder conectar persianas sem *encoder* e obter os mesmos resultados de precisão em relação à posição da persiana e do ângulo das abas.

A interface com o utilizador foi melhorada e simplificada, apresentando uma melhor organização dos menus. Foi acrescentada uma nova funcionalidade no menu, a possibilidade de correr um teste de *hardware* a fim de se poder detetar qualquer anomalia facilmente.

7.2. Trabalho Futuro

Como em qualquer *software,* também neste existe a possibilidade de ocorrer algo imprevisto. Neste momento o modo como o *software* e o *hardware* estão desenvolvidos não existe a possibilidade de reiniciar todo o sistema facilmente. Para o poder fazer é necessário desmontá-lo para se ter acesso ao conetor da bateria e assim desligá-la e voltar a ligá-la para reiniciar o sistema.

Como trabalho futuro, aponta-se a necessidade de se acrescentar a possibilidade de fazer *reset* aos dois microcontroladores (ao *ARM* e ao *EnOcean*) a partir do teclado de forma a não ser necessário desmontar o sistema e assim poder ser executado por qualquer pessoa.

Existe outra melhoria, não menos importante, relacionada com o facto de que este sistema poderá ser instalado em qualquer parte do mundo. Tendo em conta este facto seria importante tornar possível que a atualização do *software* do *ARM* pudesse ser feito via internet a partir da comunicação rádio. Este tipo de atualizações são conhecidas pelo termo OTA, que significa *Over-the-Air*. Assim os problemas que possam aparecer podem ser facilmente resolvidos sem que exista

a necessidade de que uma equipa de técnicos tenha que se deslocar à casa dos clientes para atualizar o *software*.

Capítulo VIII - Bibliografia

[1] ARM, “Cortex-M0+ Processor,” 10 Fevereiro 2016. [Online]. Available: http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php.

[2] H. E. S. Gomes, Climawin demo: Remodelação dos Componentes do Sistema (Hardware e Software), Universidade do Minho, 2014.

[3] MySmartBlinds, “How it Works,” [Online]. Available: http://www.mysmartblinds.com/automated-blinds-how-it-works.php. [Acedido em 10 Dezembro 2015].

[4] FlipFlic, “Automate Your Window Blinds with FlipFlic,” [Online]. Available: http://flipflic.com/.

[5] N. F. S. Cunha, Climawin Demo: protocolo de comunicações, controlador geral e atuação, Universidade do Minho, 2014.

[6] STMicroelectronics, “Datasheet STM32L053R8 Reference manual,” 2015.

[7] Texas Instruments, “Datasheet DRV880x DMOS Full-Bridge Motor Drivers,” 2015.

[8] Texas Instruments, “Datasheet DRV8833 Dual H-Bridge Motor Driver,” 2015.

[9] Sensirion, “Datasheet STH21 v4,” 2014.

[10] Maxim Integrated, “Datasheet MAX44009,” 2011.

[11] EnOcean, “User Manual V1.36 - Scavenger Transceiver Module STM 300 / STM 300U,” 2015.

[12] STMicroelectronics, “Datasheet STM32L053R8 production data,” 2016.

[13] STMicroelectronics, “STM32 Nucleo-64 boards User manual,” 2015.

[14] Atmel, “Datasheet ATtiny441/ATtiny841,” 2014.

[15] M. Barr, Programming Embedded Systems in C and C++, O’Reilly Media, 1999.

[16] J. Yiu, The Definitive Guide to the ARM Cortex-M0, Newnes, 2011.

Apêndice I – Interface com o utilizador

Moving Blind/Flap

Button	Description
Up	**Blind up:** One click, blind move up **(**led green blinking 1 time). **Menu 1:** Pressing 3 seconds, enters in the menu 1 (led blue blinking 3 times every 2 seconds). Is necessary to unlock menus first.
Down	**Blind down:** One click, blind move down (led green blinking 1 time). **Menu 2:** Pressing 3 seconds, enters in the menu 2 (led blue blinking 2 times every 2 seconds). Is necessary to unlock menus first.
Down Flap	**Flap down:** One click, blind move flap down (led green blinking 1 time). **Menu 3:** Pressing 3 seconds, enters in the menu 3 (led blue blinking 3 times every 2 seconds). Is necessary to unlock menus first.
Up Flap	**Flap up:** One click, blind move flap up (led green blinking 1 time). **Menu 4:** Pressing 3 seconds, enters in the menu 4 (led blue blinking 4 times every 2 seconds). Is necessary to unlock menus first.
Up Down Down Flap Up Flap	**Unlock menus:** Pressing 3 seconds in the same time all four buttons (Up, Down, Down Flap and Up Flap) to unlock the menus during 10 minutes (led green blinking 1 time). After 10 minutes menus will be locked again.

Menu 1 - Blind

Button	Description
Up	**Set blind1 size:** When the user choose this menu, (led green blinking 1 time), the window starts moving down and waiting for a second command. When the second command is received (by pressing the same button again), the manual configuration ends (led green blinking 2 times) and the current position will be saves as the maximum length of blind. Finally the blind goes back to the initial position.
Down	**Set blind2 size:** When the user choose this menu, (led green blinking 1 time), the window starts moving down and waiting for a second command. When the second command is received (by pressing the same button again), the manual configuration ends (led green blinking 2 times) and the current position will be saves as the maximum length of blind. Finally the blind goes back to the initial position.
Down Flap	**Reset blinds:** The blind1 and blind2 moving up, till the top, and restarts the position (led green blinking 1 time).
Up Flap	**Next menu:** One click, jump to next menu. **Exit:** Pressing 3 seconds, exit the menu (led red blinking 1 time).
	Exit: No press any key during 30 seconds, exit the menu (led red blinking 1 time).

Menu 2 - Valve

Button	Description
Up	**Closed and Night Cooling :** **All Closed:** It closes both valves (Led green blinking 1 time). **Night Cooling:** It opens both valves (Led green blinking 2 times).
Down	**Cooling Bypass:** It close the indoor valve and open the outdoor valve (Led green blinking 1 time).
Down Flap	**Preheat:** **Preheat 33:** It close the outdoor valve and open indoor valve 33% of the time each 30 minutes (led green blinking 1 time). **Preheat 66:** It close the outdoor valve and open indoor valve 66% of the time each 30 minutes (led green blinking 2 time). **Preheat 100:** It close the outdoor valve and open indoor valve (led green blinking 3 time).
Up Flap	**Next menu:** One click, jump to next menu. **Exit:** Pressing 3 seconds, exit the menu (led red blinking 1 time).
	Exit: No press any key during 30 seconds, exit the menu (led red blinking 1 time).

Menu 3 -

Button	Description
Up	**Learning:** **Learn in:** Learn the window (led green blinking 1 time). **Learn out:** Unlearn the window (led red blinking 1 time).
Down	**Periodic Report:** The window send the periodic report (if success, led green blinking 1 time, if fail led red blinking 1 time).
Down Flap	**Test:** Execute a sequence of test commands
Up Flap	**Next menu:** One click, jump to next menu. **Exit:** Pressing 3 seconds, exit the menu (led red blinking 1 time).
	Exit: No press any key during 30 seconds, exit the menu (led red blinking 1 time).

Menu 4 -

Button	Description
Up	**Free:**
Down	**Free:**
Down Flap	**Free:**
Up Flap	**Next menu:** One click, jump to next menu. **Exit:** Pressing 3 seconds, exit the menu (led red blinking 1 time).
	Exit: No press any key during 30 seconds, exit the menu (led red blinking 1 time).