Bluetooth para controlo de acesso de um contentor de resíduos

Gabriel Pontífice Vila

Orientador

Paulo Jorge Coelho Marques

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciatura em Engenharia Eletrotécnica e das Telecomunicações, realizada sob a orientação científica do Professor Adjunto Paulo Jorge Coelho Marques, do Instituto Politécnico de Castelo Branco.

Outubro 2024

Composição do júri

Presidente do júri

Professor Coordenador Rogério Pais Dionísio

Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Vogais

Professor Hugo Rafael de Almeida e Marques, Professor Adjunto

Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Dedicatória

Dedico este projeto a todos os que me ajudaram na realização do mesmo. Este projeto não é só meu, é também vosso, pois o conhecimento e a experiência partilhados comigo, fizeram a diferença.

É muito gratificante saber que posso contar com o apoio de todos aqueles que, mesmo não tendo um conhecimento na área, são capazes de ver outra perspetiva. É bastante importante para desbloquear a mente e analisar outras possibilidades, que sejam mais fáceis, eficazes e descomplicadas.

Obrigado por tudo.

Agradecimentos

Em primeiro lugar, agradeço a família pelo todo o apoio que me tem dado nestes anos todos, pois foram eles o meu suporte para a realização deste curso. A uma pessoa em especial, o meu pai. Mesmo que ele não esteja presente, tenho a certeza que estaria orgulhoso de ver o seu filho a formar-se. Obrigado, Pai.

Aos amigos, pelo conhecimento que partilharam comigo, pela ajuda que me deram sempre que precisei. Obrigado pela vossa disponibilidade e ajuda.

Também quero agradecer a EVOX, empresa onde trabalho e onde foi proposto este projeto. Obrigado por me terem dado o vosso conhecimento, a vossa ajuda, assim como todo o material que precisei para a realização deste projeto.

Em último lugar, mas não menos importante, quero agradecer ao professor Paulo Marques, não só por disponibilizar-se a orientar-me no projeto, mas como também aconselhar-me o melhor caminho a seguir.

Resumo

Atualmente, os mecanismos de controlo de acesso em contentores de resíduos são efetuados através da interação de um cartão NFC. Este mecanismo, apesar de funcional, não permite ao utilizador interagir através do *smartphone*. Ou se permite, obriga o usuário a tocar com o smartphone no contentor de resíduos. Com recurso através de tecnologia Bluetooth é possível efetuar esta autenticação, utilizando uma App que seja multiplataforma e que faça a ponte entre o utilizador e o contentor e permita a autenticação e desbloqueio da tampa do contentor de resíduos.

Atualmente, a eletrónica do controlo de acessos da EVOX já possui um módulo BLE. No entanto, existem algumas limitações funcionais que não permitem uma integração ágil com Apps via Bluetooth. O projeto visa o desenvolvimento de uma placa eletrónica e integração de um novo módulo mais potente que, pode até, fazer uso de sistemas de muito baixo consumo para troca de informações.

Relativamente à aplicação, esta deverá permitir a autenticação do utilizador e, posteriormente, este poderá usar esta autenticação para se identificar no contentor através do controlo de acesso. A app deverá detetar a presença do contentor e, em seguida, permitir o desbloqueio do contentor através de um conjunto de mensagens de autenticação que permitam com que o sistema seja robusto e tenha segurança de dados.

Palavras-chave

UART - Baixo consumo – Bluetooth - C/C++

Abstract

Currently, waste container access control mechanisms operate via NFC card interaction. While functional, this mechanism does not allow users to interact through their smartphones. And if it does, it requires the user to touch their smartphone to the waste container. By using Bluetooth technology, authentication can be achieved through a cross-platform app that acts as a bridge between the user and the container, enabling authentication and unlocking the container lid.

At present, EVOX's access control electronics already include a BLE module. However, there are functional limitations preventing seamless integration with Bluetooth-enabled apps. The project aims to develop an electronic board and integrate a more powerful module, potentially utilizing ultra-low-power systems for data exchange.

Regarding the application, it should enable user authentication, which the user can subsequently use to identify themselves at the container via access control. The app should detect the presence of the container and then allow container unlocking through a set of authentication messages that ensure a robust and secure system.

Keywords

UART – Low energy – Bluetooth – C/C++

Índice geral

Índice de figuras

Lista de abreviaturas, siglas e acrónimos

µP- MicroProcessadores

ADC- Analogic to Digital converter

DAC- Digital to Analogic converter

GND- Ground

I2C- Inter-Integrated Circuit

ICSP-In-Circuit Serial Programming

I/O- Input/Output

LDO- Low-dropout regulator

MCLR- Master Clear

MHz- Mega Heartz

PCB – Printed Circuit Board

PGC -Program Clock

PGD -Program Data

QFN-S- Quad Flat No leads

RAM – Random Access Memory

SMD- surface-mount device

SOIC- Small-Outline Integrated Circuit

SSOP- Shrink Small-Outline Package

THT- through-hole technology

UART- Universal Asynchronous Receiver/Transmitter

UUID- Universally unique identifie

VCC- Alimentação

1. Introdução

Os biorresíduos estão presentes sempre que preparamos alimentos e descartamos os seus restos. A nível nacional, estes representam, em média, quase 37% do nosso lixo comum, se não forem separados do restante desperdício, acabam depositados em aterros sanitários ou nas águas potáveis. Esses mesmo resíduos podem ser úteis na produção de biogás e ou fertilizantes.

A Diretiva Quadro Resíduos, estima que a partir de 2030, os aterros não passam a aceitar quaisquer resíduos apropriados para reciclagem ou outro tipo de valorização, nomeadamente resíduos urbanos.

Atualmente, os mecanismos de controlo de acesso a contentores de resíduos são efetuados através da interação com um cartão NFC. Este mecanismo obriga o utilizador a colocar o *smartphone* junto do leitor NFC no contentor.

O objetivo deste projeto é fazer uso da tecnologia *Bluetooth* para autenticação e desbloqueio da tampa do contentor de resíduos, utilizando uma App multiplataforma.

A implementação do projeto divide-se em duas fases que são as seguintes: a primeira a fase é relativa ao desenvolvimento da eletrónica e a segunda fase é diz respeito ao desenvolvimento da programação.

Na primeira fase, o desenvolvimento da *Printed Circuit Board* (PCB) foi concebido para criar uma *board* de baixo consumo, para que todos os componentes tivessem menor consumo possível durante o seu funcionamento. Para alimentar esta *board* uso uma bateria de 3.6V de maneira a que este projeto dure o máximo tempo possível, de forma autónoma.

Fez-se uma análise dos mais variáveis componentes e decidiu-se usar o PIC24FJ128GA202 por ter uma boa relação de consumo/memória.

Para o módulo *Bluetooth*, usou-se o RNBD451PE que é um componente de baixo de consumo e recente, pois já funciona com tecnologia 5.2.

Na segunda fase do projeto, fez-se o desenvolvimento do *firmware*. A plataforma utilizada para o desenvolvimento foi o MPLAB. Podia ter escolhido outro software para o desenvolvimento, mas como o PIC é da *Microchip technology* e o MPLAB também é desenvolvido pela *Microchip* torna mais fácil a interligação.

Desenvolveu-se uma App para interligar com o *Bluetooth*. Esta App foi desenvolvida em HTML e CSS, permitindo destrancar a tampa do contentor.

2. Componentes usados

2.1 Microprocessador

Existem muitas famílias de microprocessadores (µP), onde cada número ou conjunto de letras tem o seu significado. Iniciemos por decifrar este nome: **PIC**- marca que o fabricante dá aos seus µP; **24-** arquitetura, indica que pertence a família dos 16 bit; **FJ-** Família da memoria flash; **128-** Tamanho da memoria *flash* ou memória de programação (128K bytes); **GA2-** Propósito geral, isto é, não é específico para uma determinada tarefa, tendo assim todos ou quase todos os recursos disponíveis de um µP; **02-** Número de pinos que possui (28 pinos).

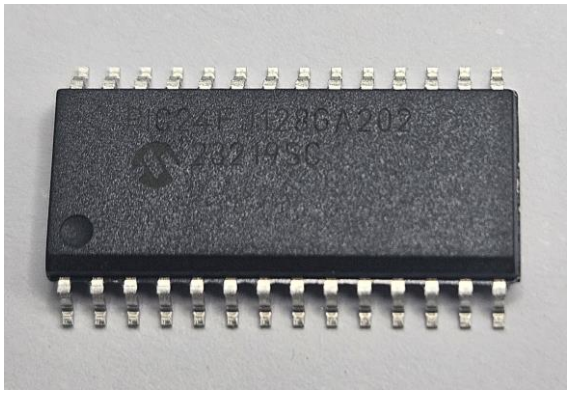

Figura 1- PIC24FJ128GA202, fabricante *Microchip Technology* [1].

Agora que já sabemos interpretar o nome do componente, segue a parte técnica, as suas especificações internas:

Device	Memory		Pins	Analog Peripherals			Digital Peripherals							Deep Sleep w/V_{BAT}	AES/DES Cryptographic
	Program Flash (bytes)	Data RAM (bytes)		10/12-Bit A/D (ch)	Comparators	CTMU (ch)	Input Capture	Output Compare/PWM	I^2C^{TM}	SPI	UART w/IrDA® 7816	EPMP/PSP	16-Bit Timers		
PIC24FJ128GA204	128K	8K	44	13	3	13	6	6	2	3	4	Y	5	Y	Y
PIC24FJ128GA202	128K	8K	28	10	3	10	6	6	2	3	4	N	5	Y	Y

Figura 2- Especificações técnicas µP

Com esta imagem, conseguimos, de uma forma rápida, ver se este µP é compatível com o projeto a realizar. Para além disso, é possível ver a análise à memória de programação; da RAM; a quantidade de pinos, se é a pretendida; o número de DAC e ADC; as várias tecnologias de comunicação e a sua quantidade; entre outras informações necessárias.

Acrescento mais informação relevante a este componente pela sua ficha técnica: É um componente de alta performance e de baixo consumo, tem um oscilador interno de 8MHz, podendo ir até 32MHz com cristal externo, alimentação pode variar 2.0V até 3.6V.

Numa última análise, tendo já uma noção da quantidade de pinos a serem usados, é observada a pegada do componente, isto é, o *footprint* que nada mais é senão o tipo de construção dos pinos com que ele é realizado. Neste caso, usei tipo SOIC de 28 pinos SMD.

A montagem deste componente pode ser em SMD (*surface-mount device*) ou THT (*through-hole technology*), por outras palavras, SMD é um tipo de montagem de superfície que não precisa de aberturas para a sua soldadura. Já nos THT, devem existir fendas para que esse componente seja soldado pela parte de baixo.

Figura 3- SMD vs THT

Os tipos de pinos mais comuns são os seguintes: SOIC, SSOP, QFN-S, entre outros existentes. Basicamente, referem-se à posição em que os pinos estão dispostos no µP. Estes podem encontrar-se mais afastados (SOIC), encurtados (SSOP) ou até mesmo por de baixo do componente (QFN-S).

Figura 4- Diferentes pinagens

Podemos falar dos vários protocolos de comunicação que geralmente os µP têm em sua posse, como I2C, UART, ou RS232, RS485. Estas duas formas de comunicação em série são normalmente usadas para longas distâncias, podendo também ser utilizadas para curtas, dependendo da aplicação. Por norma, as mais usadas são I2C e UART.

Começando por falar no I2C (*Inter-Integrated Circuit*), este possibilita a utilização de grande quantidade de componentes padronizados, os quais podem realizar diversas funções, além de possibilitar a troca de informações entre sensores/µP. Funciona, geralmente, com um master, ou seja, aquele que comanda o *clock* e os seus escravos. Normalmente, o master é o µP e os seus escravos são os sensores.

UART (*Universal Asynchronous Receiver/Transmitter*), é um protocolo de comunicação série usado para transferir dados entre dispositivos eletrônicos.

A comunicação UART envolve dois componentes principais: um transmissor (Tx) e um recetor (Rx). O (Tx) faz a transmissão de *bits*, e(RX) recebe a informação de *bits*, permitindo comunicação bidirecional.

Por fim, há que de falar dos DAC e ADC, pois são bastantes importantes em sistemas embebidos. Embora não sejam usados neste projeto, deixo aqui referido, uma vez que a sua relevância é bastante grande.

O DAC *(Digital Analogic Converter*) refere-se a conversão de um sinal digital, isto é, zeros e uns, para transpor para analógico. Geralmente é usada 0V a 5V.

ADC (*Analogic Digital Converter*), basicamente é o oposto do DAC, onde entra um nível analógico para ser convertido para digital.

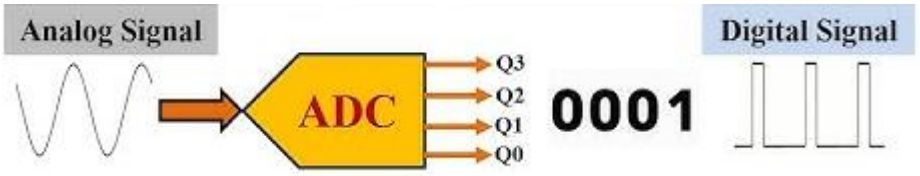

Figura 5- Exemplo do ADC

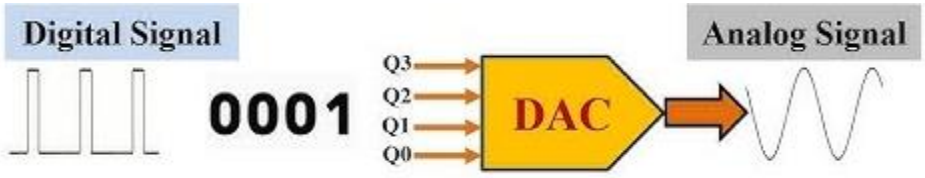

Figura 6- Exemplo do DAC

2.2 *Bluetooth*

O módulo *Bluetooth* está projetado para aplicações que exigem comunicação sem fio de baixo consumo de energia. O módulo selecionado foi o RNBD451PE, que é compatível com a norma Bluetooth 5.2, o que permite maior velocidade de dados e maior alcance, em comparação com versões anteriores.

O módulo *Bluetooth* conta com uma antena integrada o que simplifica o projeto. A sensibilidade do recetor é de até -102 dBm, e a potência de transmissão pode chegar a 12 dBm, garantindo uma boa qualidade de sinal e alcance rádio, mas com essa potência, a bateria seria gasta rapidamente, logo não será usada a potência de 12dBm mas sim de 1dBm, no que ajuda a conservar mais bateria e ter um alcance menor.

Dependendo da configuração de potência, o consumo e também a taxa de bits transmitidos, são alterados. Abaixo, são apresentados exemplos de algumas possíveis configurações:

- -95 dBm, 1 Mbps
- -92 dBm, 2 Mbps
- -102 dBm, 125 Kbps

Figura 7- Módulo *Bluetooth* RNBD451PE, fabricante *Microchip Technology* [2].

Os seguintes gráficos são representativos da potência recebida e da potência transmitida em relação a frequência. Podemos observar que se for configurado para um certo valor é bastante fiel a sua configuração. É de ressalvar que para qualquer valor que o Bluetooth suporte, vai ter o mesmo comportamento.

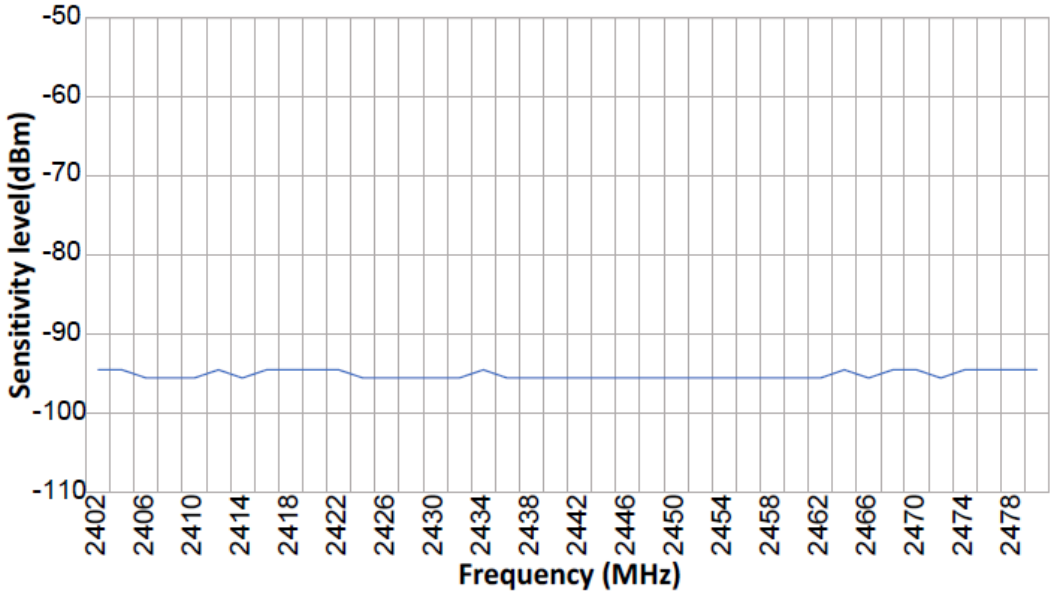

Figura 8- Sensibilidade de receção vs frequência

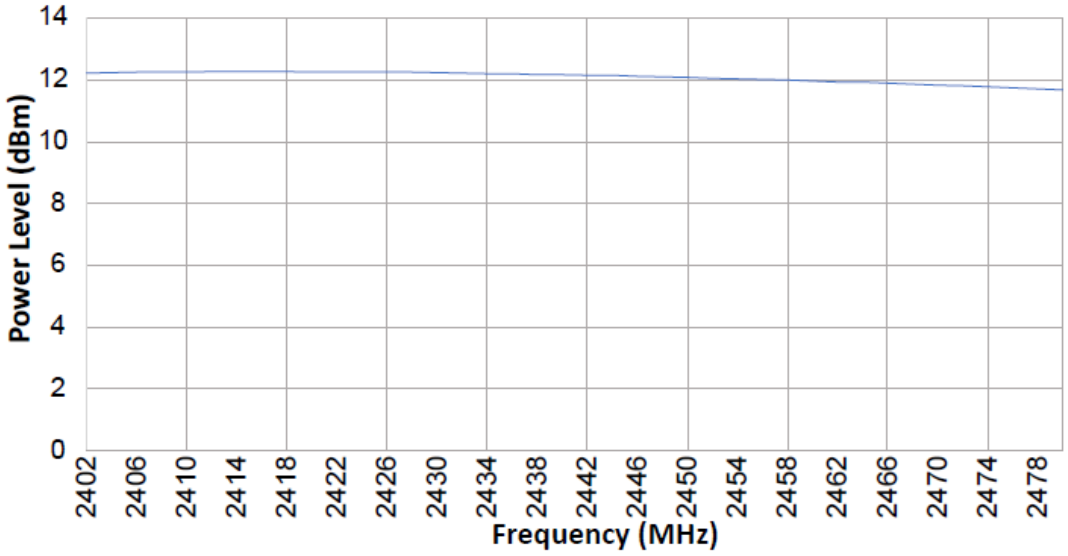

Figura 9- Potência de transmissão vs frequência

Este gráfico representa o consumo versus a temperatura, as cores de cada linha no gráfico representam a tensão, conforme na legenda.

Como é normal, com a temperatura alta, o *Bluetooth* vai consumir mais, mas isso acontece em toda a eletrónica, numa maneira geral.

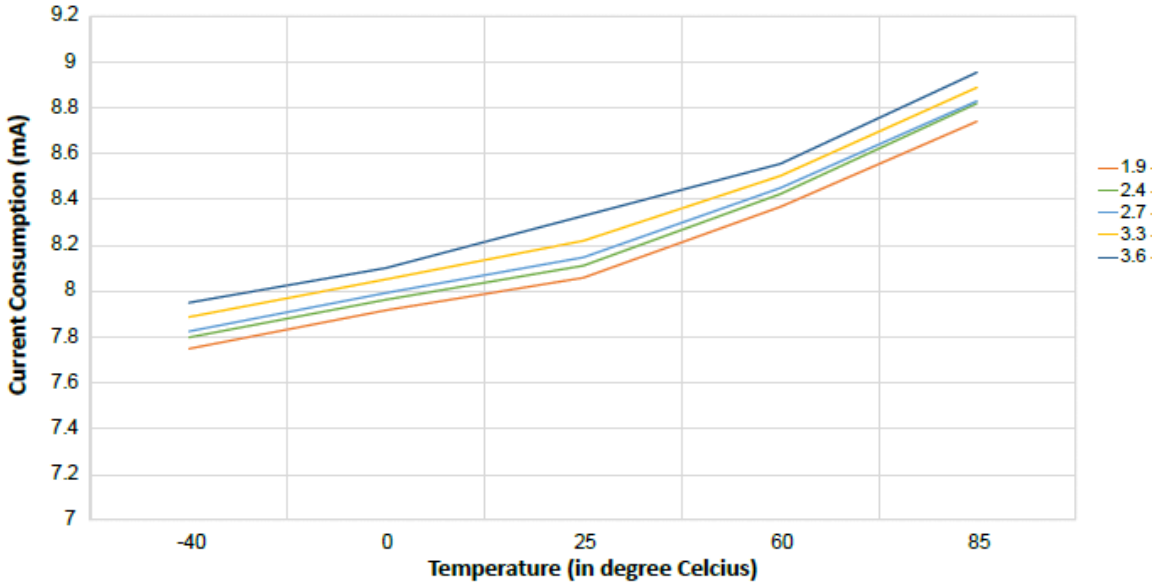

Figura 10- Consumo em funcionamento

2.3 LDO (*Low-dropout regulator*)

O LDO tem como sua função reduzir a tensão de 3.6V para 3.3V. A razão pela qual isto acontece é porque a alimentação usada, neste caso uma bateria, é de 3.6V. Para este projeto foi usada a tensão típica de 3.3V do *Bluetooth* e µP.

Existem alguns LDO que conseguem ter uma saída regulada, isto é, têm um pino extra para que possa fazer um divisor resistivo que consiga a tensão desejada na saída.

Nesse caso, tem uma saída fixa, o que quer dizer que não tem esse pino extra e a sua tensão de saída é sempre 3.3V.

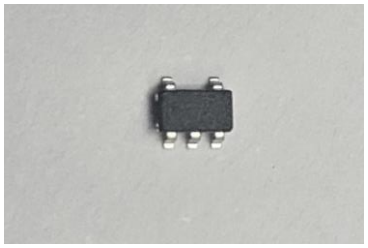

Figura 11- LDO TLV755, fabricante *Texas Instrument* [3].

		MIN	MAX	UNIT
V_{IN}	Input supply voltage	2.2	5.5	V
V_{EN}	Enable input voltage	0	5.5	V
I_{OUT}	Output current	0	250	mA
T_J	Junction temperature	–40	125	°C
T_A	Ambient temperature(3)	–40	85	°C

Figura 12- Especificações técnicas do LDO

2.4 Condensadores, Resistências, Led

Foram usados vários condensadores de desacoplamento. Estes condensadores servem para que o ruído (alta frequência) seja filtrada e não interfira com o funcionamento dos componentes.

Utilizaram-se condensadores de 10µF, 100ηF, 4.7µF. Para estes condensadores, o fabricante recomenda que sejam colocados com estes mesmos valores para o seu bom funcionamento. Tanto como no µP no *Bluetooth* e LDO, o fabricante sugere sempre um determinado valor de resistência e ou condensadores.

Para além da sugestão que o fabricante dá acerca dos condensadores, o mesmo também recomenda o uso das resistências e o seu valor.

Conseguimos ver na figura 13, que foi recomendado o uso de resistências para MCLR (*Master Clear*) e os condensadores de desacoplamento em todos os VSS (*Voltage Source Supply*) para VDD (*Voltage Drain Drain*).

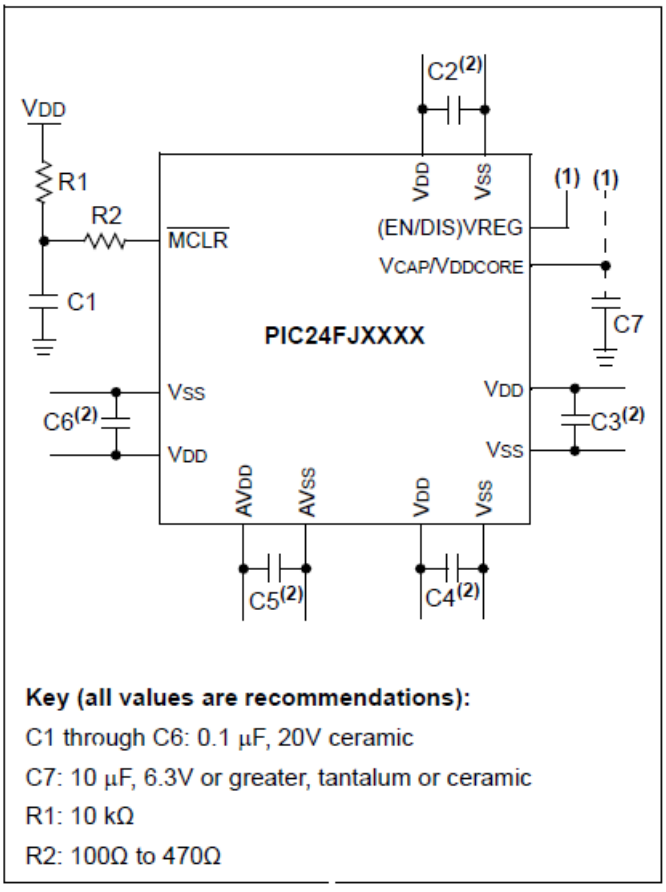

Figura 13- Ligação recomendada

3. Desenho da PCB

Com análise feita dos componentes necessários para o projeto e a verificação de qual se enquadra melhor, chega a altura do desenho da *board*. Este desenho divide-se em duas partes: a primeira desenhar o *schematic* em português esquematismo. Basicamente, é onde se começa por inserir os componentes, tal como o *footprint* correto e também definir a interligação dos vários componentes escolhidos, previamente. Para além disso, também se começa por se decidir quais as portas que vão ser usadas no µp e *Blutooth*.

De seguida, começa-se por fazer a montagem e posicionar os componentes, seguindo as normas para um bom funcionamento da *board*. Há certos componentes que têm de ser colocados de uma forma especial, pois alguns condensadores devem estar o mais próximo possível dos pinos do µP, entre outras normas que irei explicar ao longo do relatório.

Utilizei uma ferramenta gratuita chamada *Eagle* que pertence à *Autodesk*. Podia ter escolhido usar outras ferramentas, que o resultado seria igual.

3.1 *Schematic*

Começo por explicar uma ferramenta que facilita bastante o desenho do *schematic*, o nome dela é *Label*. Este tipo de ferramenta permite que um ponto seja ligado a outro ponto sem que precise usar uma linha visual. Estes dois tipos de *label* fazem a mesma coisa, o que difere é a visualização.

Figura 14- Exemplo de *label*

E como sabemos que estes dois pontos estão verdadeiramente interligados? Em primeiro lugar, devemos verificar se os pontos têm a mesma designação, isto é, *label*. Caso tenham designações diferentes procedemos à sua configuração para que consigam ter o mesmo termo. Posteriormente, ao usarmos a ferramenta "mostrar" no *Eagle*, esta destaca onde esses *labels* estão interligados.

Como podemos observar na figura 15, conseguimos ver que a cor está mais brilhante, em comparação com a figura 14. Assim temos a perceção que estão verdadeiramente conectados.

Figura 15- Exemplo de *label* conectado

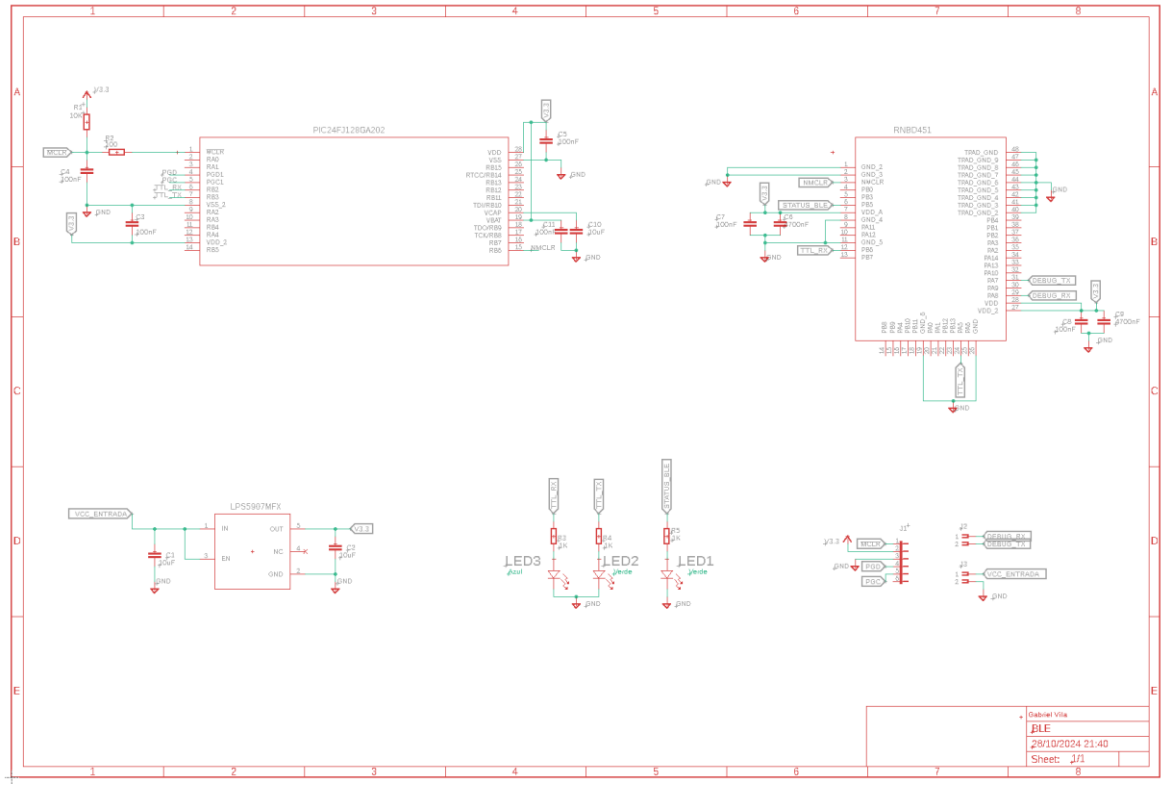

Figura 16- *Schematic*

A figura 16 mostra-nos todo o *schematic* feito. Para ser mais fácil a leitura e sua interpretação, divido em 4 partes, onde explico cada uma delas com mais detalhe.

3.1.1 Schematic LDO

Absorvemos da esquerda para a direita o primeiro *label* com o nome VCC_Entrada. Este *label* vai ser ligado ao *jumper* da entrada onde é ligado a alimentação do circuito. De seguida, vemos um condensador de 10µF. Este é um condensador de desacoplamento para filtrar o ruído, por outra maneira de dizer, filtrar as altas frequências e negar os elevados picos de tensão indesejados. Este condensador está ligado na alimentação e segue para a *ground* (GND), isto é, o lado negativo do circuito. Se GND é negativo então VCC é o lado positivo.

De seguida temos o LDO, responsável pela redução de tensão, conforme expliquei anteriormente. Na sua saída, visualizamos outro condensador de desacoplamento com a mesma função do anterior. Por fim, apresenta-se um *label* no pino de saída com nome de V3.3. É neste pino onde vão ser ligados o µP e o *Bluetooth*, pois a alimentação deles são é de 3.3V.

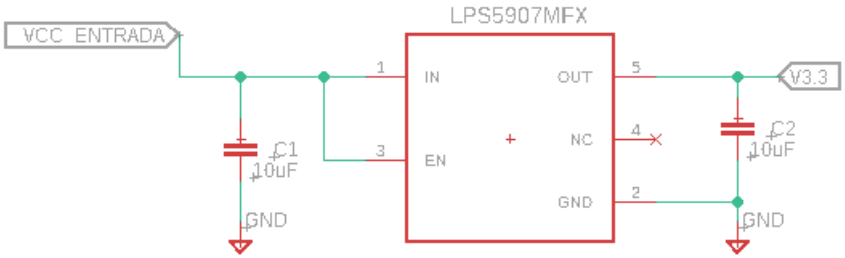

Figura 17- *Schematic* LDO

3.1.2 *Schematic* µP

Conseguimos observar que a alimentação do PIC24 é de 3.3V, onde está o *label* V3.3, vindo da saída do LDO. Nesse ponto, temos algumas resistências e outro *label* com nome de MCLR. A função da resistência de 10KΩ é para que o PIC seja capaz de identificar o estado lógico do pino, isto é, se está a 1 ou 0.

Numa forma mais clara de entender, se o pino está ativo, equivale a "1"; se está desativo, equivale a "0". Quando se usa o pino MCLR e está ativo, dá-se o *reset* ao PIC. Este *reset* faz com que a memória do PIC seja limpa e, desta forma, torna-se possível programar. Usando a tecnologia ICSP (*In-Circuit Serial Programmning*) usamos também o pino PGD (*Program Data*) e PGC (*Program Clock*) e *Ground*.

Conforme conseguimos observar pela figura 13, temos pinos específicos para esse efeito.

Como existem pinos específicos para programação, também existem pinos específicos para portas I/O (In/Out). Temos RA0 a RA4 e RB0 a RB15. Estes pinos são pinos de I/O; são configuráveis e usados para ligar sensores ou outros componentes elétricos.

O pino VCAP desempenha um papel importante na estabilização da tensão interna do regulador de tensão do microcontrolador. Muitos microcontroladores operam internamente a uma tensão mais baixa por exemplo, 2.5V do que a tensão de alimentação fornecida externamente por exemplo 3.3V, logo, este pino serve para que o µP opere também a 3.3V.

O pino VBAT é utilizado para fornecer uma fonte de alimentação alternativa e contínua para a memória de dados críticos, garantindo que esses componentes continuem a funcionar e preservar informações mesmo quando a alimentação principal do microcontrolador (VDD) não está presente.

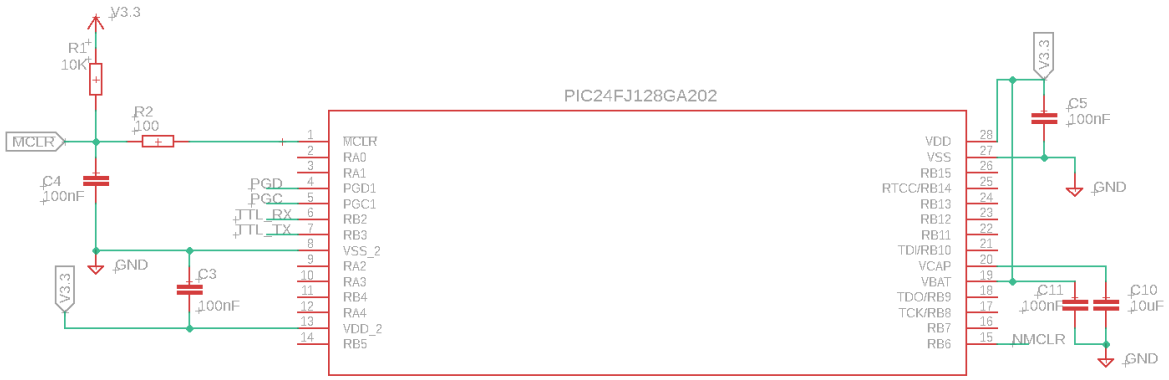

Figura 18- *Schematic* µP

3.1.3 *Schematic Bluetooth*

Em uma análise rápida os condensadores, estes são de desacoplamento e tem a mesma função dos condensadores do µP.

De seguida, temos os vários *labels* que vão ser ligados aos mais diversos pontos do *schematic*. Por exemplo, o *label* NMCLR ligar-se-á ao µP; o TTL_RX e TTL_TX também vão ser ligados ao µP. O DEBUG_TX e DEBUG_RX vao ser ligados a um *jumper*. Estes dois pinos servem para visualizar os comandos que estão a ser enviados do µP para o *Bluetooth*.

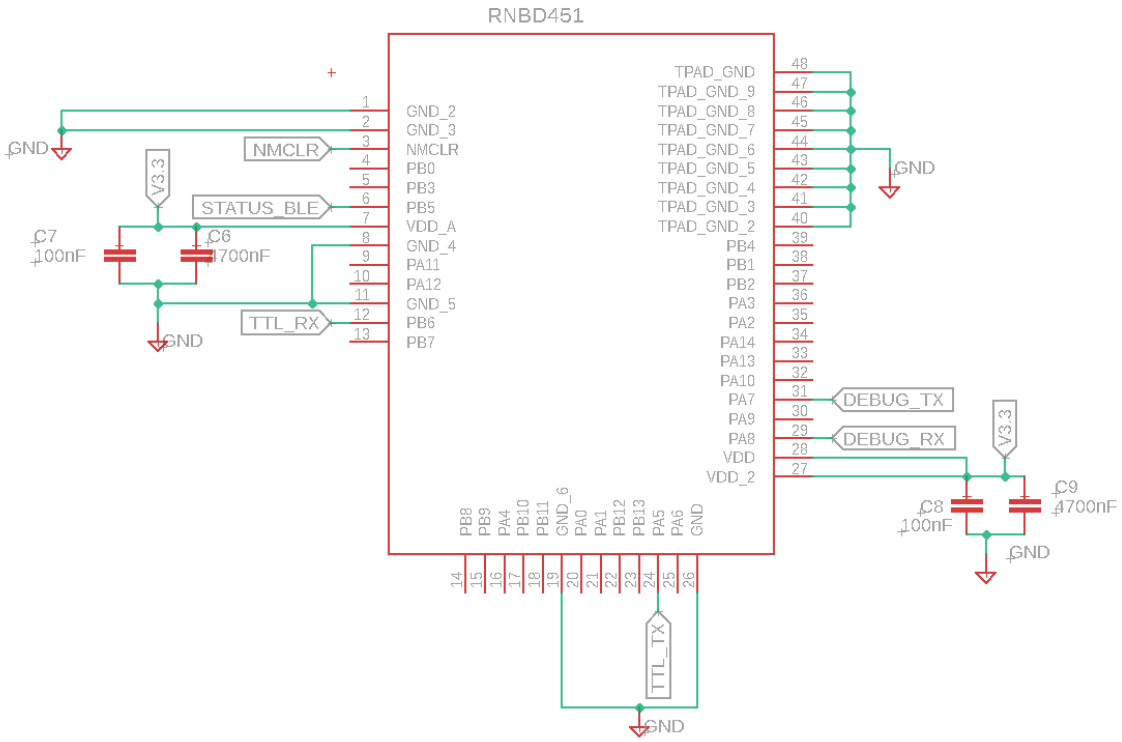

Figura 19- *Schematic bluetooth*

3.1.4 *Schematic LED/Jumpers*

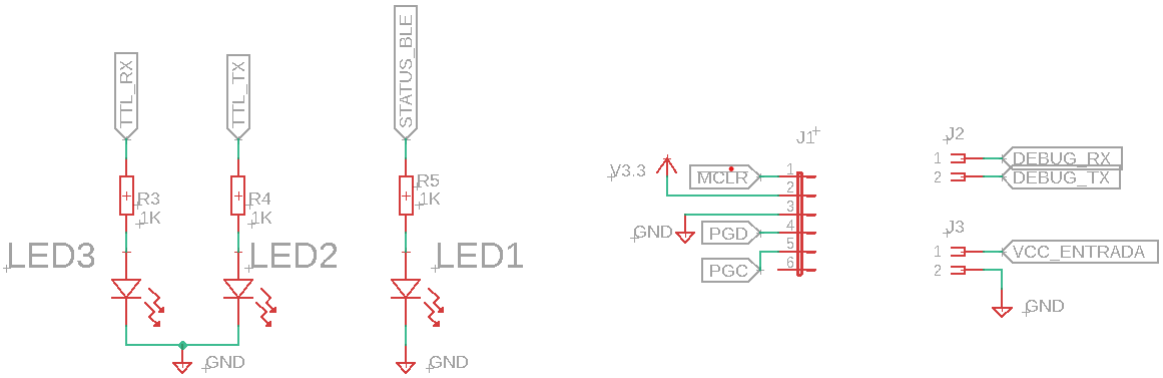

Figura 20- *Schematic LED e Jumpers*

No caso do LED 2 e 3, estes LED´s servem para nos dar um *feedback*, ou seja, informa-nos se esta haver comunicação entre o µP e o *Bluetooth*. O LED 1 notifica-nos o estado do *Bluetooth*.

De seguida, temos um *jumper (J1)* da ficha de programação. Esta ficha serve para programar o µP.

Por fim, temos mais dois *jumpers*, J2 e J3, que são muito parecidos com o da ficha de programação, no entanto, só têm dois buracos onde será soldado um *header*.

O J3 ou alimentação do circuito será soldado diretamente na *board*.

3.2 *Board*

O próximo passo segue-se à colocação dos componentes e também das vias. Neste passo, temos de ter em conta certas considerações para que funcione tudo com conformidade. É preciso seguir normas, especificações técnicas e ver o que os fabricantes recomendam.

Devemos ter atenção, primeiramente, as normas das vias; considera-se 0,40mm para alimentação e *ground*. 0,3mm para as vias de dados. Os condensadores de desacoplamento devem ficar mais perto do seu pino onde se inicia a ligação, nomeadamente, C3, C5, C6, C7, C8, C9, C10, C11. As resistências do MCLR, R1 e R2 também são importantes estar o mais próximo do pino, facilita anulação do ruido que pode aparecer junto deste pino. De seguida, o fabricante do *Bluetooth* não recomenda o uso de *ground plane* na parte de cima do mesmo, isto porque a radiação da antena seria atenuada. O passo mais acertado é o uso da *ground plane* em toda a *board*. E o que é isto de *ground plane*? Em português significa plano de terra. Ou seja, quer dizer que existe uma camada que pertence toda à *ground*. Esta camada está na parte de baixo da *board*, isto é, a *board* tem duas camadas, superior ou inferior, onde se podem interligar as vias de um lugar para o outro. A mudança de camada é feita pelo um furo pequeno que faz essa interligação.

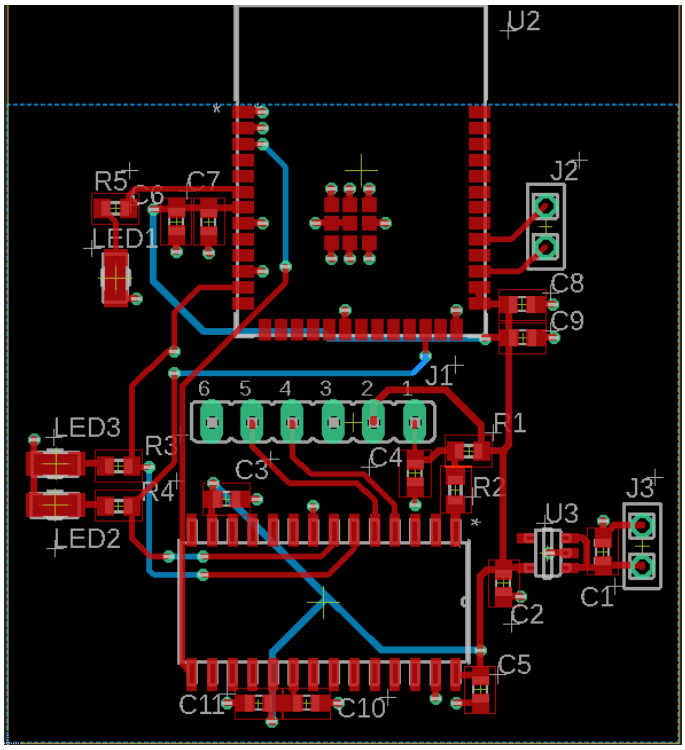

Figura 21- *Board*

4. Pick-and-Place

Geralmente quando se produz uma *board*, a fábrica trata de todo o processo de fabrico e colocação dos componentes. Neste projeto, a fabrica só fez a parte da *board*, *pads* e trilhas. A colocação dos componentes foi realizada no âmbito deste projeto (processo de *pick-and-place).*

Neste processo tendo os componentes dispostos e organizados, como vemos na Figura 23, colocamos o componente desejado no local correto. De seguida, é necessário soldar com o ferro de soldar. Contudo, também existe outra técnica usada neste processo que é colocar uma espécie de molde, estêncil.

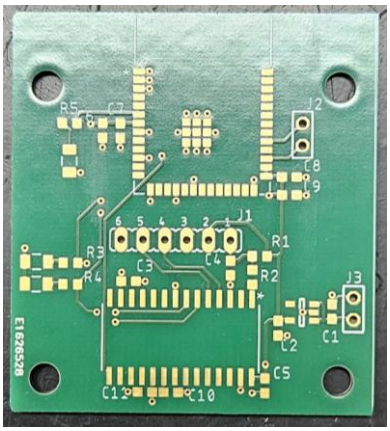

Figura 22- *Board* sem componentes

Assim temos a *board* finalizada, passamos do desenho, como vemos Figura 21, para o mundo real Figura 22. De seguida é feito o *pick-and-place* conforme explicado. Como vemos na Figura 23, os componentes prontos para serem soldados no seu respetivo lugar.

Na Figura 23 observamos que cada componente tem associado um número, este número é chamado *assembly code*, basicamente cada componente, como resistências, condensadores entre outros chips tem um número específico.

Figura 23- Componentes organizados

Na Figura 24, vemos o processo da soldadura à mão, um processo um pouco delicado e de concentração, pois os *pads* são pequenos e não é muito fácil de soldar. Geralmente quando e feita uma soldadura são usadas três coisas essenciais: fluxo, estanho e ferro de soldar. O fluxo serve para manter o calor do ferro o que ajuda a soldar, o estanho, material usados para "prender" o componente à board e por fim o ferro de soldar, que geralmente usa-se numa temperatura de 270 a 280 º C, isto porque o estanho derrete a uma temperatura de cerca de 270 ° C.

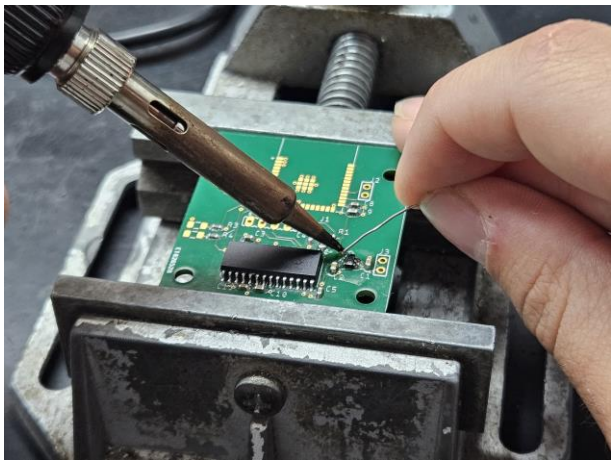

Figura 24- Soldadura dos componentes

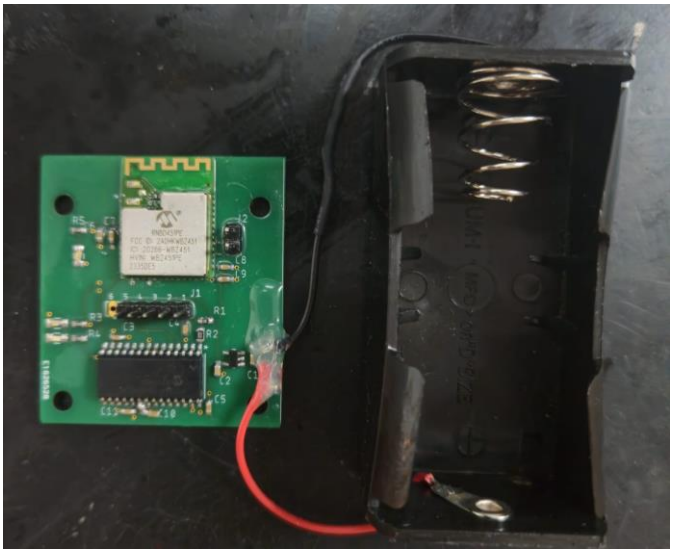

Figura 25- Eletrónica finalizada

Fazendo assim o desfecho da parte da eletrónica, visualizamos que todos os componentes foram soldados e também o suporte da bateria.

A bateria com vemos na figura 26, encaixa perfeitamente no suporte, tendo uma tensão de 3.6V como era expectável, já várias vezes mencionado neste relatório.

Figura 26- Bateria de Alimentação do circuito

5. Aplicação

A aplicação deverá permitir a autenticação com o utilizador e efetivamente usar esta autenticação para se identificar no contentor através do controlo de acesso. A App deverá detetar a presença do contentor e posteriormente permitir o desbloqueio do contentor através de um conjunto de mensagens de autenticação que permitam que o sistema seja robusto e garanta segurança de dados.

A aplicação de *software* funciona como redundância, o que significa, se por algum motivo não for possível destrancar a tampa do contentor via Bluetooth de forma automática, temos como redundância um botão digital para esse mesmo efeito.

Foi feito algo simples, mas funcional, o importante neste caso é que a aplicação consiga conectar ao módulo *Bluetooth* e consiga enviar informação para destrancar a tampa do contentor de resíduos.

O funcionamento da aplicação é o seguinte: quando acedemos à aplicação e ainda não estamos no alcance do modulo *Bluetooth*, ou por algum fator externo não estamos conectados ao *Bluetooth*, o botão digital aparece cinzento, o que indica desconectado.

Assim que há uma conexão da aplicação ao Bluetooth, o botão passa para azul, indicando que está conectado e pronto para ser usado, com isto, o status passa para conectado. Logo a seguir com o propósito desta aplicação temos o clicar do botão digital, este fica verde assim que o clicar, consequentemente os status indicam que foi enviado o comando.

Figura 27- Fluxograma da APP

5.1 Explicação do Código para a APP

Na figura 28, visualizamos parte do código para a realização da aplicação, referi o mais importante para o funcionamento da mesma. No entanto foi configurado a página HTML e consequentemente feito um código de melhoria de estética da página em CSS.

Na parte do *script* feito em JavaScript, foi criado inicialmente três elementos do HTML, o botão, um texto de status e um ícone. Posteriormente, as variáveis *device*, *characteristic* e *service* elas são usadas para armazenar o uuid necessário para o funcionamento.

De seguida a configuração dos UUID (Identificador Único Universal), este identificador é configurável, o que quer dizer que, podemos ativar serviços e desativar serviços dependendo dos bits de ativação.

Temos três UUID, cada um tem a sua função e passo a explicar: o primeiro *DEVICE_UUID* neste caso é apenas a identificação do Bluetooth para qual se vai conectar. *O SERVICE_UUID*, tem como sua função o serviços do Bluetooth que é tentar obter uma conexão, por último *CHARACTERISTIC_UUID*, é a característica do serviço que permite a leitura e a escrita de dados.

A primeira função *connectBluetooth(),* tem como objetivo de se conectar com o *Bluetooth*, adicionei um filtro para que possa conectar a um só *Bluetooth* com o UUID especifico. Assim que se conectar, o botão que estava previamente cinzento, pois não tinha conexão com nenhum *Bluetooth* passa para a cor azul, o que indica que teve uma conexão bem-sucedida com o *Bluetooth*, o *status*, um texto informativo também sofre uma atualização. No fim desta função temos uma mensagem de erro, que combina uma mensagem *Erro ao conectar ao dispositivo,* mais o conteúdo do variável erro, que contém a descrição do problema real que ocorreu.

Por fim, o clique do botão, assim que é clicado envia um bit (0x10) e se for bem-sucedido, a cor do botão e o estado dos *status* mudam. Também há um texto de erros para o caso de tentar carregar no botão e não foi possível enviar o pretendido.

```
<h1>Exemplo de App</h1>
<button id="openButton" disabled>Abrir</button> <!-- Botão de abertura -->
<div id="status">Status: Desconectado <span class="icon-connected"></span></div>

<script>
    const button = document.getElementById('openButton');
    const statusText = document.getElementById('status');
    const iconConnected = document.querySelector('.icon-connected');
    let device;
    let characteristic;

    //ID do dispositivo Bluetooth.
    const DEVICE_UUID = '0000feda-0000-1000-8000-00805f9b34fb-ff01';

    //Representa o serviço que o BLE oferece.
    //Este serviço contém uma ou mais características, neste caso tentativa de conexao
    const SERVICE_UUID = '49535343-fe7d-4ae5-8fa9-9fafd205e455';

    //As características são onde a leitura ou escrita de dados realmente ocorre.
    const CHARACTERISTIC_UUID = '4d434850-22e4-4246-af03-0c4a2f906358';

    // Função para conectar ao dispositivo Bluetooth
    async function connectBluetooth() {
        try {
            device = await navigator.bluetooth.requestDevice({
                filters: [{ services: [SERVICE_UUID] }] //Filtro para conectar ao especifico
            });

            // Conectar ao dispositivo selecionado
            const server = await device.gatt.connect();
            const service = await server.getPrimaryService(SERVICE_UUID);
            characteristic = await service.getCharacteristic(CHARACTERISTIC_UUID);

            // Atualizar interface ao conectar
            button.classList.add('connected');  // Mudar a cor do botão
            button.disabled = false;  // Ativar o botão "Abrir"
            statusText.textContent = `Status: Conectado a ${device.name}`; // Atualiza o text
            statusText.classList.add('connected'); // Alterar a cor do status
            iconConnected.classList.add('connected'); // Mostrar ícone de conexão

        } catch (error) {
            console.error('Erro ao conectar ao dispositivo: ' + error);
            statusText.textContent = 'Erro ao conectar: ' + error;
        }
    }

    // Enviar comando ao clicar no botão "Abrir"
    button.addEventListener('click', async () => {
        if (characteristic) {
            try {
                // Envia um bit para informacao de abrir
                const value = new Uint8Array([0x10]);
                await characteristic.writeValue(value);
                statusText.textContent = 'Comando enviado'; // Atualizar o status
            } catch (error) {
                console.error('Erro ao enviar o comando: ' + error);
                statusText.textContent = 'Erro ao enviar o comando: ' + error;
            }
        } else {
            statusText.textContent = 'Por favor, conecte primeiro.';
        }
    });

    // Conectar ao dispositivo automaticamente ao carregar a página
    window.onload = connectBluetooth;
```

Figura 28- Enxerto do código da aplicação

5.2 Etapas da APP

1° fase, ainda sem conexão a um *bluetooth*, e na tentativa de ser conectar a um.

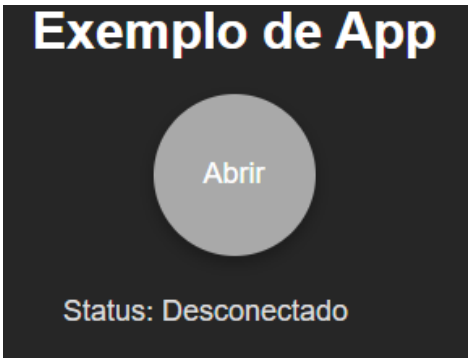

Figura 29- Ble desconectado

2° fase, app está conectada ao *bluetooth* e à espera de ser carregado no botão.

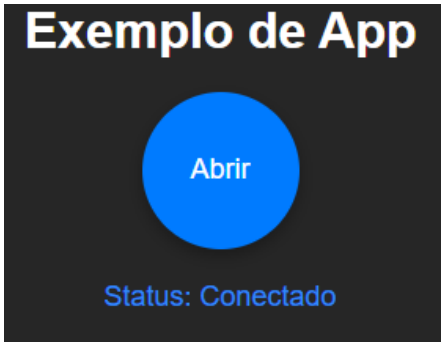

Figura 30- Ble conectado~

3° fase, o botão foi carregado e foi enviado o *bit* com sucesso.

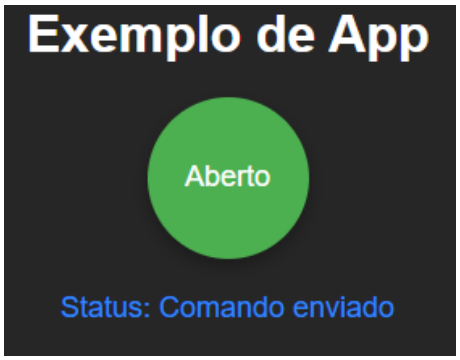

Figura 31- App envia bit

6. Função *main* do µP

Na função *main* onde geralmente é desenvolvido a logica principal do código, começamos por incluir algumas bibliotecas importantes para o código, também incluímos vários #*define*, para os mais variados pinos a usar, este define é simplesmente para não usar o nome específico do pino e sim usar um nome mais comum, por exemplo, #*define* RX_PIN LATBbits.LATB3, neste caso, começa-se sempre pelo um #*define*, de seguida o nome pretendido a usar RX_PIN depois deste o nome original LATBbits.LATB3.

Assim que definimos todos os pinos, declaramos as funções usadas neste código, este tem como sua função de ser mais fácil de orientação do código e ser mais organizado para o utilizador ou outra pessoa que venha escrever no mesmo, logo depois as variáveis globais, usei duas do tipo *char: data*[] e *response*[].

Agora entramos na função *main*, onde tudo começa. Começamos por declarar as variáveis usadas no código, de seguida chamamos uma função *SYSTEM_Initialize*. Esta função chama todas as funções para que o PIC funcione com todos os recursos necessários. Os *defines* previamente declarados são agora chamados e postos num estado *high* "1" ou *low* "0", TX_PIN = 1.

Comecei por garantir que as *strings* usadas comecem todas com caracteres nulos anulando o lixo que estas podem conter, memset(data, '\0', sizeof (data)); este *memset* é uma função da biblioteca padrão do C (<string.h>), onde vai buscar a *string data*, e o seu tamanho, consequentemente altera tudo para caracteres nulos.

Com tudo declarado no PIC e pronto a usar, falta configurar o *bluetooth*. Neste caso comecei por usar uma função chamada *sendCommand*, onde irei explicar mais à frente, basicamente esta função é responsável por enviar e receber comandos da UART.

Ao usar esta função do *sendCommand* com o código "$$$", faz com que o *bluetooth* entre no modo comando ou por outras palavras, num modo privilegiado, onde é possível configurar o modulo do *bluetooth*. De seguida é enviado o comando de "SBI,03". Este comando é usado para alterar *o baud rate* para 115200Bps.

Iniciamos um *loop* de ficar à espera de conexão com outro *bluetooth*, de aguardar e compara o bit enviado pelo botão da app, e por fim fazer a desconexão desse mesmo *bluetooth* e começar novamente a procurar outro.

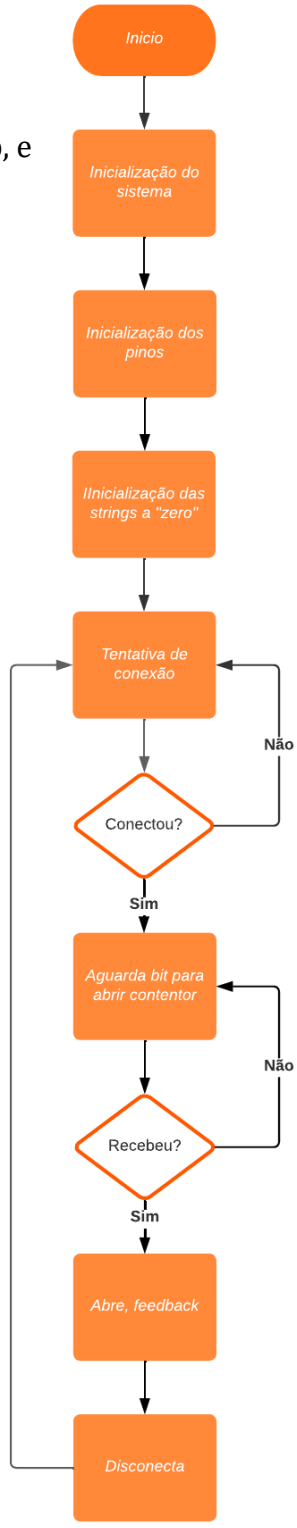

Figura 32- Fluxograma da função *main*

6.1 Função *sendCommand*

A função *sendCommand*, é muito importante pois consegue ler e escrever da UART. Tem como parâmetros de entrada **command*, é um ponteiro do tipo *char* que aponta para uma *string*, que contem o comando a ser enviado para UART. *Response*, um ponteiro do tipo *char* que representa a resposta esperada, por último *timeOut*, é o tempo limite de espera para a resposta (em segundos).

De seguida na declaração de variáveis temos o *buffer*, é um *array* de caracteres usadas para armazenar a resposta, *timeCount*, um contador que monitora o tempo que passou, *bufferPosMem*, um índice que aponta para a posição atual do *buffer*, *receivedBuffer*, variável que armazena cada *byte* da UART.

Um exemplo de como fica a função sendo chamada no *main*:

```
sendCommand("$$$\r", "CMD>", 5);
```

```
while (*command != '\0') {
        UART1_Write(*command);
        command++;
        }
```

De seguida o primeiro *While*, que faz parte do comando *sendCommand*, serve para escrever na UART, quando chamamos a função, cada caracter vai ser enviado para a UART um de cada vez. Daí usamos o **command* que é onde está o primeiro caracter armazenado. Este verifica se é um caracter nulo, e se não for envia esse caracter para a função UART1_Write. Esta função é previamente criada para ser usada quando geramos o código. Quando *command* é incrementado na posição de memoria, será escrito o próximo caracter recebido na UART. Quando **command* for um caracter nulo '\0', este *while* é terminado.

Depois da parte de escrita vem a leitura. Começamos por limpar o *buffer* pondo "0", para que não haja lixo na mesma, memset(buffer, 0, MAX_LENGTH). O tamanho da *string* é definido por MAX_LENGHT, definido nas variáveis globais do projeto.

O *bufferPosiMem* também é colocado a 0.

Temos o segundo *While*, *while* (*timeCount* < *timeOut*), é usado para ficar à espera de ler da UART, o *timeOut* é definido quando esta função e declarada, ou seja, no exemplo que dei, a ultima virgula corresponde ao tempo de espera que a função vai estar à espera, até que o *timeCount* seja maior que o *timeOut*, este *timeCount* é sempre incrementado um valor sempre que percorre este ciclo *While* .O *recivedByte* é onde é armazenado o caracter que foi lido pela a função *Read*(), mais uma vez esta função é previamente gerada quando iniciamos a UART. O seguinte if (*bufferPosiMem* < MAX_LENGTH - 1), verifica se o tamanho do buffer ainda não excedeu o limite máximo previamente declarado, guarda um byte na *string* para o caracter nulo, o byte recebido na variável *receivedByte* é armazenada no buffer, onde é incrementa uma posição na posição de memória da variável.

O código (strstr(buffer, Response) != NULL), compara as duas *string*, buffer e Response e verifica-as, se forem iguais, retorna 1, quer dizer que foi com sucesso.

O próximo código verifica o tempo de espera, onde TMR1, um temporizador do PIC, se um segundo tiver passado, (TMR1 >= 1000), o tempo decorrido *timeCount* é incrementado e o temporizador faz o *reset*.

Com esta verificação acaba a função de ler e escrever na UART

```
unsigned char sendCommand(char *command, char *Response, unsigned int timeOut) {

    char buffer[MAX_LENGTH]; // Buffer para armazenar a resposta
    unsigned int timeCount = 0; //variavel para armezenar a contagem do tempo
    unsigned int bufferPosiMem = 0; // variavel para armazenar a posição na resposta
    char receivedByte; // Variável para armazenar o byte recebido


    while (*command != '\0') {
        UART1_Write(*command);
        command++;
    }

    __delay_ms(200);

    // Limpa o buffer antes de iniciar a leitura
    memset(buffer, 0, MAX_LENGTH);
    bufferPosiMem = 0; // Reseta o índice do buffer

    // Ler a resposta dentro do timeout especificado
    while (timeCount < timeOut) {
        // LE um byte da UART
        receivedByte = Read();

        if (receivedByte == -1) {
            //  printf("TimeOut de leitura da UART\n");
            return -1;
        }

        // Armazena o byte recebido no buffer, se não exceder o tamanho máximo
        if (bufferPosiMem < MAX_LENGTH - 1) { // -1 para deixar espaço para o caracter nulo
            buffer[bufferPosiMem++] = receivedByte;
            buffer[bufferPosiMem] = '\0'; // Adiciona um caracter nulo a  de string
        }

        // Verifica se a resposta esperada foi encontrada
        if (strstr(buffer, Response) != NULL) {

            return 1; // Sucesso ao encontrar a resposta
        }

        timeCount++;
    }

    printf("Erro na leitura da UART");
    return 0;
}
```

Figura 33- Função *sendCommand*

7. Exemplo na prática

Agora que já sabemos o que é pretendido com o projeto, passo a explicar passo a passo do processo da programação até ao fim.

Primeiramente, como é logico, precisamos de alimentar o circuito, pondo a bateria. De seguida, ligamos o nosso programador, neste caso PicKit3 (Figura 33), tendo atenção as ligações, pois se estiverem ao contrário não é possível programar. Geralmente os *PicKit* vem sempre como uma marca como indica na Figura 34, o que indica o pino 1 da programação, na qual corresponde o pino 1 da *board,* como vamos na Figura 36.

Figura 34- PicKit3 + cabo alimentação

Figura 35- Posicionamento ideal

Figura 36- Pino 1 da board

Assim que tivermos o nosso código pronto para ser descarregado para µP, é essencial ter atenção aos erros, e com esta ferramenta "*clean and build*", uma ferramenta muito utiliza para ver se há erros sem precisar recorrer a programação do µP. Na Figura 37, assinalado a verde, vemos qual é a ferramenta em causa. E porquê é que esta ferramenta é muito importante? A ferramenta varre o codigo todo, não só o *main*, onde enventualmente estamos a programar, mas todos os ficherios .c e .h que pertencem ao projeto.

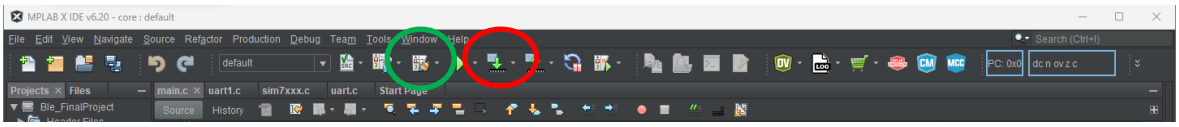

Figura 37- Ferramenta de *clean and build*

Consequentemente ao utilizar esta ferramenta, vamos ter um *feedback* de *Build Successful* no *output* do IDE, com forme vemos na Figura 38

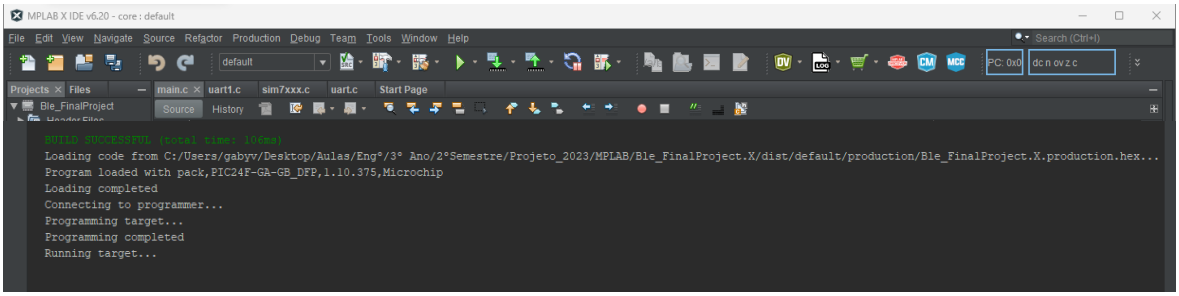

Figura 38- *Build Successful*

Agora como já temos o programa sem erros, programamos o µP. Com isto, utilizamos outra ferramenta, com o nome de *"Make and Program Device Main Project"*, assinalado a vermelho na Figura 37, onde basicamente vai fazer tambem o "*clean and build* " e finalmente programar, mas, antes disso temos que escolher e ferramenta utlizada para programar, como indica na Figura 39.

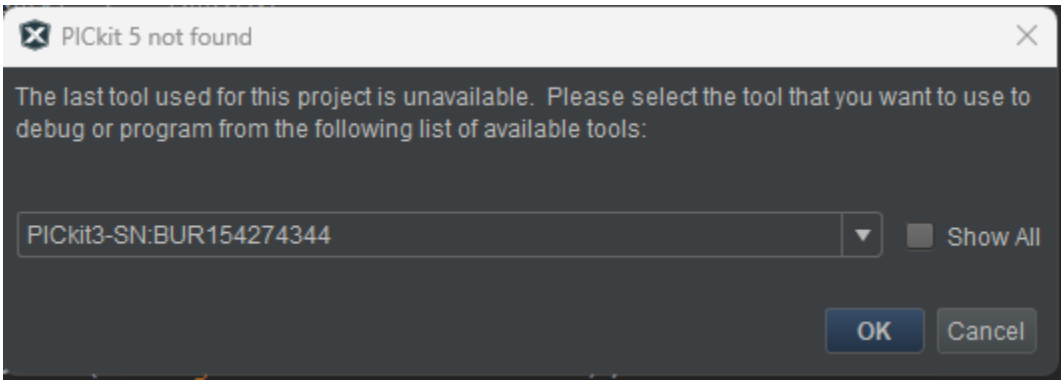

Figura 38- Escolha da ferramenta

O passo final será visualizar as mensagens recebidas vindas do *ouput* do IDE, para ver se foi bem sucedido, conforme vemos nas Figuras 38 e 40.

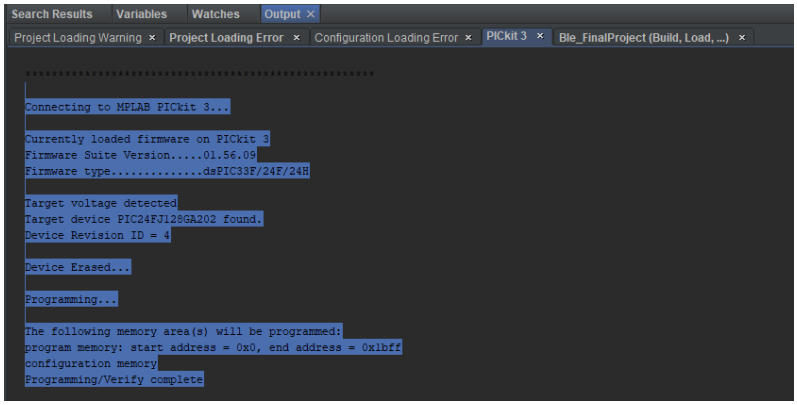

Figura 39- *Programming/Verify complete*

Agora que temos o programa a correr no µP, podemos ver finalmente o projeto a funcionar. E como é possivel ver as mensagem enviadas e recebidas pela UART? Conseguimos ver, com a utilização de um pequeno adaptador chamado CP210, a função deste adaptador é converter o sinal do recebido para TTL onde aí é recebido e processado com ajuda de um programa HTerm. Como todos outros adaptadores temos que ter atenção ás ligações, onde geralmente é utilizado a *Ground* e o RX conforme neste caso.

Figura 40- USB to TTL *Converter* (CP210)

Com ajuda do programa HTerm e devidamente configurado com as especificações do projeto, como *baud rate*, *bit's* de dados, *parity* e stop bits, como vemos na Figura 41.

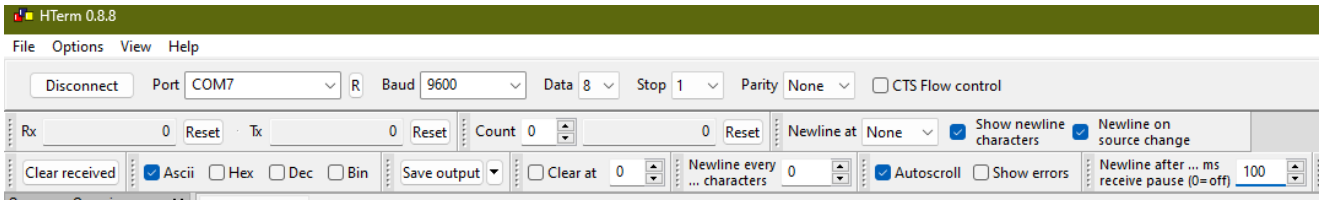

Figura 41- Configurações do Hterm

Neste ponto estamos legíveis de ver as mensagens enviadas e recebidas do µP para o *bluetooth*.

Começamos por ter uma conexão bem sucedia, no HTerm temos a mensagem de sucesso e na App, como já vimos anteriormente, o estado de conectado.

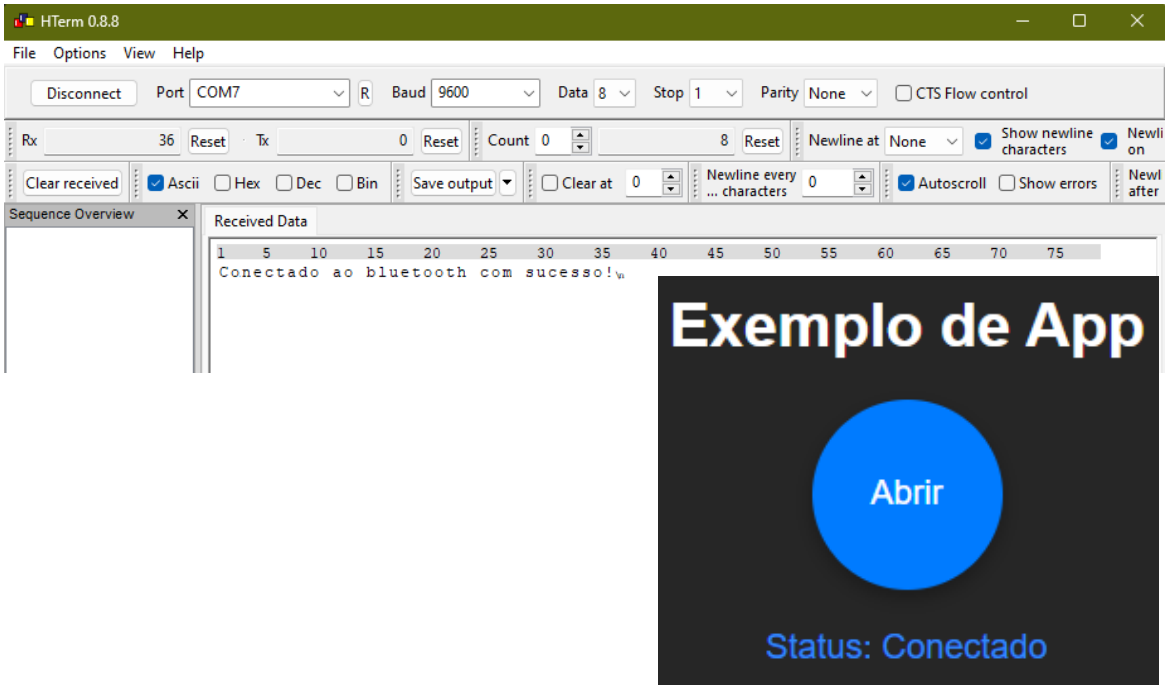

Figura 42- Estado, conectado ao bluetooth

Como já sabemos, o *bluetooth* fica a espera que seja carregado no botão da app, assim que carregamos os seguintes eventos acontecem.

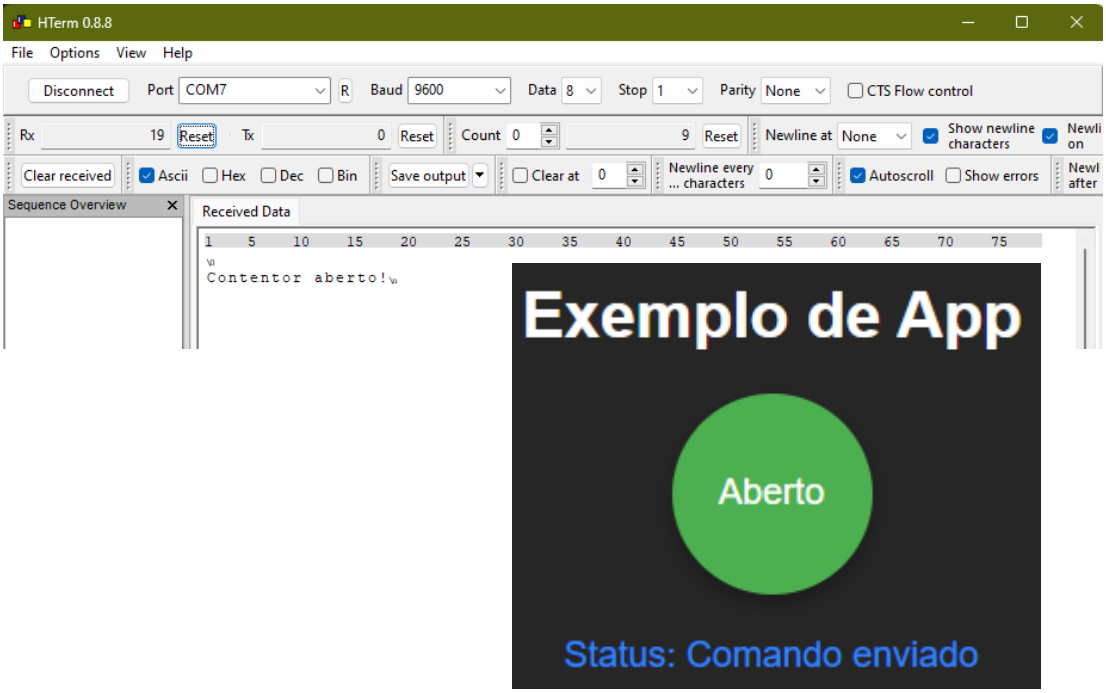

Figura 43- Estado, envio de comando abrir

Por fim, o *bluetooth* vais se desconectar com a app, logo os seguintes eventos vão corresponder ao mesmo.

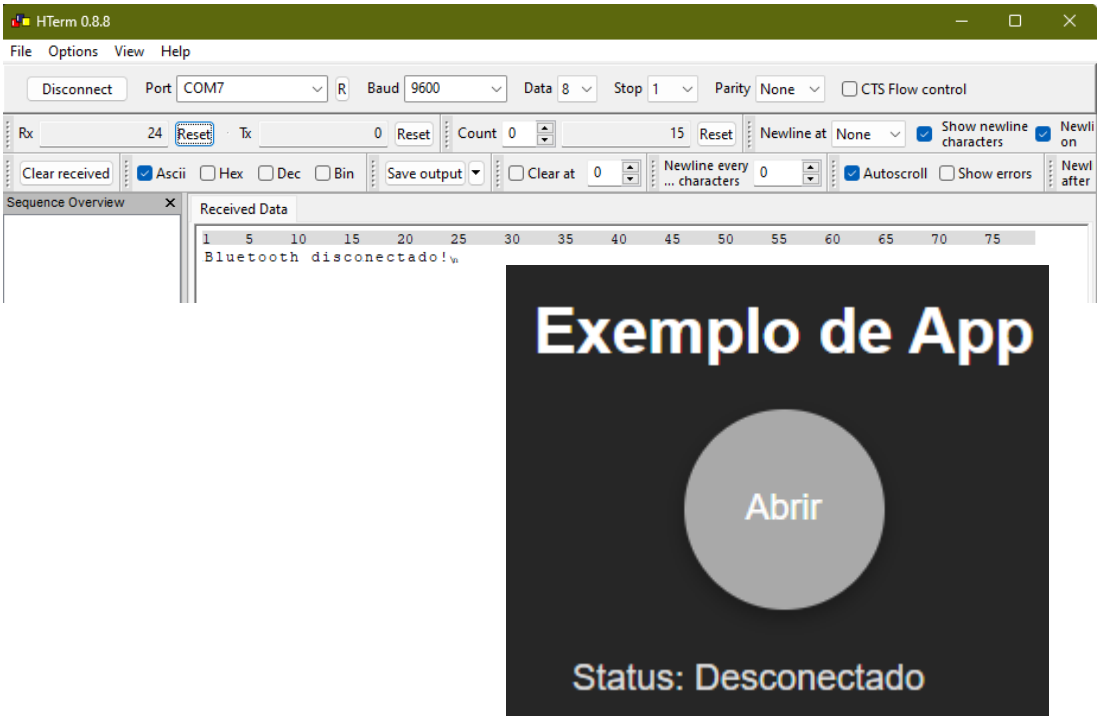

Figura 44- Estado, desconectado

8. Conclusão

A realização do projeto tem como principal objetivo o de acrescentar um método de abertura dos contentores de resíduos, que no futuro será implementado no projeto de controlo de acesso, na Evox.

O projeto teve como base todo o conhecimento adquirido ao longo destes anos, pois ter o conhecimento da eletrónica, programação e radiação de antenas é a base importante do curso. Desta forma, angariei conhecimentos que me permitem ser um profissional de sucesso.

O trabalho efetuado irá ter impacto na vida das pessoas num futuro próximo, pois não é só a facilidade do despejo do resíduo, mas também o meio ambiente vai agradecer. Ter a consciência que precisamos de mudar certos hábitos agora, para que o futuro seja mais risório e que consigamos preservar este mundo ao máximo.

9. Trabalho futuro

Num trabalho futuro, começaria por explorar a segurança da comunicação entre o *bluetooth* do projeto, com a app envolvida do *bluetooth* do dispositivo onde haveria a conexão, onde o fator importante seria por descobrir as vulnerabilidades.

10. Referências

[1]. Microchip Technology Inc. Documentação do PIC24FJ128GA202.

https://www.microchip.com/en-us/product/PIC24FJ128GA202.

Acedido em agosto 2024

[2]. Microchip Technology Inc. Documentação do RNBD451. https://www.microchip.com/en-us/product/RNBD451PE#document-table. Acedido em agosto 2024.

[3]. *Texas Instrument*. Documentacão do TLV755. https://www.digikey.pt/pt/products/detail/texas-instruments/TLV75533PDBVR/9356541

Acedido e agosto 2024