UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Bruno Suguimoto Iwami

Advertising em Bluetooth Low Energy

São Carlos

2018

Bruno Suguimoto Iwami

Advertising em Bluetooth Low Energy

Monografia apresentada ao Curso de Engenharia de Computação, da Escola de Engenharia de São Carlos e Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Carlos Dias Maciel

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Iwami, Bruno Suguimoto
I898a Advertising em bluetooth low energy: ibeacons / Bruno Suguimoto Iwami; orientador Carlos Dias . São Carlos, 2018.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos e Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2018.

1. Advertising. 2. Bluetooth. 3. iBeacon. 4. Low Energy. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Bruno Suguimoto Iwami

Título: "Advertising em Bluetooth Low Energy"

Trabalho de Conclusão de Curso defendido em 28/06/2018.

Comissão Julgadora:	**Resultado:**
Prof. Associado Carlos Dias Maciel (Orientador) - SEL/EESC/USP	Aprovado
Mestre Tadeu Junior Gross Doutorando - SEL/EESC/USP	Aprovado
Mestre Júlio Augusto Druzina Massignan Doutorando - SEL/EESC/USP	Aprovado

Coordenador do Curso Interunidades Engenharia de Computação:

Prof. Dr. Maximiliam Luppe

Este trabalho é dedicado à empresa Korth RFID, como uma contribuição pessoal no desenvolvimento de novas aplicações visando aprimorar a modernização da área agrícola com novas tecnologias.

AGRADECIMENTOS

Agradeço, primeiramente, à minha família, meus pais, Alcino Iwami e Maria Cristina Suguimoto Iwami, e meu irmão, Rodrigo Suguimoto Iwami, pelo apoio e suporte no desenvolvimento deste Trabalho de Conclusão de Curso e pela vida que me proporcionaram de um modo geral, amo muito vocês.

Agradeço ao Danniel Sebastião Dias, pelo suporte técnico e consultoria no desenvolvimento do projeto e pela amizade e companheirismo no decorrer da minha graduação.

Agradeço à empresa *Korth RFID* por me proporcionar a oportunidade de trabalhar no desenvolvimento do projeto e pelo fornecimento de materiais necessários para o mesmo.

Agradeço ao meu Orientador Professor Doutor Carlos Dias Maciel tanto pelo suporte, orientação e auxílio no cronograma de desenvolvimento deste trabalho como no decorrer da minha graduação com seu acompanhamento.

"O sucesso é uma consequência e não um objetivo."
Gustave Flaubert

RESUMO

IWAMI, B. **Advertising em Bluetooth Low Energy**. 2018. 65p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Este trabalho consiste no estudo e implementação da funcionalidade *advertising* de um módulo *Bluetooth*, mais especificamente, no modo *iBeacon* de *advertising*, que é o padrão utilizado pela empresa *Apple*, pioneira no desenvolvimento deste modo de operação. O foco se deu, principalmente, na implementação direta no módulo (*NINA-B112*) da empresa *Ublox*. O módulo vem pré-programado com o *softdevice s132* da empresa *Nordic Semiconductor* a qual disponibiliza uma *SDK* muito versátil para desenvolvimento sendo esta utilizada por várias empresas comercialmente.

O estudo da funcionalidade *advertising* teve como objetivo principal o entendimento de como são formados os pacotes de dados enviados pelo módulo *Bluetooth*, isto é, a ordem pela qual os bytes são disponibilizados e sua interpretação. Além do estudo da formação dos *packets* foi possível também fazer o cálculo de uma estimativa da distância do módulo central (o qual desempenha o papel de ler os pacotes enviados por outros dispositivos, bem como enviá-los) aos periféricos (módulos programados apenas para emitir sinais, sem fazer leitura dos mesmos) através da força do sinal recebido, o *RSSI*.

A aplicação consiste no acionamento de dispositivos de Entrada/Saída, no módulo Central, de acordo com a distância estimada do módulo estritamente emissor (Periférico) mais próximo, sendo este acionamento da forma de alertas visuais ou sonoros.

Além de estudo aprofundado na tecnologia *Bluetooth*, houve, também a implementação de uma interface para interação com o usuário em tempo de aplicação, podendo este fazer alterações em configurações pré-programadas no módulo.

Palavras-chave: Bluetooth. Low Energy. Advertising. Ublox. iBeacon. Nordic Semiconductor. rssi. embarcados. implementação. firmware. nina-b112.

ABSTRACT

IWAMI, B. **iBeacons**. 2018. 65p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

This Final paper consists in the study and implementation of the *advertising* function on a Bluetooth module, more specifically, an iBeacon advertising, which is the standard advertising structure adopted by *Apple inc.* pioneer on the development of this operation mode. The focus was mainly on the direct implementation in the module (NINA-B112) of the company *Ublox*. The module comes with the *softdevice s132* from *Nordic Semiconductor inc.* which provides a very versatile SDK which is used by many companies on commercial purpose for developing their applications.

The study of the *advertising* function had as main objective the understanding of the data packets send by the Bluetooth module, that is, the order in which the bytes are made available and their interpretation. In addition to the study of the packet formation it was also possible to calculate an estimate of the distance from the central module (which plays the role of reading packets sent by other devices, as well as sending them) to the peripherals (modules programmed to only send signals, without reading them) through the strength of the received signal, *RSSI*.

The application consists in the I/O activation, on the Central module, according to the estimate distance from the closest strictly emitter module (Peripheral), this activation can be either visual or audible alert.

Besides the study on Bluetooth Technology, there was also an implementation of an interface for interaction between the user and the device in application time, so that the user can make some modification on preprogrammed settings inside the module itself.

Keywords: Bluetooth. Low Energy. Advertising. Ublox. iBeacon. Nordic Semiconductor. rssi. embedded. implementation. firmware. nina-b112.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ATT	Attribute Protocol
BDM	Background Debug Mode
DFU	Device Firmware Update
EVK	Evaluation Kit
GAP	Generic Access Profile
GATT	Generic Attribute
PWM	Pulse Width Modulation
RGB	Red Green Blue
SDK	Software Development Kit
SWD	Serial Wire Debug
TTL	Transistor Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
UUID	Universally Unique Identifier

SUMÁRIO

1 INTRODUÇÃO

Com o desenvolvimento da nova tecnologia *Bluetooth Low Energy*, foi criado um modo de operação no qual o dispositivo envia, continuamente, de acordo com sua programação, sinais em radiofrequência, chamados de *advertising*. Estes sinais, podem, então, ser interpretados seguindo algumas regras de construção para a estrutura dos *bytes* enviados. Um conjunto específico de regras de interpretação destes dados ficou registrado pela empresa *Apple* e ganhou o nome de *iBeacon*.

Figura 1: Imagem representativa do iBeacon

Fonte: iBeacon (2018)

A partir da interpretação destes pacotes de dados, os dispositivos equipados com um módulo *Bluetooth* podem ler estes sinais enviados e reagir à eles de acordo com sua programação.

Inicialmente, o modo *advertising* tinha como objetivo atingir a indústria mundial de varejo, proporcionando ofertas personalizadas e até mesmo pagamento de produtos nas lojas. Atualmente, estão surgindo, aos poucos, novas aplicações para esta nova tecnologia. Como por exemplo, em mercados, nos locais onde existem produtos em promoção e que merecem destaque, são colocados *beacons* emitindo sinais referentes ao produto, e então, quando um consumidor se aproximar destes sinais e estiver equipado com um dispositivo que possui um aplicativo que faz a leitura dos dados enviados, este será avisado sobre o produto em questão.

Figura 2: Representação do funcionamento do iBeacon

Fonte: iBeacon Application (2018)

Existem outras aplicações para o modo *advertising* fora do âmbito comercial. Este modo de operação pode, também, ser utilizado para se medir a distância entre o emissor do sinal e o receptor, sendo possível aplicações como o chamado *GPS indoor*. Além de outros serviços oferecidos que encapsulam o comportamento de parte do dispositivo, por exemplo o envio contínuo de informações como batimentos cardíacos, taxa de glicose sanguínea, velocidade de corrida, entre outros listados no próprio site da *Bluetooth*.

2 MATERIAIS

Neste capítulo serão explicados os materiais utilizados no desenvolvimento do projeto. Serão divididos em duas subsessões: os materiais físicos e os virtuais.

2.1 Materiais Físicos

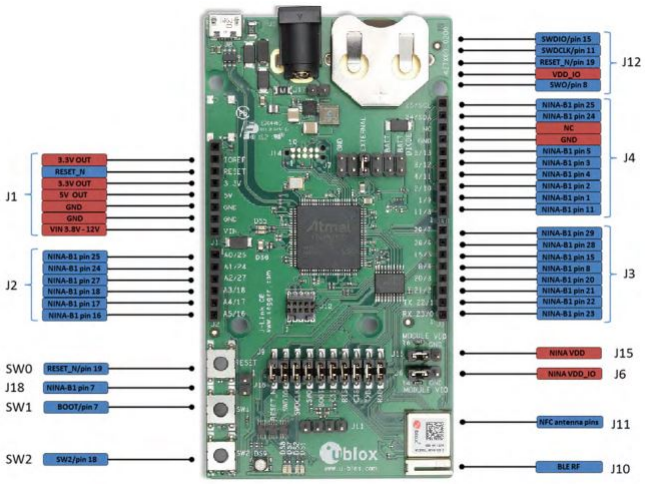
Na implementação física do sistema, inicialmente, foi utilizada a placa de desenvolvimento fornecida pela empresa *Ublox*, a *EVK-NINA-B1*, a qual já possui alguns pinos de entrada e saída necessários para ativação dos alertas sonoros e visuais, *leds* internos para teste, foi acoplado um *buzzer* (sem oscilador interno) e uma sirene alimentada externamente à placa de desenvolvimento. Para a integração de tudo fez-se uso de cabos *jumpers*, *protoboard* e, posteriormente, placas específicas para a aplicação, fornecidas pela empresa *Korth RFID*, a placa do dispositivo Central e Periférico onde foi acoplado o módulo NINA-B112. Além destes componentes utilizados, foram utilizados instrumentos como multímetro, osciloscópio e ferro de solda. As seções seguintes contém o detalhamento mais aprofundado de cada instrumento:

2.1.1 EVK NINA-B1

Na Figura 3 pode-se observar todas as possíveis conexões através de pinos da placa de desenvolvimento produzida e fornecida pela empresa Ublox. Pode-se acessar, através do site da U-Blox[1] o datasheet da mesma, onde é possível verificar os pinos utilizados com o *softdevice s132*. A placa de desenvolvimento já possui um módulo de conversão *TTL* para *USB* que faz, também, a alimentação da mesma. Esta pode também ser feita, alternativamente, via pilha moeda, pino (possui um pino específico para este fim) ou *plug* do tipo *p4* para conexão de uma fonte externa. Sendo possível alimentá-la com uma tensão de 3.8V à 12V. Além disso, vem equipada com um *led RGB* e 3 botões conectados internamente aos pinos do módulo.

[1] Referência para o datasheet da EVK NINA B1(U-BLOX, 2018)

Figura 3: Evaluation Kit Board Nina B1



Fonte: U-Blox (2018)

2.1.2 Buzzer

Buzzer cuja tradução direta para o Português é “buzina”, é um dispositivo que emite som de acordo com a frequência de pulsos que é enviado à ele. No caso, foi utilizado um buzzer sem oscilador interno, ou seja, foi necessária a implementação de um PWM para a emissão sonora. A Figura 4 representa um buzzer similar ao utilizado na aplicação.

Figura 4: Buzzer

Fonte: Buzzer (2018)

2.1.3 Sirene

Foi utilizado uma sirene comum de mercado para emitir o alerta, também ativada somente por um sinal positivo em sua entrada. A Figura 5, representa uma sirene similar à utilizada na aplicação.

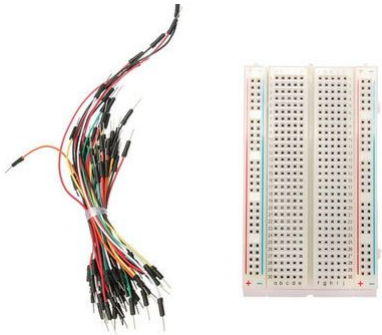
Figura 5: Sirene

Fonte: Sirene (2018)

2.1.4 Protoboard e Jumpers

Instrumentos básicos para qualquer prototipação em eletrônica devido à sua facilidade em fazer conexões com outros dispositivos de Entrada e Saída (*I/O*). A Figura 6, representa um *protoboard* e cabos *jumpers* utilizados em muitos projetos de equipamentos eletrônicos.

Figura 6: *Protoboard* e *jumpers*



Fonte: Protoboard (2018)

2.1.5 Multímetro e Osciloscópio

Instrumentos de medida muito utilizados em projetos eletrônicos, com o multímetro é possível medir a corrente, tensão, resistência e checar curto-circuitos. Com o osciloscópio, é possível visualizar a variação do sinal em função do tempo, podendo variar os parâmetros de medida de diversas maneiras.

2.1.6 *PEmicro USB Multilink*

Para a gravação do *firmware* na placa final do dispositivo, foi utilizado o *USB Multilink* da *PEmicro*, que é uma interface de desenvolvimento que permite o computador ter acesso ao *Background Debug Mode* (*BDM*) e à interface *JTAG*, sendo possível fazer a gravação do *firmware* e também fazer o *debug* do mesmo utilizando os pinos *SWD* para a gravação. A Figura 7, mostra o dispositivo utilizado para a gravação.

Figura 7: *PEmicro Multilink*

Fonte: PE Micro (2018)

2.2 Materiais Virtuais

Na parte virtual do sistema, isto é, os softwares utilizados, destacamos a *IDE Eclipse* (*Eclipse Neon 3 version 4.6.3*[2]), foi necessário fazer algumas configurações para a programação direta no módulo, instalando o *Toolchain* e *Build Tools* necessários para o desenvolvimento. Foi utilizado o *Realterm*[3] e *Putty*[4] para monitoramento e debug do código através da saída Serial e o *PicoScope*, para a visualização de dados no osciloscópio (*PicoScope 2000 series*[5]). Além destes softwares foi utilizado, também, o *nrfUtil*[6] e o *nrfConnect* (aplicativo *mobile*) da *Nordic Semiconductor*, dos quais o primeiro foi utilizado para geração de chaves públicas e privadas para a aplicação do *Bootloader* e o segundo para monitoramento e testes dos dispositivos em modo *advertising*.

[2] Referência para o download da *IDE* (ECLIPSE, 2018)
[3] Referência para o download do *Realterm* (REALTERM, 2018)
[4] Referência para o download do *Putty* (PUTTY, 2018)
[5] Referência para o download do *PicoScope* (PICOSCOPE, 2018)
[6] Referência para o download do *nrfutil* (NRFUTIL, 2018)

3 CONCEITOS TEÓRICOS

Nesta seção serão explicados os conceitos teóricos estudados e utilizados no desenvolvimento do projeto relacionados à tecnologia *Bluetooth*.

3.1 *Generic Access Profile* (*GAP*)

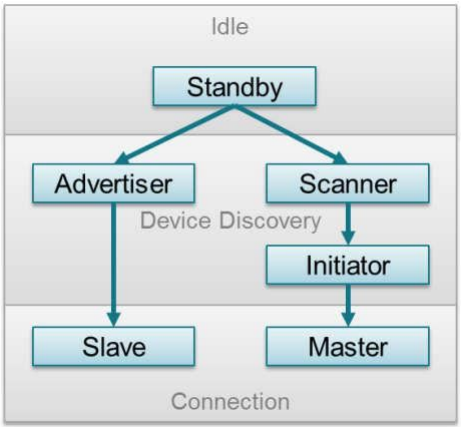
Generic Access Profile, cuja tradução direta para o português é Perfil de Acesso Genérico, é uma camada do protocolo do *Bluetooth Low Energy* responsável pela função de conexão. Ela é responsável pelo controle do modo de acesso e procedimentos do dispositivo, dentre eles, a descoberta de dispositivos, início e término da conexão, inicialização de características de segurança e configuração do dispositivo. Esta camada define vários papéis desempenhados pelos dispositivos, mas os principais são os *Dispositivos Centrais* e *Dispositivos Periféricos*. Outras funcionalidades fundamentais relacionada à conexão são os *advertises* e as conexões, que são o tema principal deste projeto e serão exploradas mais detalhadamente no decorrer deste trabalho.

3.1.1 Papéis Centrais e Periféricos(*Master* e *Slave*)

O Perfil de Acesso Genérico define vários papéis para os dispositivos, entre eles, estão o papel de Central e de Periférico. O papel de dispositivo periférico ou *slave* é desempenhado por pequenos dispositivos que podem se conectar à um dispositivo central mais potente. Estes dispositivos são caracterizados pelo envio constante de informações (*advertising*) e desta maneira, aguardam uma conexão ser iniciada por um dispositivo Central (ou *Master*). Um ponto importante a ser mencionado é que um dispositivo Periférico só pode ser conectado à um dispositivo Central, ao passo que um Central pode se conectar à vários Periféricos. Existem vários exemplos de dispositivos periféricos como monitores de frequência cardíaca, *tags* indicadores de proximidade, monitores de pressão sanguínea, sensores de nível de glicose, dentre inúmeras outras aplicações.

Em contrapartida, os dispositivos Centrais, ou *Masters*, são caracterizados pelo início da conexão, fazendo operações de *scan* de *advertisings* e enviando requisições aos Periféricos. Estes dispositivos, quando desejam se conectar à um Periférico, enviam um pacote com dados de requisição de conexão e, se o Periférico estiver programado para aceitar a conexão, a mesma é estabelecida. A Figura 8 representa a topologia de transmissão de dispositivos Centrais e Periféricos.

Figura 8: Diagrama de Estados do GAP



Fonte: GAP STATE DIAGRAM (2018)

3.2 *Generic Attributes* (*GATT*)

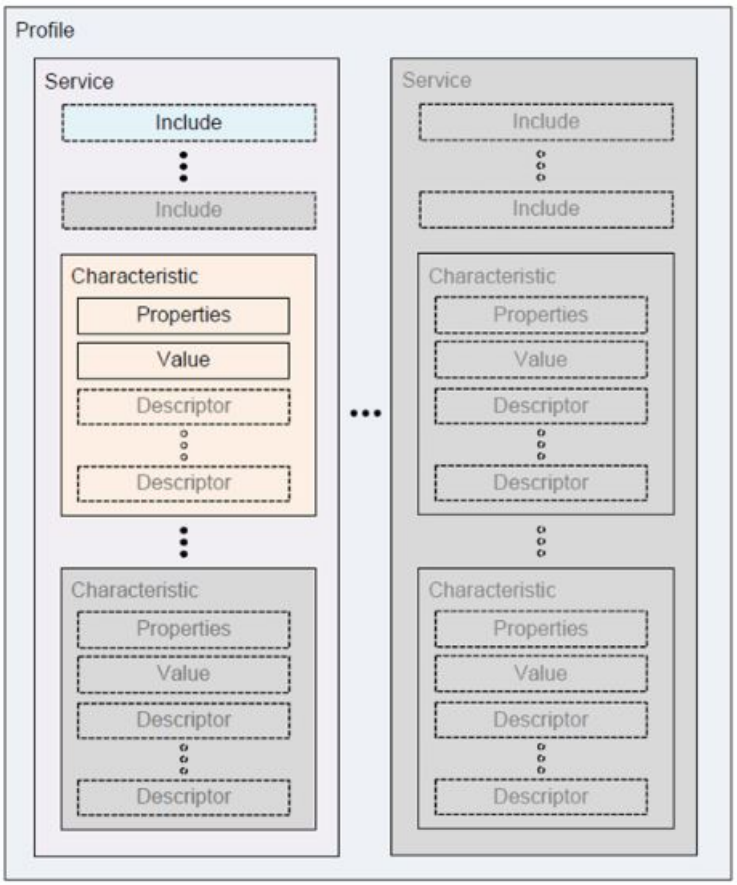
Generic Attributes traduzidos diretamente para o português significa "Atributos Genéricos", os quais são listados no site oficial da tecnologia *Bluetooth*[1] como padrões para comunicação de dados.

3.2.1 Serviços

Um perfil de atributo genérico (*GATT Profile*) descreve funcionalidades, casos de uso e comportamentos gerais deste mesmo tipo de serviço, ou seja, o comportamento de partes de um dispositivo são encapsulados na forma de coletâneas de características e interações com outras partes, estes dados encapsulados são chamados de Serviço[2]. Isto é, um padrão adotado para leitura dos dados de acordo com um determinado tipo de serviço previamente especificado. A Figura 9 mostra a hierarquia de funcionamento de perfis *GATT*. Pode-se encontrar uma lista de exemplos de perfis *GATT* definidos em Bluetooth GATT (2018b).

[1] (BLUETOOTH, 2018)
[2] (BLUETOOTH GATT, 2018a)

Figura 9: Perfil *GATT*



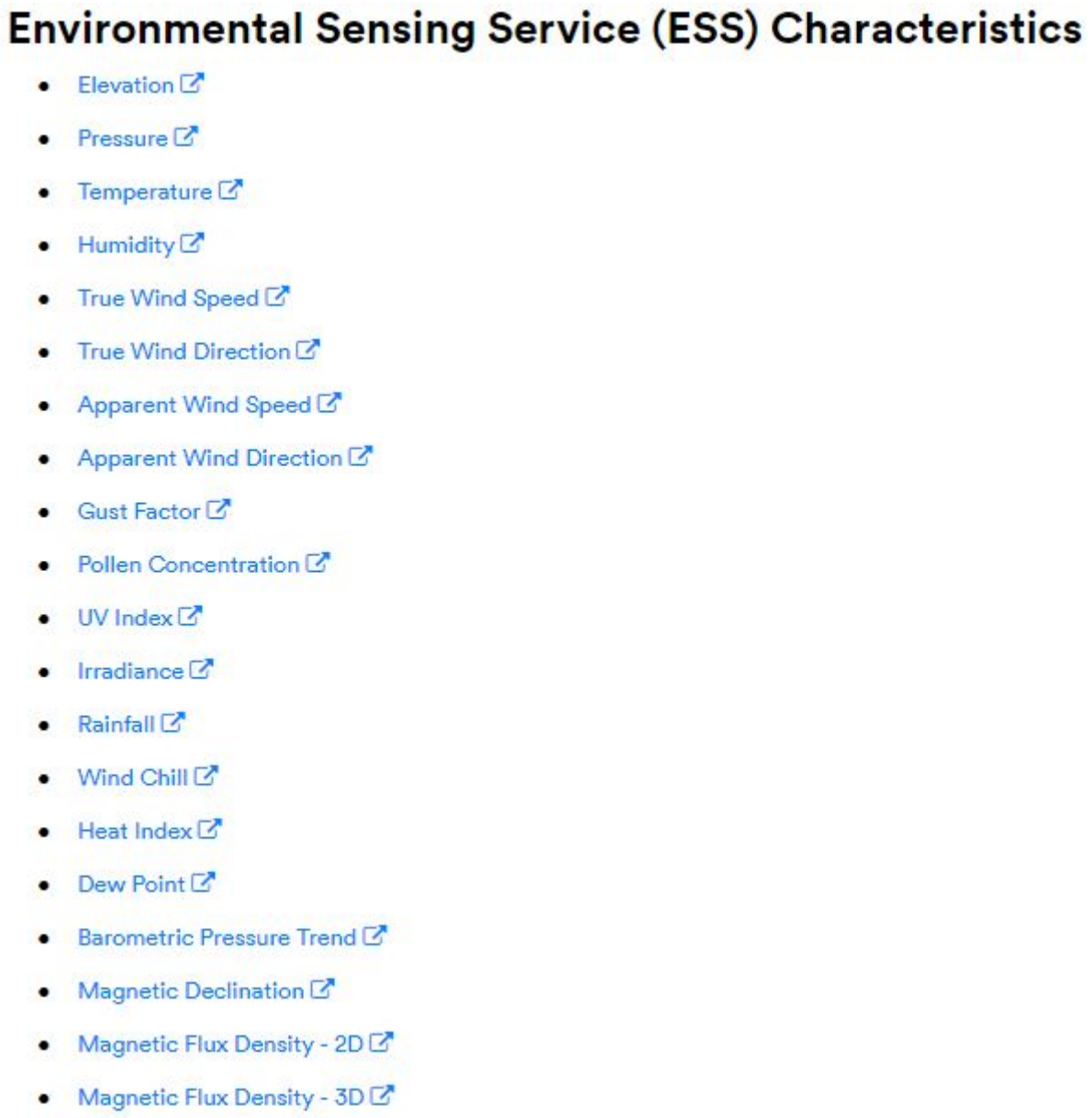
Fonte: Bluetooth GATT (2018a)

Os Atributos Genéricos são especificados em um nível acima do protocolo *ATT* (*Attribute Protocol*). O *ATT* especifica a maneira como o dispositivo fará a leitura e entendimento dos dados genéricos, fazendo então acesso ao hardware, como descoberta de dispositivos, estabelecimento de conexões, envio de dados e escrita de atributos. Já o *GATT* utiliza este protocolo *ATT* para fazer a descoberta de serviços e escrita e leitura de características em outros dispositivos. Pode-se encontrar a lista de serviços *GATT* padronizados oficialmente em Bluetooth GATT (2018c).

3.2.2 Características

Estes Serviços contém características que podem ser lidas ou escritas, relacionados ao serviço à qual pertencem. As características são campos definidos por um *handle*, o tipo do atributo (*UUID*), a permissão do atributo e o seu valor. Na Figura 10, pode-se observar um exemplo das características de um serviço de Sensoriamento Ambiental:

Figura 10: Exemplo de Serviço *GATT*



Fonte: Bluetooth GATT (2018d)

Na Figura 11, observa-se um exemplo de descrição do serviço "*Heart Rate Service*":

Figura 11: Descrição do Serviço de Frequência Cardíaca

Heart Rate Service	Handle	Type of attribute (UUID)	Attribute permission	Attribute value
Service Declaration	0x000C	**Service declaration** Standard UUID$_{service}$ 0x2800	Read Only, No Authentication, No Authorization	Heart Rate Service 0x180D
Characteristic Declaration	0x000D	**Characteristic declaration** Standard UUID$_{characteristic}$ 0x2803	Read Only, No Authentication, No Authorization	Properties (Notify), Value Handle (0x000E), UUID for Heart Rate Measurement characteristic (0x2A37)
Characteristic Value Declaration	0x000E	**Heart Rate Measurement Characteristic** UUID found in the Characteristic declaration value 0x2A37	Higher layer profile or implementation specific	Beats Per Minute E.g. "167"
Descriptor Declaration	0x000F	**Client Characteristic Configuration Descriptor (CCCD)** Standard UUID$_{cccd}$ 0x2902	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.	Notification enabled 0x000X
Characteristic Declaration	0x0010	**Characteristic declaration** Standard UUID$_{characteristic}$ 0x2803	Read Only, No Authentication, No Authorization	Properties (READ), Value Handle (0x0011), UUID for Body Sensor Location (0x2A38)
Characteristic Value Declaration	0x0011	**Body Sensor Location** UUID found in the Characteristic declaration value 0x2A38	Higher layer profile or implementation specific	Sensor Location (8-bit integer) E.g. 3 equals "Finger"

Fonte: Heart Rate Service (2018)

Da Figura 11, pode-se observar os atributos disponibilizados na primeira coluna, seguidos pelo seu *handle*, tipo, permissão e valor. Adiante será explicado cada um desses parâmetros:

a) O *handle* do atributo é um número de 16 *bits* que varia de acordo com o número de atributos presentes e faz a identificação dos mesmos no servidor, permitindo que o cliente faça referência à ele em requisições de leitura e escrita. Este valor nem sempre é sequencial e é muito utilizado pelo *softdevice* para fazer referência à vários atributos. Na perspectiva do programador é uma maneira eficiente de passar valores e informações entre funções além facilitar o controle da aplicação sobre o atributo e obter qualquer informação necessária.

b) O *Type of Attribute*, ou, traduzido diretamente para o português, o tipo do atributo é um *UUID* (*Universally Unique Identifier*), ou seja, um valor de 16 ou 128 *bits* utilizado para identificar a especificação de cada atributo.

c) *Attribute Permissions* são as permissões que os atributos possuem, elas definem as regras de interação com cada um deles, ou seja, quando um atributo pode ser lido e/ou escrito e qual tipo de autorização é necessária para fazer estas operações

d) O *Attribute Value* (valor do atributo) é a informação em si e pode ser qualquer tipo de dado, como o valor de frequência cardíaca dado em batimentos por minuto, o estado de um interruptor de luz, ou até mesmo uma frase qualquer. A partir do exemplo da Figura 11, no atributo "*Service Declaration*" (*handle* 0x000C), seu valor contém o *UUID* do serviço ("*Heart Rate Service 0x180D*") para identificar qual serviço ele se refere. No atributo "*Characteristic Declaration*" (*handle 0x000D*), seu valor é dado pelo conjunto das propriedades (*notify*), o *handle* do atributo (no caso, *0x000E*) e o *UUID* da característica *Heart Rate Measurement* (0x2A37). Finalmente, no campo do valor do atributo *Characteristic Value Declaration* encontramos a medida de batimentos por minuto.

A Figura 12 ilustra as possibilidades para o campo de propriedades das características:

Figura 12: Propriedades da característica de um dado serviço

Properties	Value (Bit field)	Description (From BLE Core Specification v4.2)
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value
Write without response	0x04	If set, permit writes of the Characteristic Value without response
Write	0x08	If set, permits writes of the Characteristic Value with response
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgement. If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgement. If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value.
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor. If set, the Characteristic Extended Properties Descriptor shall exist.

Fonte: Heart Rate Service (2018)

São definidos também papéis de Cliente e Servidor. Os procedimentos *GATT* são divididos em 3 tipos básicos: Descoberta de Procedimentos, Procedimentos inicializados

pelo Cliente e Procedimentos inicializados pelo Servidor[3]. O papel de Servidor *GATT* consiste em armazenar dados transportados por *ATT* e aceitar requisições, comandos e confirmações *ATT* enviados pelo Cliente, então envia respostas às requisições bem como indicações e notificações assincronamente ao Cliente de acordo com a ocorrência de determinados eventos.

3.2.3 Papéis Cliente e Servidor (*Client* e *Server*)

Clientes são os dispositivos que acessam remotamente os recursos fornecidos pelo Servidor *GATT* . Realizam operações como leitura, escrita, notificação ou indicação. Normalmente, Clientes *GATT* são também Centrais/*Masters GAP*. Se o cliente precisa enviar dados para o Servidor, ele utiliza funções de escrita e se precisa receber dados do servidor utiliza funções de leitura.

Servidores são os dispositivos que armazenam dados localmente e fornecem métodos de acesso remoto aos Clientes *GATT*. O papel desempenhado pelos Servidores *GATT* normalmente é o de Periférico/*Slave GAP*. Se o servidor precisa enviar dados para um Cliente, sem o mesmo fazer uma requisição, utiliza-se funções de notificação e indicação (no entanto, o Cliente deve fazer a inscrição nestes parâmetros antes que qualquer dado seja enviado).

3.3 *Advertising*

Advertising é uma das funções do *BLE*, que consiste num meio de transmissão de informações definindo as "intenções" do dispositivo. O protocolo *Bluetooth* define um formato específico de formação dos pacotes tanto para *Advertising* como para transferência de dados. Na perspectiva dos desenvolvedores de aplicações, geralmente, os dados mais importantes desses pacotes são os *Advertising Payload*, ou seja, um campo de 0 à 31 *bytes* que podem ser customizados de acordo com a aplicação. A estrutura de formação destes pacotes de dados segue o seguinte formato:

O 1º *Byte* determina o *Length* (ou tamanho), em *bytes*, do pacote de dados a ser transmitido sem levar em consideração o próprio *byte*.

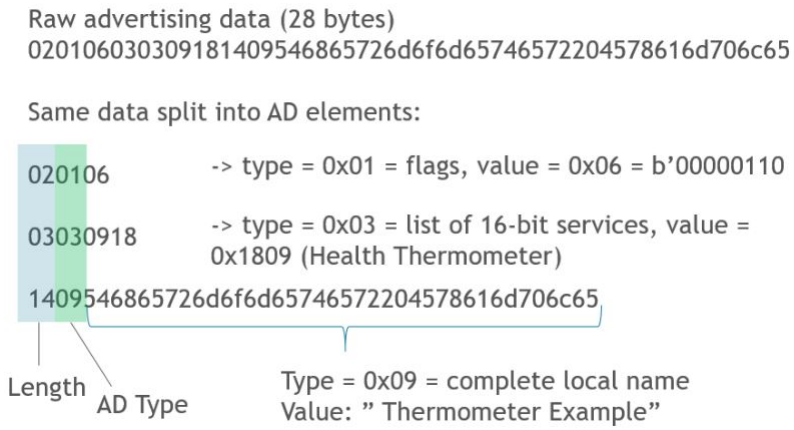
O 2º *Byte* determina o tipo do *Advertising* (*AD type*), responsável por informar o tipo de dado será transmitido. Pode-se acessar a lista completa de tipos de *AD type* em BLE Adv type (2018)

No entanto, os tipos mais usuais são *0x01* (*flags*), *0x03* (*Complete List of 16-bit Service Class UUIDs*), *0x09* (*Complete Local Name*), *0x08* (*Shortened Local Name*) e *0xFF* (*Manufacturer Specific Data*).

[3] (BLUETOOTH GATT, 2018a)

Por fim, após os *bytes* de tamanho e tipo do *advertising*, encontram-se os *bytes* do dado em si. Na Figura 13, pode-se observar um exemplo de *advertising payload*:

Figura 13: Exemplo de *Advertising Packet*



Fonte: Silabs GAP Advertising (2018)

Neste exemplo, o *payload* é dividido em 3 elementos de *advertising*, sendo que o primeiro *byte* é sempre o indicador de tamanho, sendo possível identificar os limites de cada elemento do *advertising*. O primeiro elemento é do tipo *flags*, o *payload* é de um *byte* somente e é possível escrever 8 *flags* neste campo. No caso do exemplo, duas *flags* são marcadas:

a) *Bit* 1: *LE General Discoverable Mode*

b) *Bit* 2: *BR/EDR Not Supported*

O segundo elemento inclui uma lista de serviços adotados (16-*bit UUID*). No caso do exemplo, somente um serviço é transmitido (*Health Thermometer*, *UUID 0x1809*[4]). Vale notar que valores que necessitam de mais de um *byte* são escritos em ordem reversa, pois os pacotes do *BLE* são padronizados em ordem *little-endian*. Por último, o terceiro elemento consiste no nome do dispositivo, contendo o *advertising type Complete Local Name 0x09*. O nome do dispositivo é escrito em hexadecimal e esse nome será mostrado caso algum outro dispositivo realize um *scan* próximo à ele.

Além do *packet* de dados provenientes deste modo de operação, outras informações são atreladas ao processo de *advertising* no momento de sua inicialização como o seu intervalo de operação, tempo de encerramento, *scan response* e *tx power*. Que serão abordados nas próximas seções deste trabalho.

[4] Lista de Serviços GATT disponibilizados em (BLUETOOTH GATT, 2018c)

3.3.1 *Advertising Interval*

O *Advertising Interval* é o parâmetro que define a frequência, ou seja, o tempo entre um envio de pacote e outro, este parâmetro pode variar de 30 milissegundos à 6 segundos que pode ser variado de acordo com a aplicação e o desejado consumo de energia.

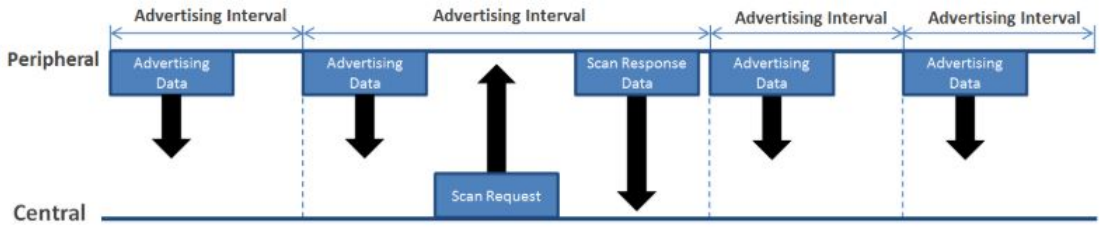
3.3.2 *Advertising Timeout*

Advertising Timeout define o tempo em que o dispositivo faz *advertising* e após expirado este tempo o mesmo encerra sua execução. O mais comum é deixar este parâmetro em zero, desta forma ele nunca cessa a operação de *advertising*.

3.3.3 *Scan Response*

Scan Response é semelhante ao pacote de *advertising*, é utilizado caso seja necessário a transmissão de um maior número de *bytes*. No momento em que um dispositivo se conecta à outro que está em modo de *advertising* é enviado um sinal de *Scan Response Request*, ou seja, o dispositivo que iniciou a conexão "pergunta" para o outro se o mesmo possui um pacote de *Scan Response* em caso afirmativo, este pacote é interpretado da mesma maneira que o *advertising packet* lido, possuindo o mesmo tamanho e formato. A Figura 14 exemplifica o funcionamento o *Advertising Interval* em conjunto com o *Scan Response*:

Figura 14: Estrutura de envio de pacotes *Advertising* e *Scan Response*



Fonte: GAP ADV SR (2018)

3.3.4 *Tx Power*

Este parâmetro define a força com que o sinal será transmitido dispositivo *Bluetooth*, sendo possível, desta maneira, variar a distância com que o sinal é recebido. Quanto maior o *Tx Power*, maior a distância que o sinal conseguirá ser transmitido. Os valores de *Tx Power* são fixos, sendo possível variar entre: -30, -20, -16, -12, -8, -4, 0 e 4. A Figura 15 mostra os valores de *Tx Power* possíveis, bem como a força do sinal esperada à um metro de distância.

Figura 15: Estimativa de distância alcançada por nível de *Tx Power*

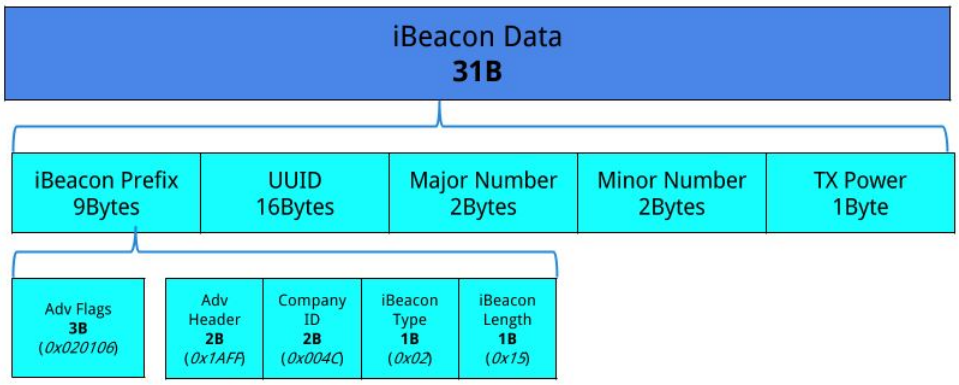
Hexadecimal value	TX Power level	Decimal value	RSSI @ 1 meter	Range (meters)
e2	0	-30 dBm	-115 dBm	2
ec	1	-20 dBm	-84 dBm	4
f0	2	-16 dBm	-81 dBm	10
f4	**3**	**-12 dBm**	**-77 dBm**	**20**
f8	4	-8 dBm	-72 dBm	30
fc	5	-4 dBm	-69 dBm	40
00	6	0 dBm	-65 dBm	60
04	7	4 dBm	-59 dBm	70

Fonte: GAP ADV SR (2018)

3.3.5 *iBeacon*

iBeacon é o nome dado pelo padrão adotado pela empresa *Apple* na criação do formato do pacote de *advertising*. Seguindo o padrão de construção dos *advertising packet*, o modo de operação *iBeacon* possui por definição o seguinte formato de pacote de dados, representado na Figura 16:

Figura 16: Estrutura de dados do pacote *iBeacon*



Fonte: Arm BLE Beacons (2018)

Do pacote representado pela Figura 16, o *iBeacon Prefix* definido pelos *bytes* *0x0201061AFF004C02015* pode ser interpretado da seguinte maneira:

a) *0x02*: tamanho do primeiro elemento do *advertising* equivalente à 2 *bytes*.

b) *0x01*: tipo do primeiro elemento, equivalente à *Flags*.

c) *0x06*: *flags* do tipo *LE General Discoverable Mode* e *BR/EDR not supported*.

Inicia-se, então, a descrição de outro elemento, o *iBeacon advertising* em si, definidos pelos *bytes*:

d) *0x1A*: define o tamanho do *advertising*, ou seja, 26 *bytes* fixos para o *advertising* do tipo *iBeacon*.

e) *0xFF*: define o tipo do elemento, no caso *Manufacturer Specific Data*.

Após a definição do tipo *Manufacturer Specific Data*, o padrão *iBeacon* adota os próximos *bytes* como sendo os seguintes:

f) *0x004C*: identificador da empresa, no caso *0x004C* é o identificador da *Apple*.

g) *0x02*: identificador do tipo de *advertising iBeacon*.

h) *0x15*: tamanho do pacote de dados *iBeacon*.

Em seguida, temos os seguintes campos definidos pelo *iBeacon*, são o *UUID*, *Major*, *Minor* e *Tx Power* (*RSSI* medido à um metro de distância)[5].

3.3.6 *UUID*

Este campo é destinado à uma *string* de 16 *Bytes* usada para diferenciar um grupo de *beacons* relacionados ou para a transmissão de qualquer tipo de dado dentro deste tamanho. Um exemplo usual deste campo seria: se uma marca genérica mantivesse uma rede de *beacons* em uma cadeia de lojas, todos os *beacons* dessa marca utilizariam o mesmo *UUID*. Isto permitiria que um aplicativo dedicado soubesse qual *advertising* é proveniente do beacon desta marca.

3.3.7 *Major*

O campo *Major* é preenchido por uma *string* de 2 *Bytes* utilizado, normalmente, para a distinção de um subgrupo de *iBeacons*. Tomando o exemplo anterior, se uma loja possuísse 4 produtos da marca genérica citada anteriormente, todas elas apresentariam o mesmo *Major*. Desta forma, a marca teria controle de qual loja, exatamente, o consumidor está.

3.3.8 *Minor*

O campo *Minor*, semelhante ao *Major*, é também um campo de uma *string* de 2 *Bytes* utilizado para distinção individual de cada *beacon*. Baseando-se no exemplo anterior, um *beacon* localizado na frente da loja teria seu *Minor* único, permitindo, então, que o aplicativo dedicado da marca, saiba exatamente a localização do cliente na loja.

[5] Referência em iBeacon guide (2018)

3.3.9 *RSSI at one Meter*

Finalmente, o último campo do *advertising packet* é o *RSSI at one meter*, comum e erroneamente chamada do *Tx Power*, que é utilizado para determinar a proximidade do *iBeacon*, este valor define a força do sinal calibrada à exatamente 1 metro de distância do dispositivo, através do mesmo, é possível estimar uma distância aproximada do dispositivo. Este valor é definido em *hexadecimal* e é um complemento de 2.

3.4 *Connection*

O módulo *BLE* pode operar de duas maneiras distintas: modo *scanning* ou *advertising*. O modo *scanning* nada mais é do que a leitura de *advertising* emitidos por outros dispositivos, este procedimento é chamado de *Device Discovery* (Descoberta de Dispositivos). Quando está em modo *scanning*, este pode ser um *scan* do tipo passivo ou ativo. A principal diferença entre eles é que o *scan* ativo pode enviar uma requisição de informação adicional para o *advertiser*, o, mencionado anteriormente, *Scan Response Request*, já o *scan* passivo somente faz a leitura dos *advertising packets*.

Para os dispositivos operarem no modo de *scan* é necessária a configuração de alguns parâmetros como *scan interval* (intervalo de conexão), que define a quantidade de tempo na qual os dispositivos enviam e recebem dados na camada de conexão gerenciados pelo protocolo *Bluetooth Low Energy Stack*, este evento de conexão pode variar de 10 milissegundos a 10 segundos); *scan window* (janela de conexão), representa a duração da permanência na camada de conexão realizando a operação de *scan* e por fim *scan duration* (duração do scan) definindo o tempo em que o dispositivo permanece no estado de *scan*, podendo variar de 10 milissegundos à 10 segundos.

4 DESENVOLVIMENTO

No o início do desenvolvimento, primeiramente, foi feito o experimento de envio de comandos *AT* utilizando a placa de desenvolvimento para a familiarização do protocolo *Bluetooth Low Energy*, utilizando o *firmware* previamente instalado no módulo *NINA B112* e o *software mobile* da *Nordic*, o *nRF Connect* (disponível na *Play Store*). Este *software* faz o *scan* de *advertises Bluetooth* e permite enviar uma requisição de conexão e conectar-se à ele caso o dispositivo esteja configurado para aceitar conexão.

Após a familiarização com o protocolo, deu-se início ao desenvolvimento do *firmware* a ser gravado no módulo, utilizando a *IDE Eclipse* para a confecção do código. Foram feitas configurações para utilização da mesma para compilar os Exemplos fornecidos pela *SDK da Nordic*, como a aquisição de pacotes específicos de *Build Tools* e *Toolchain* necessários, bem como pequenas alterações no *Preprocessor build* para reconhecimento de símbolos utilizados.

Finalizadas as configurações iniciais para início da confecção do código, foi compilado e gravado o Exemplo *ble_app_uart* contido na *SDK* dentro do diretório *SDK/examples/ble_peripheral/*, que consiste em uma aplicação simples de *Bluetooth Low Energy advertising*, permissão de conexão e serviço *NUS* (*Nordic Uart Service*) desenvolvido pela *Nordic*, uma comunicação seguindo o protocolo *UART* (*Universal Asynchronous Receiver Transmitter*) em uma conexão *Bluetooth Low Energy*. A partir deste exemplo foram desenvolvidos os códigos dos dispositivos Periféricos e Central, que serão detalhados a seguir.

4.1 Dispositivo Periférico

Para o dispositivo Periférico foi utilizado uma placa contendo um botão para ligar e desligar o módulo, *leds* indicativos de funcionamento e um botão para carregamento de configurações específicas, no caso, foi implementado um conjunto de características que definem o módulo como modo de configuração ou modo de operação normal.

Este dispositivo consiste basicamente na implementação do modo *Advertising* e criação do Serviço *Nordic UART* com características de escrita e leitura (*Rx* e *Tx* do *UART*). A seguir será explicado as implementações realizadas:

4.1.1 Configurações de *Advertising*

Para iniciar o modo de *advertising* é realizada uma chamada a função *advertising_init()* a qual carrega informações referentes ao *advertising* definidas por constantes e então inicia esse procedimento através da chamada *ble_advertising_init()* que pode ser acessada em

SDK (2018, BLE Libraries/Advertising Module/Functions).

Na função de carregamento de dados, foram feitas alterações nos valores de *UUID* do *advertising* para que este possuísse o formato padronizado pelos *iBeacons*, presentes na Figura 16. Nesta mesma função, são feitas configurações referentes ao *Scan Response*, o qual foi definido de maneira a transmitir um dado do tipo *Complete Local Name*, ou seja, o *advertising packet* contém dados referentes à aplicação e o *scan response* contém o nome do dispositivo. Foram utilizados valores aleatórios no campo de *UUID* do *packet* somente obedecendo o formato de *UUID* do *iBeacon*.

A aplicação desenvolvida utiliza principalmente os valores dos campos *Major* e *RSSI at one meter* do *advertising packet*. O *Tx Power* foi definido com o valor *0xC7*, que em complemento de 2 pode ser interpretado como -56dBm. O valor do *Major*, em modo de operação normal é configurado com *0x07C7* e o nome do dispositivo é *CME_PER*, no entanto, no modo de configuração, o valor do *Major* passa a ser *0x4B54* e o nome do dispositivo é alterado para *CME_CFG*.

Foram feitas alterações nos parâmetros de *advertising interval*, definição de um intervalo de 200 milissegundos, e *advertising timeout*, que foi configurado para não cessar o modo *advertising*. Foi utilizada a *flag BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE* para informar o dispositivo que só pode se conectar a dispositivos do tipo *Bluetooth Low Energy* e que qualquer dispositivo pode se conectar a ele.

Foi utilizada a flag *BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE* para informar o dispositivo que só pode se conectar a dispositivos do tipo Bluetooth Low Energy e que qualquer dispositivo pode se conectar a ele.

4.1.2 *Flash Memory*

Outra função desenvolvida foi a gravação de dados na memória interna do módulo, como este possui um espaço de até 512 *Kbytes* para programação, pode-se gravar a aplicação, o *softdevice*, o *bootloader* e reservar um espaço para a gravação de dados.

Para isto foi utilizada a biblioteca *nrf_storage.h* fornecida pela *SDK* da *Nordic Semiconductor*. Para a utilização dos métodos fornecidos pela biblioteca, é necessário, primeiramente, criar uma instância para gerenciar o acesso à memória através da chamada *NRF_FSTORAGE_DEF* ilustrada em SDK (2018, SDK Common Libraries/Flash Storage/Macros).

Após criada a instância, deve-se inicializar a biblioteca através da chamada *nrf_fstorage_init* que pode ser acessada em SDK (2018, SDK Common Libraries/Flash Storage/Functions).

Após todas as inicializações, pode-se utilizar os métodos como *nrf_fstorage_write()*, *nrf_fstorage_erase()* e *nrf_fstorage_read()*, onde seus parâmetros, em geral, são a instância

inicializada e o endereço de memória a ser acessado, como pode ser observado em SDK (2018, SDK Common Libraries/Flash Storage/Functions)

Para que não houvesse sobrescrita de endereços de memória, foi necessário fazer algumas modificações no arquivo *.ld*, que contém um *Linker Script* para gerenciamento de endereçamento de memória, e fazer a alteração na inicialização da biblioteca de *storage* do módulo, onde se delimita os espaços de memória disponíveis para acesso e gravação de dados pelas funções da mesma.

Utilizando a memória *flash* interna para armazenamento de dados, pode-se armazenar configurações referentes ao modo de operação e carregá-las ao inicializar a aplicações. As informações armazenadas na memória não-volátil interna do módulo foram as seguintes:

a) *Factory Default*: *Flag* para indicar se o módulo carregaria configurações no modo padrão de fábrica, ou seja, configurações pré-estabelecidas, ou se carregaria informações a partir da memória *flash*. Foi necessária esta implementação pois, inicialmente, o módulo não possui nada em seu espaço de memória reservado para gravação de dados, portanto, na primeira execução da aplicação os dados são gravados após a inicialização de todos os componentes virtuais.

b) *Reset Counter*: um contador que incrementa a cada inicialização do módulo. Durante o desenvolvimento fez-se necessário para se verificar se o módulo estava operando constantemente ou se estava tendo algum erro e re-iniciando, sendo a re-inicialização causada por *software* ou *hardware*.

c) *Major*: valor do campo *Major* do modo *advertising iBeacon*.

d) *Minor*: valor do campo *Minor* do modo *advertising iBeacon*.

e) Advertising Timeout: valor que define o tempo em que o dispositivo opera em modo advertising.

f) Advertising Interval: valor que define o intervalo entre cada envio de advertising.

g) *Tx Power*: valor que define a força com que o sinal será enviado, carregado antes do início das configurações do modo *Advertising*.

h) *Mode*: *flag* para definir se o dispositivo está operando como dispositivo central ou dispositivo periférico.

i) *Device name*: nome pelo qual o dispositivo será reconhecido ao ser encontrado por um dispositivo realizando *scan*.

Vale salientar que as operações de acesso à memória demandam tempo e custo de operação, devido à este motivo o desenvolvimento foi realizado salvando informações em um *buffer* estático e este *buffer* enviado como parâmetro às funções de escrita em memória não-volátil. Portanto, foram implementadas funções de manipulação do *buffer*, como escrita, leitura, re-inicialização e apagamento de campos.

4.1.3 *Timer*

Outra função disponibilizada pela biblioteca da *Nordic Semiconductor SDK* é o *Timer*. Utilizado como um marcador de tempo, para inicializá-lo existem alguns parâmetros que devem ser passados, configurando, desta maneira, o estilo do *Timer* a ser criado para que se adapte à aplicação. Esta ferramenta possui dois modos de operação, definidos pela enum do tipo *app_timer_mode_t*, sendo eles definidos por *APP_TIMER_MODE_REPEATED* ou *APP_TIMER_MODE_SINGLE_SHOT*: a primeira define a expiração contínua do *timer* após contado o tempo marcado, a segunda define uma expiração única do *timer*, após expirado, ele é cancelado e destruído, como pode-se verificar em SDK (2018, SDK Common Libraries/Application Timer/Enumerations).

Esta aplicação de *timer* definida pela biblioteca da *Nordic Semiconductor* possui o seguinte modo de funcionamento: primeiro é necessário fazer uma chamada à função da inicialização do mesmo, *app_timer_init()*[1], em seguida, é necessário criar uma instância do *timer* a ser utilizado utilizando o *macro APP_TIMER_DEF*[2].

Finalmente, pode-se, então, utilizar funções de *app_timer_create()* e *app_timer_start()*, a função de criação tem como parâmetro a instância criada, o modo de operação do *timer* e a função a ser chamada ao expirar o *timer* (*callback*) . Na função de início do *timer* o tempo de duração do *timer*, ou seja, o tempo necessário para que seja chamada a função de *callback* é enviado como parâmetro. Para mais detalhes, consultar SDK (2018, SDK Common Libraries/Application Timer/Functions)

Para fazer a contagem do tempo de execução da aplicação, foi implementado um contador que é incrementado a cada chamada da *callback* definida e o tempo de duração do *timer* foi definido em 100 milissegundos, podendo, desta forma, obter o resultado desejado.

Também foi implementado na chamada da *callback* uma função para piscar o *led* como um indicativo de funcionamento do dispositivo, implementado da seguinte maneira, a cada 30 segundos, o *led* é acionado por 200 milissegundos e depois desativado.

4.1.4 *Board Pinout*

Para a adequação do *softdevice s132* à placa de desenvolvimento ou à placa final do dispositivo foram necessárias algumas alterações na biblioteca de definição dos pinos. Para isto foi consultada a pinagem utilizada pelo módulo *NINA B112*[3] e seus respectivos pinos *GPIO* (*General Purpose I/O*) e então feitas as alterações no header de definição referentes à pinagem utilizada pela placa de desenvolvimento e pela placa específica final da aplicação.

[1] Pode ser acessada em SDK (2018, SDK Common Libraries, Application Timer, Functions)
[2] Pode ser acessado emSDK (2018, SDK Common Libraries/Application Timer/Macros)
[3] Pin Mapping do módulo Nina B112 em (SIM, 2018)

4.1.5 *Configuration Shell*

No intuito de realizar algumas mudanças de configuração após o módulo ser iniciado, foi criada uma interface do tipo *Shell* para que o usuário possa fazer alterações em alguns parâmetros de funcionamento do dispositivo utilizando o serviço *NUS* (*Nordic Uart Service*), através do mesmo, foram implementadas funções de tratamento de mensagens recebidas e de alterações em certas configurações detalhadas mais profundamente adiante.

Este desenvolvimento se iniciou através da mensagem recebida pela função *nus_data_handler*, para mais detalhes, consultar a referência SDK (2018, Bluetooth Low Energy Examples/BLE Peripheral/UART/Serial Port Emulation over BLE).

A partir da função implementada para o exemplo, foram feitas algumas alterações conforme pode ser visto na Figura 17. Foi criada uma variável global *uart* (do tipo *data_t* que contém dois valores, o primeiro é o length (do tipo *uint16_t*, para o tamanho do dado) e o segundo, *p_data* (do tipo *uint8_t[]* para armazenamento do dado em si)) além de uma flag do tipo *boolean* para indicar o recebimento de nova mensagem (*uart_rcvd*).

Figura 17: Função implementada para tratamento de mensagem recebida via *NUS*

```
void nus_data_handler(ble_nus_evt_t * p_evt) {

    if (p_evt->type == BLE_NUS_EVT_RX_DATA) {

        if (p_evt->params.rx_data.length <= m_ble_nus_max_data_len
                && p_evt->params.rx_data.length > 0) {
            uart.length = p_evt->params.rx_data.length;

            for (uint8_t i = 0; i < uart.length; i++)
                uart.p_data[i] = p_evt->params.rx_data.p_data[i];
            uart_rcvd = true;
        } else {
            printf("Command too long.\n");
        }
    }
}
```

Foi necessário a criação da variável global e a *flag* indicadora de mensagem pois a operação de tratamento da mensagem possui um alto custo computacional, e como a interrupção possui uma prioridade mais alta de execução, a aplicação parava de executar outras funções para tratar a interrupção, devido à isto, foi necessário fazê-la fora da mesma, portanto, no laço principal da função main foi verificada constantemente o recebimento de mensagens e então feito o tratamento.

Para o tratamento da mensagem, foi criado um *parser* que faz a divisão da mensagem recebida de acordo com os espaços (utilizando a função *isgraph*) e divide a mensagem

em um ponteiro para *array* (*(char *argv[])* onde cada *array* contém uma palavra e um contador de quantas palavras foram encontradas (*uint32_t argc*).

Para as chamadas de funções da *Shell*, foi implementada uma estrutura composta por uma *string* (*char *COMMAND*) e um ponteiro para função que recebe como parâmetro o número de palavras encontradas (*argc*) e o ponteiro de *array* resultante da função *parser* mencionada anteriormente (*argv*). Pode-se verificar a estrutura implementada na Figura 18.

Figura 18: Cabeçalho da estrutura utilizada acessar diferentes funcionalidades da *Shell*

```
typedef struct {
    char *COMMAND;
    uint32_t (*SHELL_FUNC)(uint32_t argc, char *argv[]);
} REMOTE_SHELL_COMMAND_STRUCT;
```

A inicialização da estrutura mencionada ocorre da seguinte forma, representada pela Figura 19.

Figura 19: Inicialização da estrutura da Shell

```
REMOTE_SHELL_COMMAND_STRUCT remote_shell_cmds[] = {
        { "major\0",        remote_shell_major          },
        { "minor\0",        remote_shell_minor          },
        { "advint\0",       remote_shell_adv_interval   },
        { "txpwr\0",        remote_shell_tx_power       },
        { "mode\0",         remote_shell_opmode         },
        { "name\0",         remote_shell_device_name    },
        { "time\0",         remote_shell_device_clock   },
        { "rstcnt\0",       remote_shell_reset_cnt      },
        { "fd\0",           remote_shell_facdefault     },
        { "save\0",         remote_shell_save_settings  },
        { "rst\0",          remote_shell_reset          },
        { "exit\0",         remote_shell_exit           },
        {  NULL,            NULL                        }
};
```

Esta estrutura foi utilizada para facilitar a procura pelo comando recebido e armazenado em *argv[0]* e então fazer a chamada da função correspondente. As funções implementadas pela *Shell* foram as seguintes:

a) *major*: retorna o valor do campo *major* do *advertise* do tipo *iBeacon* carregado ao iniciar a aplicação. Pode ser acompanhado de um parâmetro composto por 4 caracteres representando 2 dígitos *hexadecimal*. Por exemplo: *major 0x07c7* comando para mudar o valor do campo *major* para *0x07c7*.

b) *minor*: retorna o valor do campo *minor* do *advertise* do tipo *iBeacon* carregado ao iniciar a aplicação. Pode ser acompanhado de um parâmetro composto por 4 caracteres representando 2 dígitos *hexadecimal*. Por exemplo: *minor 0x0001* comando para mudar o valor do campo *minor* para *0x0001*.

c) *advint*: retorna o valor do *advertise interval* carregado ao iniciar a aplicação. Pode ser acompanhado de um parâmetro que representa um número inteiro positivo, podendo variar de 30 a 6000, valor que representa em milissegundos o intervalo de *advertising*. Por exemplo: *advint 400* altera o valor do intervalo de *advertising* para 400 milissegundos.

d) *txpwr*: retorna o valor do campo *Tx Power*, carregado ao iniciar o módulo, que define a força com que o sinal é enviado. Pode ser acompanhado de um parâmetro que representa um número inteiro, o qual pode admitir os seguintes valores: *-30, -20, -16, -12, -8, -4, 0 e 4*. Por exemplo: *txpwr -8*, altera o valor do *tx power* para -8dBm.

e) *mode*: retorna o modo de operação do dispositivo, sendo ele um dispositivo central ou periférico.

f) *name*: retorna o nome com que o dispositivo foi configurado no pacote de *scan response*. Pode ser acompanhado de um parâmetro que representa uma *string* de até 8 caracteres. Foi feito um tratamento na função de parse em que são reconhecidas as aspas ("), podendo, desta forma, o nome conter espaços e caracteres maiúsculos e minúsculos. Por exemplo: *name "NOVONOME"*, comando para alterar o nome do dispositivo para "NOVONOME".

g) *time*: retorna o tempo corrido desde que a aplicação foi iniciada.

h) rstcnt: retorna o valor do contador de *reset* do módulo, sendo incrementado a cada inicialização. Pode ser acompanhado do parâmetro zero que faz com que o contador seja zerado. Por exemplo: *rstcnt 0*, comando para zerar o contador de resets.

i) *fd*: retorna o valor da *flag Factory Default*, se esta vale *0x09*, o módulo carregará informações a partir da memória *flash* interna, caso contrário, carregará informações padronizadas. Pode ser acompanhado de um parâmetro que representa um valor inteiro para fazer a alteração do valor atual da *flag fd*. Por exemplo: *fd 9*, comando para alterar a *flag* para *0x09*.

j) *save*: comando para salvar as alterações realizadas. Ao alterar qualquer configuração do módulo, o novo valor é escrito no *buffer* mencionado na sessão *Flash Memory*, e então é necessário enviar este comando para gravar o *buffer* na memória *flash* interna do módulo.

k) *rst*: comando para fazer a re-inicialização do módulo. É necessário fazer a re-inicialização para que as alterações feitas e salvas possam entrar em vigor.

l) *exit*: comando para encerrar a conexão do dispositivo.

Em cada função chamada, após a identificação do comando, é feito um tratamento dos possíveis parâmetros a serem recebidos. Retornando uma mensagem de confirmação caso a operação seja executada com sucesso ou uma mensagem de erro caso contrário.

4.2 Dispositivo Central

Para o dispositivo central foi utilizado uma placa contendo *leds* indicativos; um *buzzer* e uma sirene, utilizado para emissão de alertas; um botão de ação, que será explicado adiante, e um botão para inicialização do dispositivo em modo de configuração ou modo de operação normal.

Este dispositivo consiste em um módulo que realiza operações de scan para a identificação de *advertisings* provenientes de outros dispositivos. Em resumo, é um módulo que faz a identificação de outros dispositivos, os armazena, de acordo com seu *major value*, em uma lista encadeada ordenada de acordo com a média da força de sinal recebida de cada pacote de *advertising*.

Neste modo de operação foram utilizadas as funções implementadas para o dispositivo periférico, com algumas modificações e adição de outras implementações:

4.2.1 Configurações de *Scan*

Analogamente à função de *advertising*, é necessário, primeiramente, fazer algumas configurações de alguns parâmetros e então fazer a chamada da função de inicialização desses parâmetros através do método *ble_conn_params_init*. Para mais informações, consultar SDK (2018, API Reference/BLE Libraries/Functions).

Após definidas as configurações dos parâmetros de conexão, é possível iniciar o processo de *scan* com a chamada da função *sd_ble_gap_scan_start* obtida em SDK (2018, API References/Generic Access Profile (GAP)/Functions)

O parâmetro para a função de inicialização do processo de *scan* é definido por uma constante do tipo *ble_gap_scan_params_t* que pode ser observada Figura 20.

Figura 20: Definição da variável para configuração de *Scan*

```
ble_gap_scan_params_t const m_scan_params = { .active = 1, .interval =
SCAN_INTERVAL, .window = SCAN_WINDOW, .timeout = SCAN_TIMEOUT,
#if (NRF_SD_BLE_API_VERSION <= 2)
        .selective = 0,
        .p_whitelist = NULL,
#endif
#if (NRF_SD_BLE_API_VERSION >= 3)
        .use_whitelist = 0,
#endif
        };
```

As constantes *SCAN_INTERVAL*, *SCAN_WINDOW* e *SCAN_TIMEOUT* foram definidas conforme mostra a Figura 21.

Figura 21: Definição das constantes para configuração de Scan

```
#define SCAN_INTERVAL                (uint16_t) MSEC_TO_UNITS(100, UNIT_0_625_MS)
#define SCAN_WINDOW                  (uint16_t) MSEC_TO_UNITS(100, UNIT_0_625_MS)
#define SCAN_TIMEOUT                 0x0001
```

É necessária a utilização do macro *MSEC_TO_UNITS* pois os parâmetros de intervalo de *scan* e janela de scan são definidos em unidades de 0,625 milissegundos pela biblioteca da *SDK*.

Após inicializada a função de scan, caso algum dispositivo seja encontrado, é feita uma chamada à função *ble_evt_handler*, que possui como parâmetro uma variável do tipo *ble_evt_t*, a qual carrega diversas informações referentes ao tipo de evento ocorrido. No caso de um *advertise* encontrado, seu *id* é *BLE_GAP_EVT_ADV_REPORT*, como pode ser observado no trecho de código mostrado na Figura 22.

Caso um *advertise* seja encontrado, foi implementada uma função para o tratamento do mesmo. Esta função é responsável por interpretar os dados recebidos, verificar se o pacote segue o padrão *iBeacon*, de acordo com o especificado anteriormente e fazer a verificação do campo *major*. Nesta aplicação, são visados dispositivos cujo campo *major* é definido com o valor *0x07c7*. Caso seja encontrado um dispositivo com esta característica, seus dados são enviados à um gerenciador de lista encadeada para a inserção dos mesmos, criando desta forma, uma lista que representa dispositivos encontrados.

Figura 22: *Event Handler*

```
void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context) {
    uint32_t err_code;

    switch (p_ble_evt->header.evt_id) {

    case BLE_GAP_EVT_TIMEOUT:
        scan_timeout = true;
        err_code = start_scanning();
        if (err_code == NRF_ERROR_INVALID_STATE) {
            err_code = sd_ble_gap_scan_stop();
            if (!err_code)
                err_code = start_scanning();
        }
        break;
    case BLE_GAP_EVT_ADV_REPORT:   //device found
        adv_found(p_ble_evt);
        break;
    case BLE_GAP_EVT_CONNECTED:
        service_discovery(p_ble_evt);
        set_m_conn_handle(p_ble_evt);
```

4.2.2 Lista Encadeada para dispositivos encontrados

Para a lista encadeada, foi necessário fazer a inicialização do gerenciador de memória fornecido pela *SDK*, através da mesma, os endereços de memória alocados dinamicamente estão protegidos, sem que algum endereço utilizado pela aplicação seja sobrescrito.

Para a inserção de um novo elemento à lista encadeada, são enviadas as seguintes informações: o *MAC address Bluetooth* do dispositivo, o *RSSI*, o *RSSI* à um metro de distância (contido no pacote de *advertising* do *iBeacon*) e o *Tx Power* do dispositivo. Através do *Tx Power*, *RSSI* e *RSSI at one meter* é possível estimar uma distância aproximada do dispositivo, a fórmula para essa estimativa será explicada na próxima sessão. Desta forma, a lista de dispositivos utilizada possui a estrutura ilustrada na Figura 23.

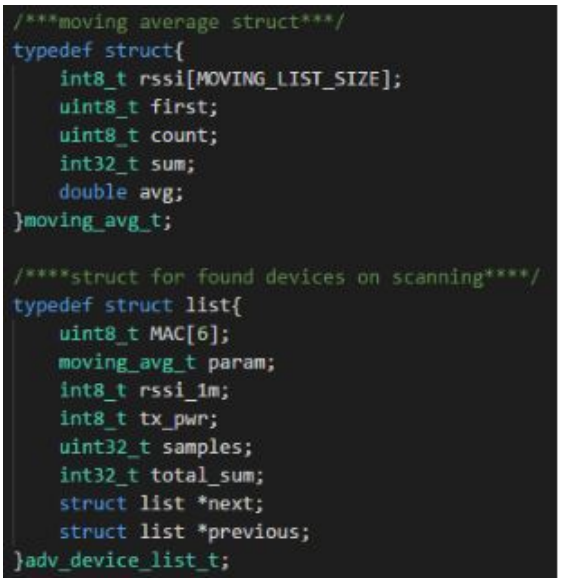
Os campos da estrutura de lista de dispositivos são os seguintes:

a) *MAC*: um array do tipo *uint8_t* com tamanho 6, para armazenamento do endereço *MAC Bluetooth* do elemento.

b) *param*: este elemento do tipo *moving_avg_t* foi utilizado para o cálculo da média dos *RSSI*s obtidos. Devido à alta instabilidade e grande discrepância na força do sinal recebido (*RSSI*) fez-se o uso de uma média dos valores recebidos, para isto foi utilizada uma média móvel em uma lista circular e os elementos foram ordenados de acordo com o valor *avg* (média dos *RSSI*s obtidos).

c) *rssi_1m*: valor lido do último campo do *advertise* do *iBeacon*.

Figura 23: Estrutura implementada para a lista encadeada



```
/***moving average struct***/
typedef struct{
    int8_t rssi[MOVING_LIST_SIZE];
    uint8_t first;
    uint8_t count;
    int32_t sum;
    double avg;
}moving_avg_t;

/****struct for found devices on scanning****/
typedef struct list{
    uint8_t MAC[6];
    moving_avg_t param;
    int8_t rssi_1m;
    int8_t tx_pwr;
    uint32_t samples;
    int32_t total_sum;
    struct list *next;
    struct list *previous;
}adv_device_list_t;
```

d) *tx_pwr*: força com que o sinal do dispositivo é transmitida.

e) *samples*: número de amostras obtidas, utilizada para calcular uma média a partir do somatório total de todos os *RSSI*s obtidos.

f) *total_sum*: somatório total de todos os *RSSI*s obtidos

g) *next* e *previous*: ponteiros para o próximo elemento e elemento anterior da lista encadeada, respectivamente.

Os dispositivos são identificados pelos seu *MAC address*, portanto, primeiro, verifica-se se algum elemento da lista possui um *MAC address* igual ao do dispositivo encontrado. Em caso afirmativo, seus dados são atualizados, caso contrário, um novo elemento é adicionado à lista.

Caso o novo elemento seja adicionado à lista, e a lista está vazia, ele ocupará a primeira posição da lista. No entanto, se a lista já possuir elementos, o parâmetro de ordenação da lista é a média de *RSSI* dos mesmos, portanto, o novo elemento terá seu *RSSI* comparado à média de *RSSI* de cada elemento já presente na lista. Se seu *RSSI* for maior que o *RSSI* de um dado elemento da lista (vale salientar que a lista está sempre ordenada e é percorrida do primeiro ao último elemento), o novo elemento é adicionado antes do elemento cujo *RSSI* é menor. Caso chegue ao fim da lista e não encontre *RSSI* menor que o seu, ele é adicionado ao final da lista. Desta forma, a lista mantém-se ordenada decrescentemente em relação ao *RSSI* de cada elemento.

No caso do elemento já possuir seu *MAC address* na lista, seus dados são atualizados. O processo de atualização de um elemento consiste na adição do seu valor de *RSSI* lido

por último à estrutura *moving_avg_t*. Como pode ser observado na Figura 24.

Figura 24: Trecho de código implementado para atualização de parâmetros de média.

```
void utils_update_avg_param(adv_device_list_t *elem, double rssi){
    elem->total_sum += rssi;
    if(elem->param.count == MOVING_LIST_SIZE){
        elem->param.sum -= elem->param.rssi[elem->param.first];
        elem->param.sum += rssi;
        elem->param.rssi[elem->param.first] = rssi;
        if(elem->param.first >= 0 && elem->param.first < MOVING_LIST_SIZE - 1)
            elem->param.first++;
        else if(elem->param.first == MOVING_LIST_SIZE - 1)
            elem->param.first = 0;
    }
    else{
        elem->param.rssi[elem->param.count] = rssi;
        elem->param.count++;
        elem->param.sum += rssi;
    }
    elem->param.avg = (elem->param.sum/elem->param.count);
}
```

Uma breve explicação da atualização das informações do dispositivo: primeiramente verifica-se se a lista está em seu tamanho máximo (*MOVING_LIST_SIZE*), no caso, foi escolhido um tamanho de dez amostras. Caso ela já contenha dez amostras, é subtraído do campo *sum* (que contém a soma dos *RSSI*s da estrutura) o valor de *RSSI* do primeiro elemento da lista (denotado pelo índice *first*) e então adicionado o novo valor lido à somatória. Em seguida, é atualizado o valor do elemento de índice *first* e, se este elemento não for o último da lista, é incrementado, caso contrário, recebe zero.

Se a lista não estiver cheia, simplesmente é adicionado um novo elemento e seu contador é incrementado. Após a atualização dos parâmetros de média móvel, o elemento atualizado é retirado da lista e inserido novamente fazendo as verificações sobre o valor da média de *RSSI* dos demais elementos da lista e inserindo-o ordenadamente.

Vale mencionar que a lista é apagada a cada dez interrupções causadas pelo evento de *scan timeout*, evitando, desta maneira, que os contadores de sample atinjam seus limites e dispositivos que ocupavam a primeira posição da lista, caso forem desligados ou saírem rapidamente das proximidades, não ocupem eternamente esta posição.

4.2.3 Cálculo da Distância

O cálculo da distância foi implementado conforme a ilustração da Figura 25.

Figura 25: Trecho de código implementado para Cálculo da distância de dispositivos.

```
double utils_distance_calculate(int8_t rssi_1m, double rssi, int8_t tx_pwr){
    double rssi1m = tx_pwr + rssi_1m + adjust;
    double exp = ((rssi1m - rssi)) / (10 * ENV_CONST);
    return pow(10, exp);
}
```

A partir dos parâmetros *RSSI* à um metro, *RSSI* lido e *Tx Power* do dispositivo pode-se estimar a distância do mesmo ao dispositivo Central que está realizando operação de *Scan*. O material utilizado como referência para cálculo da distância se encontra em RSSI Indoor Localization (2015). E pode ser representada pela sequinte equação

$$\log_{10}(dist) = (rssi1m - rssi)/(10 * ENVCONST) \tag{4.1}$$

Onde as variáveis são:

a) *dist*: distância estimada do dispositivo analisado.

b) *rssi1m*: valor do campo *RSSI at one meter* lido do pacote *advertising* do *iBeacon*.

c) *rssi*: valor do *RSSI* recebido pelo *advertise*.

d) *ENVCONST*: constante de ambiente utilizada para adequar a estimativa de acordo com o ambiente de funcionamento do dispositivo.

No caso desta aplicação, para que não fosse necessária a alteração no valor do campo *RSSI at one meter* do *iBeacon advertising* foi feito uma estimativa do valor do *RSSI* à um metro de distância de acordo com o valor do *Tx Power* com que o dispositivo foi configurado, normalmente calibrado com o valor mais alto (4 dBm). Esta estimativa consiste em somar o valor do *Tx Power* do dispositivo ao valor de *RSSI at one meter*, fixado com o valor *0xc7* que, em complemento de dois, pode ser interpretado como -56dBm. À este valor é somado um valor de ajuste, inicialmente valendo zero, mas que pode ser alterado no caso do usuário desejar realizar uma nova calibração de distância do dispositivo, explicada mais detalhadamente no decorrer deste trabalho.

Outro fator importante para o cálculo da estimativa de distância é a constante *ENVCONST* que deve ser configurada de acordo com o ambiente da aplicação, esta constante normalmente assume valores entre 1 e 5 e é escolhida de acordo com as interferências externas do meio em que se encontra o dispositivo. Após a realização de alguns testes experimentais de medidas de distância e distância estimada pela fórmula, pode-se concluir, que, em ambiente fechado, populado por computadores e outros dispositivos eletrônicos, o valor de *ENVCONST* que proporciona uma estimativa mais próxima da distância real é 2.

4.2.4 Acionamento de Alertas

A proposta é que o módulo central acione um tipo de alerta de acordo com a proximidade de dispositivos periféricos. Para isto, a cada interrupção gerada pelo *scan timeout* foi feita a chamada à uma função de acionamento de alertas, como pode ser ilustrada na Figura 26.

Figura 26: Trecho de código implementado para a escolha do tipo de alerta a ser acionado.

```
bool board_assert_warning_signal() {
    bool check = false;

    adv_device_list_t *tmp = utils_check_op_mac();
    if(!tmp){
        check = board_priority_handler(WARNING_SIGNAL_NONE);
        return false;
    }

    double distance = utils_distance_calculate(tmp->rssi_1m, tmp->param.avg,
            tmp->tx_pwr);

    if (distance <= (double) warning_distance[WARNING_SIGNAL_HIGH - 1]
            && distance > 0) {
        check = board_priority_handler(WARNING_SIGNAL_HIGH);

    } else if (distance <= (double) warning_distance[WARNING_SIGNAL_MEDIUM - 1]
            && distance > (double) warning_distance[WARNING_SIGNAL_HIGH - 1]) {
        check = board_priority_handler(WARNING_SIGNAL_MEDIUM);

    } else if (distance <= (double) warning_distance[WARNING_SIGNAL_LOW - 1]
            && distance
                    > (double) warning_distance[WARNING_SIGNAL_MEDIUM - 1]) {
        check = board_priority_handler(WARNING_SIGNAL_LOW);

    } else if (distance > (double) warning_distance[WARNING_SIGNAL_LOW - 1]) {
        check = board_priority_handler(WARNING_SIGNAL_NONE);
    } else {
        check = board_priority_handler(WARNING_SIGNAL_NONE);
    }
    return check;
}
```

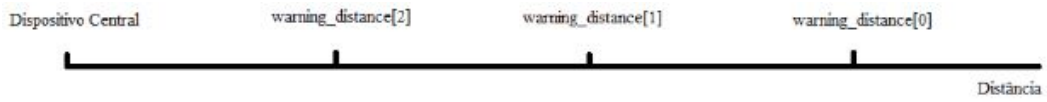
Primeiramente, é feita a verificação se o *MAC address* do primeiro elemento da lista encadeada de dispositivos é um *MAC Address* a ser ignorado, esta verificação é feita pela chamada *utils_check_op_mac* a qual faz esta verificação e caso o primeiro elemento possua o *MAC address* salvo como *MAC* a ser ignorado a função retorna um ponteiro para o segundo elemento da lista, caso contrário, retorna um ponteiro para o primeiro elemento. Em seguida, é feita a chamada para o cálculo da distância estimada de acordo com os parâmetros salvos pela estrutura da lista e então, o valor da distância é armazenado e comparado.

Para fazer a comparação, foi criado um *array* global do tipo *uint8_t* de tamanho

3, definindo, desta maneira, 3 distâncias (com valores padrões que podem ser alterados futuramente através da *Shell*) para o acionamento de diferentes alertas. As distâncias definidas são as seguintes: na primeira posição (posição zero) encontra-se a maior distância, definida por padrão como 50, na segunda posição, a distância média definida como 20 e para a terceira posição a distância mais próxima, defnida como 1. Foi feito um tratamento para que as distâncias sempre obedeçam uma ordem, isto é, o valor da primeira posição, deve sempre ser maior que da segunda e terceira; o valor da segunda posição deve sempre ser menor que da primeira e maior que da terceira e da terceira posição deve sempre ser menor de todas.

No intuito de facilitar a visualização, foi definido um *enum* com constantes com valores 0, 1, 2 e 3, sendo elas *WARNING_SIGNAL_NONE*, *WARNING_SIGNAL_LOW*, *WARNING_SIGNAL_MEDIUM* e *WARNING_SIGNAL_HIGH* respectivamente. Uma representação das distâncias pode ser observada na Figura 27.

Figura 27: Representação das distâncias no *array warning_distance[]*.



Após a definição do dispositivo em qual limite de distância o mesmo se encontra, é feita uma chamada à função para acionamento dos alertas *board_priority_handler* que possui como parâmetro o tipo de alerta a ser acionado. Nesta função de acionamento de alerta foi implementado uma ordem de prioridade com relação ao novo alerta a ser acionado, isto é, uma alerta do tipo *WARNING_SIGNAL_HIGH* desativa todos os outros tipos de alerta e passa a entrar em vigor. No caso de um alerta do tipo *WARNING_SIGNAL_MEDIUM*, este passa a entrar em vigor somente se o alerta anterior for do tipo *WARNING_SIGNAL_LOW* ou *WARNING_SIGNAL_NONE* (sem acionamento de alerta) e, por último, caso o novo alerta seja do tipo *WARNING_SIGNAL_LOW*, passará a ser acionado somente se o alerta anterior for do tipo *WARNING_SIGNAL_NONE*.

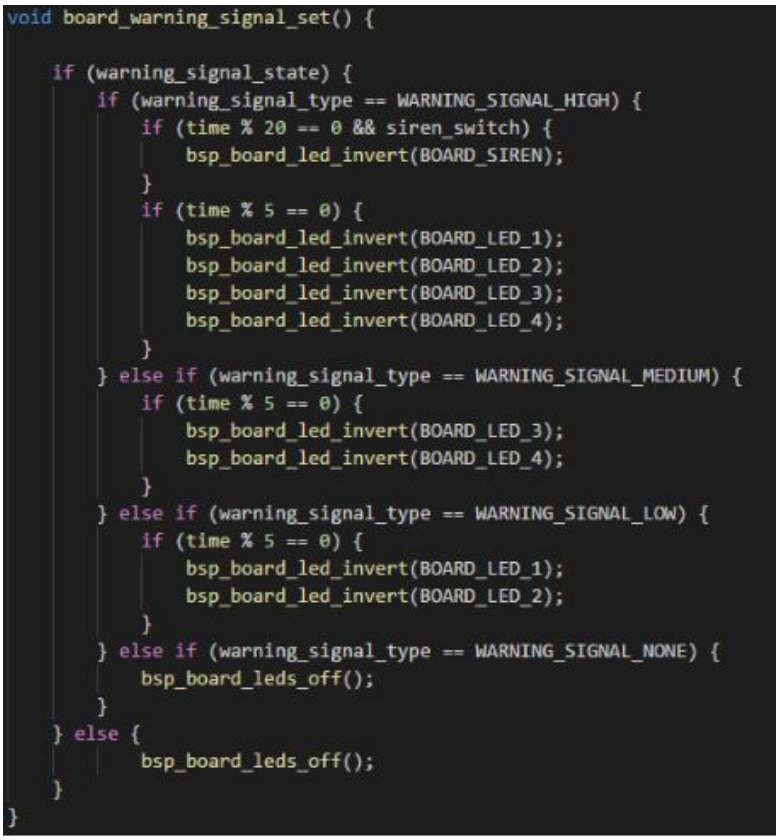
Para cada tipo de alerta foi implementado um tempo de *timeout* para o mesmo, isto é, um tempo para que o alerta se encerre e retorne ao estado *WARNING_SIGNAL_NONE*. Em cada acionamento de alerta, uma variável global *warning_signal_state* do tipo *boolean* é definida juntamente com uma variável global *warning_signal_type*, do tipo *uint8_t*, com o tipo de alerta a ser acionado. A Figura 28 ilustra a função implementada para fazer o acionamento dos alertas.

Figura 28: Trecho de código utilizado para o tratamento de prioridades do sinal.

```
bool board_priority_handler(warning_signal_type_t next_signal_type) {
    static uint32_t end_time;   //variavel estatica para armazenamento do tempo de timeout
    if (next_signal_type > 3 || next_signal_type < 0)
        return false;

    if (next_signal_type > warning_signal_type) {//se o proximo sinal tem prioridade
        if (warning_signal_state) {                 //se esta ligado o sinal
            bsp_board_leds_off();
        } else {
            warning_signal_state = true;        //alerta ligado
        }
        warning_signal_type = next_signal_type;     //atualiza tipo de alerta
        end_time = time + SIGNAL_DURATION;          //inicializacao do timeout
        return true;

    } else if (next_signal_type == warning_signal_type) {
        end_time = time + SIGNAL_DURATION;      //renovacao do timeout
        return true;

    //caso o sinal recebido possua menor prioridade
    } else if (next_signal_type < warning_signal_type) {
        if (time >= end_time) {     //verifica;'ao do timeout
            bsp_board_leds_off();
            end_time = time + SIGNAL_DURATION;
            warning_signal_type = next_signal_type;
        }
        return true;

    //caso nao tenha dispositivos proximos
    } else if (next_signal_type == WARNING_SIGNAL_NONE) {
        warning_signal_state = false;
        warning_signal_type = WARNING_SIGNAL_NONE;
    }
    return false;
}
```

As variáveis globais foram utilizadas para que outros trechos do código pudessem fazer a varredura de seus valores, como o acionamento dos *leds*, *buzzer* e sirene. O acionamento dos *leds* e da sirene foi implementado na função de *callback* do timer contador de tempo da aplicação ilustrado pela Figura 29.

Figura 29: Trecho de código utilizado para o acionamento dos sinais de acordo com o tempo e variáveis globais.



```
void board_warning_signal_set() {

    if (warning_signal_state) {
        if (warning_signal_type == WARNING_SIGNAL_HIGH) {
            if (time % 20 == 0 && siren_switch) {
                bsp_board_led_invert(BOARD_SIREN);
            }
            if (time % 5 == 0) {
                bsp_board_led_invert(BOARD_LED_1);
                bsp_board_led_invert(BOARD_LED_2);
                bsp_board_led_invert(BOARD_LED_3);
                bsp_board_led_invert(BOARD_LED_4);
            }
        } else if (warning_signal_type == WARNING_SIGNAL_MEDIUM) {
            if (time % 5 == 0) {
                bsp_board_led_invert(BOARD_LED_3);
                bsp_board_led_invert(BOARD_LED_4);
            }
        } else if (warning_signal_type == WARNING_SIGNAL_LOW) {
            if (time % 5 == 0) {
                bsp_board_led_invert(BOARD_LED_1);
                bsp_board_led_invert(BOARD_LED_2);
            }
        } else if (warning_signal_type == WARNING_SIGNAL_NONE) {
            bsp_board_leds_off();
        }
    } else {
            bsp_board_leds_off();
    }
}
```

Onde as constantes *BOARD_LED_1*, *BOARD_LED_2*, *BOARD_LED_3*, *BOARD_LED_4* e *BOARD_SIREN* foram definidas na biblioteca de definição dos pinos.

Para a ativação do *buzzer* foi utilizada a implementação do *PWM*. Para isto, foram feitas as configurações dos parâmetros para a criação do mesmo e então chamada a função de inicialização do *PWM*, a função *app_pwm_init*, que pode ser acessada em SDK (2018, SDK common libraries/Pulse-width modulation (PWM)/Functions).

Para se fazer esta chamada, é necessário, primeiro, criar uma instância do *PWM* a partir da chamada do macro *APP_PWM_INSTANCE* e carregar configurações do mesmo a partir do macro *APP_PWM_DEFAULT_CONFIG_1CH* ambos fornecidos em SDK (2018, SDK common libraries/Pulse-width modulation (PWM)/Macros).

Para isto, foi definido um pino específico, na biblioteca de definição dos pinos, para o *PWM*, no caso, foi utilizada a configuração de somente um canal, portanto, somente um pino precisou ser definido para este fim. Como parâmetro adicional, foi utilizada uma frequência de 1.200KHz para inicialização.

Para temporizar o buzzer foi criado um timer específico para este fim, com tempo

para timeout de 100 milissegundos, tendo como função de *callback* uma verificação na variável global de estado de alerta (*warning_state*) juntamente com um contador estático para controle dos intervalos de acionamento do *buzzer*. Ao ser acionado, uma variável global *beep* do tipo *boolean* é definida e varrida no *loop* principal da aplicação. Caso a variável *beep* seja *true*, ocorre o acionamento do *PWM* através da chamada *app_pwm_enable*, caso contrário, uma chamada à função *app_pwm_disable* é feita as quais pode ser acessada em SDK (2018, SDK common libraries/Pulse-width modulation (PWM)/Functions).

4.2.5 Configurações adicionais da Shell

Para o módulo Central, foram adicionados alguns comandos adicionais à *Shell* implementada, como mostra a Figura 30.

Figura 30: Inicialização da estrutura da Shell para dispositivo Central.

```
REMOTE_SHELL_COMMAND_STRUCT remote_shell_cmds[] = {
        { "major\0",        remote_shell_major          },
        { "minor\0",        remote_shell_minor          },
        { "advint\0",       remote_shell_adv_interval   },
        { "txpwr\0",        remote_shell_tx_power       },
        { "mode\0",         remote_shell_opmode         },
        { "name\0",         remote_shell_device_name    },
        { "time\0",         remote_shell_device_clock   },
        { "save\0",         remote_shell_save_settings  },
        { "rst\0",          remote_shell_reset          },
        { "rstcnt\0",       remote_shell_reset_cnt      },
        { "fd\0",           remote_shell_facdefault     },
        { "c\0",            remote_shell_calibrate_rssi },
        { "op",             remote_shell_save_op_mac    },
        { "dist\0",         remote_shell_warn_dist_set  },
        { "beep\0",         remote_shell_beep_enable    },
        { "siren\0",        remote_shell_siren_enable   },
        { "exit\0",         remote_shell_exit           },
        {  NULL,            NULL                        }
};
```

Além dos comandos implementados anteriormente, houve a adição dos seguintes comandos:

a) *c*: Comando para calibrar a distância estimada. Retorna o valor do ajuste utilizado e a distância do dispositivo mais próximo. Aceita como parâmetro um número inteiro entre 0 e 100. A calibração foi implementada da seguinte maneira, o parâmetro a ser passado (o número inteiro) é interpretado como a distância desejada para o dispositivo mais próximo, desta maneira, ocorre a chamada de uma função que utiliza a fórmula mencionada anteriormente para cálculo da distância, porém, em função do valor de ajuste, como pode ser ilustrado na Figura 31. Vale salientar que a variável de ajuste (*adjust*) é uma

variável global do tipo *double*. E caso o parâmetro enviado seja zero, o valor da variável *adjust* é zerado. Por exemplo o comando "c 3"faz com que a variável *adjust* seja calculada para que a distância do dispositivo mais próxima seja 3.

b) *op*: retorna o endereço *MAC* a ser ignorado no momento de acionar algum tipo de alerta, se o dispositivo mais próximo possuir este endereço *MAC*, a etapa de definição do tipo de alerta ignorará o mesmo e analisará o próximo elemento da lista. Aceita como parâmetro zero ou um. Caso seja zero, apaga o endereço salvo a ser ignorado, caso contrário, salva o endereço MAC do dispositivo mais próximo. Por exemplo o comando "op 1"faz com que o *MAC address* do dispositivo mais próximo seja ignorado no momento de definição do alerta. Esta chamada também é feita caso o botão de ação do módulo seja pressionado. Se for pressionado por 3 segundos o módulo interpreta como o envio de um comando *op 0*, caso seja somente pressionado, é interpretado como *op 1*.

c) *dist*: retorna os valores do *array warning_distance*. Aceita um ou dois parâmetros, caso seja enviado um parâmetro, este deve estar entre 0 e 3, retornando o valor de *warning_distance* na posição correspondente ao parâmetro enviado. Por outro lado, se forem enviados dois parâmetros, ocorre a alteração do valor de *warning_distance* na posição correspondente ao primeiro parâmetro para o valor correspondente ao segundo parâmetro. Por exemplo o comando "dist 0 40"altera o valor do *array warning_distance[0]* para 40.

d) *beep*: retorna se o *buzzer* está ativo ou não, ou seja, se este vai ser acionado em caso de alerta. Aceita como parâmetro 0 ou 1, caso seja 0, desativa o *buzzer*, caso contrário, ativa-o.

e) *siren*: retorna se a sirene está ativa ou não, ou seja, se esta vai ser acionada em caso de alerta. Aceita como parâmetro 0 ou 1, caso seja 0, desativa a sirene, caso contrário, ativa-a.

Figura 31: Função implementada para calibração da distância.

```
void utils_calibrate_adjust(uint8_t distance, int8_t tx_pwr, int8_t rssi_1m, int8_t rssi){
    double aux = 10*ENV_CONST*(log10(distance));
    if(aux < 200 && aux > -200)
    adjust = (double)(aux + rssi - tx_pwr - rssi_1m);
}
```

4.2.6 *NRF DFU* (*Nordic Device Firmware Update*)

O *NRF DFU* é uma aplicação, fornecida pela *SDK* da *Nordic Semiconductor*, do tipo *Bootloader*, que é, basicamente, um trecho de código que é executado antes da inicialização de qualquer Sistema Operacional e são utilizados para fazer o carregamento de outro Sistema Operacional ou Aplicação.

No caso da aplicação utilizada (*NRF DFU*), foi necessário utilizar o *software nrfutil.exe* para a geração de chaves privadas e públicas através do envio de linhas de código via terminal mostradas a seguir.

```
nrfutil.exe keys generate private.key
```

```
nrfutil.exe keys display --key pk
--format code private.key --out_file public_key.c
```

Após a aquisição das chaves, foi necessário fazer o *download* da biblioteca *uECC* bem como a substituição do arquivo *dfu_public_key.c*, presente na pasta da aplicação, pelo arquivo *public_key.c*, gerado anteriormente, para a compilação do código do *Bootloader*.

Compilado o código do *Bootloader* foi gerado o pacote imagem contendo o *softdevice*, *bootloader* e a aplicação para a gravação no módulo. Esta etapa foi executada da seguinte maneira: primeiramente, foi necessário gerar o *Booloader Settings* da aplicação através do comando:

```
nrfutil settings generate --family NRF52 --application myapplication.hex
--application-version 0 --bootloader-version 0 --bl-settings-version 1
bootloader_settings.hex
```

Após a obtenção do *Bootloader Settings*, foi utilizado o comando *mergehex* para juntar as imagens do *Bootloader* e do *Bootloader Settings* em uma única imagem. Então foi feita a união de todos os elementos em uma única imagem para a gravação do módulo.

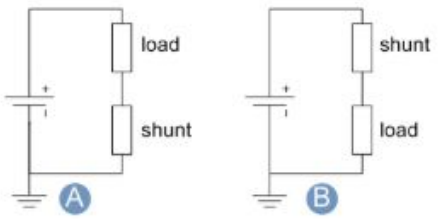
Com relação ao código do *Bootloader* foi feita uma pequena adição de condição para a execução do código. Caso o botão do dispositivo esteja pressionado ao ser ligado, o mesmo inicializará em modo *DFU*, que consiste em um modo de *advertising* com o nome *DFU* o qual aguarda uma conexão e o dispositivo conectado pode fazer a transferência de imagens de aplicações, ou seja, com este *Bootloader* é possível fazer atualização de aplicação utilizando o *Bluetooth Low Energy*. Se o botão não estiver pressionado, ocorre a execução normal da aplicação gravada no dispositivo.

4.2.7 Medidas de Consumo de Energia

O consumo de energia é crucial no desenvolvimento de sistemas embarcados pois alguns dispositivos desenvolvidos dependem de bateria e muitas vezes a aplicação pode não ser viável devido ao alto consumo de energia. Tendo isto em vista, foram feitas algumas alterações no código visando uma redução no número de operações. Desta forma, foi feita a desabilitação da comunicação *UART*, a implementação de um tratamento diferente de varredura de variáveis (por exemplo, varredura feita na *callback* do *timer* ao invés do loop da função principal), entre outras.

Para a estimativa do consumo de energia, foi utilizado o método de medida de tensão no resistor *Shunt*, que é um resistor de baixa resistividade anexado à alimentação do circuito, como pode ser observado na Figura 32.

Figura 32: Método para medição de consumo de energia através do Resistor *Shunt*.



A. Often the shunt is placed in the grounded side to eliminate the common mode voltage. However, other disadvantages exist.
B. In this configuration, the common mode voltage could be too high for the ammeter.

Fonte: (SHUNT RESISTOR, 2018)

De acordo com a Figura 32, foi feita a ligação do tipo B e então, com o Osciloscópio foi medida a tensão entre o resistor *Shunt* e o circuito da aplicação. Um detalhe a ser mencionado foram os picos de tensão em cada envio de *advertising*, desta forma pode-se comprovar que quanto menor o tempo de intervalo de *advertising*, maior é o consumo de energia final do sistema.

Através de estimativas e aproximações, foi calculada a área sob o gráfico gerado de tensão por tempo e chegou-se a uma aproximação de consumo de 2.2mAh para o dispositivo periférico que foi implementado para que sua alimentação fosse proveniente de pilhas comuns do tipo AAA. No caso do dispositivo central não se fez esta medida pois o mesmo foi implementado considerando uma conexão constante à uma fonte de alimentação.

5 CONCLUSÃO

Ao fim deste projeto, pode-se constatar que muitos ensinamentos e conceitos, inicialmente teóricos, obtidos durante a graduação foram colocados em prática durante o desenvolvimento do mesmo. Sendo estes conhecimentos da área de *software* e *hardware*, o que reflete fortemente a intenção do curso de Engenharia de Computação, o conhecimento e integração de âmbos os meios computacionais, físicos e virtuais.

Foram colocados em práticas conceitos de *software* como desenvolvimento e estruturamento de códigos, diferentes maneiras de implementações de funcionalidades e suas vantagens e desvantagens com relação ao custo computacional e resultado desejado além da especificação de *software*s para sistemas embarcados.

Também pode-se experimentar conceitos de *hardware* como a implementação de circuitos adicionais para os componentes utilizados, conceitos de variação de sinais e níveis de tensão e a experiência prática com componentes elétricos.

Vale enfatizar o estudo da integração de ambas as áreas (*software* e *hardware*) através da implementação de sistemas embarcados, que foi o alvo deste trabalho.

Pode-se adquirir conhecimentos sobre a tecnologia *Bluetooth Low Energy*, como seus modos de operação e funcionamento de protocolos adotados oficialmente. A partir de tais conhecimentos é possível entender e desenvolver, à nível mais detalhado, várias aplicações corriqueiras.

Houve também um estudo sobre força de sinal do tipo *RFID* e suas variações e instabilidade, bem como o tratamento dos mesmos.

Finalmente, as maiores aquisições no desenvolver deste projeto, foram: a verificação de conteúdos estudados previamente colocados em prática e o aprendizado do padrão de funcionamento da tecnologia *Bluetooth*.

REFERÊNCIAS

ARM.MBED. **Understanding the different types of BLE Beacons**: alt beacon, beacon, ibeacon, uribeacon. USA, 2018. Disponível em: <https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/>. Acesso em: 27 feb. 2018.

BAú DA ELETRôNICA. **Buzzer**: Item buzzer 5v. São Paulo, 2018. Disponível em: <http://www.baudaeletronica.com.br/buzzer-5v.html>. Acesso em: 20 jan. 2018.

BLE-STACK USER'S GUIDE. **GAP STATE DIAGRAM**: Generic access profile. USA, 2018. Disponível em: <http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html>. Acesso em: 15 feb. 2018.

BLUETOOTH. **Advertising Types**: Advertising types. USA, 2018. Disponível em: <https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile>. Acesso em: 18 feb. 2018.

_____. **BLUETOOTH**: Company website. 5209 Lake Washington Blvd NE, 2018. Disponível em: <https://www.bluetooth.com>. Acesso em: 20 out. 2017.

_____. **GATT Overview**: Generic attribute. USA, 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt/generic-attributes-overview>. Acesso em: 15 feb. 2018.

_____. **GATT Profile**: Generic attribute profile. USA, 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt>. Acesso em: 15 feb. 2018.

_____. **GATT Profile Services**: Generic attribute profile. USA, 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt/services>. Acesso em: 15 feb. 2018.

_____. **GATT Service Example**: Generic attribute profile. USA, 2018. Disponível em: <https://www.bluetooth.com/specifications/assigned-numbers/environmental-sensing-service-characteristics>. Acesso em: 15 feb. 2018.

CME Project: My nina b112 advertising application. [S.l.].

ECLIPSE. **Eclipse Download Page**: Eclipse neon 3 packages. USA, 2018. Disponível em: <http://www.eclipse.org/downloads/packages/release/Neon/3>. Acesso em: 23 oct. 2017.

FACULTY OF COMPUTER SCIENCE. **Evaluation of Reliability of RSSI for Indoor Localization**: Rssi for indoor localization. Germany, 2015. Disponível em: <https://www.rn.inf.tu-dresden.de/dargie/papers/icwcuca.pdf>. Acesso em: 14 jan. 2018.

IBEACON INSIDER. **What is iBeacon? A guide to Beacons**: What is ibeacon? what are ibeacons? USA, 2018. Disponível em: <http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacons/>. Acesso em: 27 feb. 2018.

IBEACON-PORTUGAL. **iBeacon Image**: Company website. Portugal, 2018. Disponível em: <http://ibeacon-portugal.com/sobre-nos-ibeacon-portugal/>. Acesso em: 25 nov. 2017.

KONTAKT.IO. **Kontakt.io Tx Power**: Tx power value and distance. USA, 2018. Disponível em: <https://support.kontakt.io/hc/en-gb/articles/201621521-Transmission-power-Range-and-RSSI>. Acesso em: 22 feb. 2018.

METAL CASTY. **Sirene**: Item sirene 12v. São Paulo, 2018. Disponível em: <http://www.metalcasty.com.br/produto/sirene-audio-de-incendio/>. Acesso em: 20 jan. 2018.

NORDIC SEMICONDUCTOR. **GAP advertising and scan response**: Nordic ble gap advertising. USA, 2018. Disponível em: <https://devzone.nordicsemi.com/b/blog/posts/bluetooth-smart-and-the-nordics-softdevices-part-1>. Acesso em: 20 feb. 2018.

_____. **GATT Service Description**: Heart rate service description. P.O Box 436, Skoyen, Oslo, 2018. Disponível em: <https://devzone.nordicsemi.com/tutorials/b/bluetooth-low-energy/posts/ble-characteristics-a-beginners-tutorial>. Acesso em: 15 feb. 2018.

_____. **NORDIC**: Company website. P.O Box 436, Skoyen, Oslo, 2018. Disponível em: <http://www.nordicsemi.com/>. Acesso em: 19 out. 2017.

_____. **NRF Util**: Nrf util github page. P.O Box 436, Skoyen, Oslo, 2018. Disponível em: <https://github.com/NordicSemiconductor/pc-nrfutil>. Acesso em: 15 feb. 2018.

PE MICRO. **PE Micro**: Interface para bdm e jtag. USA, 2018. Disponível em: <http://www.pemicro.com/multilink/>. Acesso em: 20 jan. 2018.

PICO TECHNOLOGY. **Pico Technology Download Page**: Picoscope. USA, 2018. Disponível em: <https://www.picotech.com/downloads>. Acesso em: 23 oct. 2017.

PUTTY. **Putty Official Website**: Putty. USA, 2018. Disponível em: <https://www.putty.org/>. Acesso em: 23 oct. 2017.

RESISTOR GUIDE. **Shunt Resistor**: Shunt resistor. USA, 2018. Disponível em: <http://www.resistorguide.com/shunt-resistor/>. Acesso em: 28 feb. 2018.

SDK, N. S. **Nordic Software Development Kit**. [S.l.], 2018. Disponível em: <http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v15.0.0%2Findex.html>. Acesso em: 24 maio 2018.

SILABS. **GAP advertising**: Silabs ble gap advertising. USA, 2018. Disponível em: <https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2017/02/10/bluetooth_advertisin-hGsf>. Acesso em: 20 feb. 2018.

SILVATRONICS. **Protoboard**: Item protoboard e jumpers. São Paulo, 2018. Disponível em: <https://www.silvatronics.com.br/pd-40a8a5-protoboard-400-pontos-e-65-jumpers-macho-macho.html>. Acesso em: 20 jan. 2018.

SIM, U. N. B. **Nina System Integration Manual**. [S.l.], 2018. Disponível em: <https://www.u-blox.com/sites/default/files/NINA-B1_SIM_%28UBX-15026175%29.pdf>. Acesso em: 27 maio 2018.

SOURCEFORGE. **Realterm on Sourceforge**: Sourceforge download page for realterm. USA, 2018. Disponível em: <https://sourceforge.net/projects/realterm/>. Acesso em: 23 oct. 2017.

U-BLOX. **Nina Evaluation Kit User Guide**: Nina evk user guide. Europe, 2018. Disponível em: <https://www.u-blox.com/sites/default/files/EVK-NINA-B1_UserGuide_%28UBX-15028120%29.pdf>. Acesso em: 15 jan. 2018.

_____. **UBLOX**: Company website. Europe, 2018. Disponível em: <https://www.u-blox.com/en>. Acesso em: 16 out. 2017.

VEKTOR DIGITAL. **iBeacon Application Image**: Company website. San Francisco, 2018. Disponível em: <https://vektordigital.com/2014/01/05/real-world-ibeacon-applications/>. Acesso em: 10 jan. 2018.