INSTITUTO NACIONAL DE TELECOMUNICAÇÕES – “INATEL”

RELATÓRIO SOBRE O PROCESSO SELETIVO PARA A VAGA ESPECIALISTA I (HW/SWE)

Teste de Seleção

PARTICIPANTE:

1. Luiz Eduardo Costa Rodrigues Matrícula: 201845

Dezembro / 2020

SUMÁRIO

ÍNDICE DE FIGURAS

1. O Teste de Seleção

Você foi selecionado para fazer testes em um projeto no ICC. Este projeto tem o intuito de avisar quando a porta de um cofre é aberta e possui as seguintes características:

1 - Possui Comunicação LoRa - Cliente P2P.
2 - Uma entrada para o sensor magnético.
4 - Dois leds (verde e vermelho).
5 - Um buzzer.

Foi projetado um firmware que controla todos os periféricos descritos acima. Dentre as funções desse firmware você terá que testar as seguintes ações relacionada a violação da porta.

Cenário: Violação da porta.

Ações que o firmware deve tomar durante o cenário de violação:

1 - Estabelecer comunicação com o gateway LoRa e ficar conectado a todo momento
2 - Quando houver violação da porta, o sensor magnético é ativado e gera uma notificação que é enviado pela comunicação LoRa.
3 - Ao detectar a violação o led vermelho acende e buzzer fica tocando a todo momento.
4 - Violação resolvida led verde acende.

Sua função é testar se essas 4 funções estão funcionando de acordo. Para isso será necessário:

1 - Criar um repositório no git e colocar todas as respostas dessa atividade e enviar o link por e-mail.

2 - Descrever de forma sucinta, passo a passo, de como você fará os testes de cada ação- (formato word, seja organizado!).

3 - Escolher uma das ações que será testada e fazer uma solução para teste automatizado. Fique a vontade para usar qualquer tipo de ferramenta como Raspberry, BeagleBone, Arduino, KL25Z, MSP32, etc. (Seja criativo!)

Desejável:
Utilizar a linguagem C ou Python para resolução do teste automatizado.

2. Introdução

LoRa (Long Range) é uma tecnologia de comunicação sem fio, baseado em princípios de radiofrequência que permite comunicações em longas distâncias (na ordem de grandeza de alguns quilômetros), utilizando para isso um baixo consumo de energia elétrica. Ela utiliza frequências sub-gigahertz (abaixo de 1GHz), em bandas dedicadas de acordo com as regiões do planeta e cada uma delas possui uma frequência de operação homologada para o LoRa, sendo no Brasil a frequência de 915MHz.

Com base nesta tecnologia e cenário proposto diante do teste de seleção, foi desenvolvido uma solução em Linguagem C utilizando o Módulo ESP32 LoRa 868/915MHz com OLED 0.96″ programado na Arduino IDE.

2.1. Metodologia

O método encontrado para solucionar o teste de seleção foi implementar as lógicas do firmware capazes de fazer toda a verificação dos estados das IO's da porta do cofre nas condições citadas.

Devido ao teste permitir utilizar qualquer ferramenta. Optou-se por um microcontrolador compatível com a tecnologia LoRa e de familiaridade do participante, o Módulo ESP32 LoRa 868/915MHz com OLED 0.96″.

A IDE trabalhada e a linguagem também foi uma cujo participante já tinha conhecimento, agilizando assim, boa parte no decorrer do desenvolvimento do teste.

Como todo o firmware da porta do banco já havia sido desenvolvido, entende-se que, para a solução do teste, deve-se configurar para ser um Receptor, uma vez que receberá dados e irá apresenta-los. Porém, para maior enriquecimento do teste, acrescentou alguns incrementos para serem utilizados, caso não tenham sidos elaborados durante o desenvolvimento do mesmo.

A lógica implementada foi em função de alguns pontos chaves do teste de seleção, utilizando a linguagem C no Arduino IDE para solução da mesma.

Ao longo do presente relatório tem-se a descrição detalhada do desenvolvimento do teste, enfatizando algumas funções da biblioteca do microcontrolador utilizado com a tecnologia LoRa.

3. O Hardware

O Módulo ESP32 LoRa 868/915MHz é extremamente útil e poderoso, pois reúne numa só placa um ESP32 (portanto, possui conectividade wi-fi e Bluetooth), conectividade LoRa, GPIOs e um display OLED 0.96″. Na Figura 1 tem-se o módulo utilizado durante o desenvolvimento do teste.

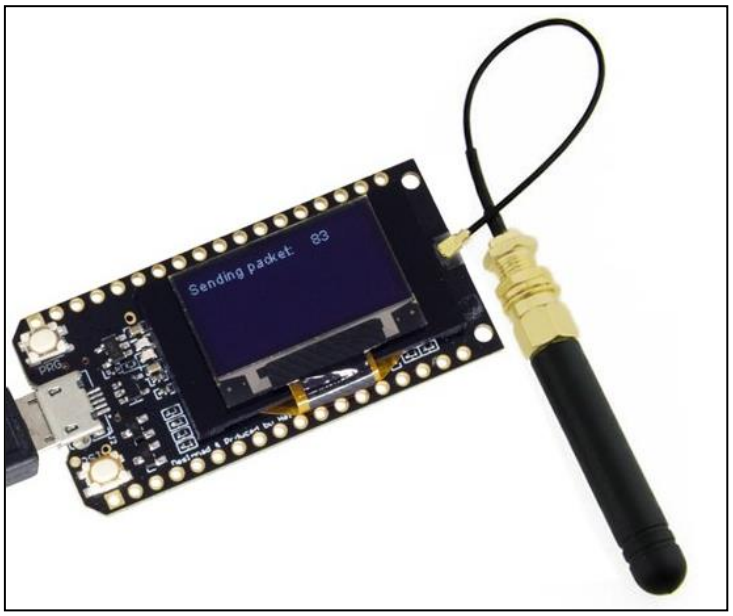

Figura 1 - *Módulo ESP32 LoRa 868/915MHz com OLED 0.96" e uma antena SMA.*

Dentre as suas principais especificações, tem-se:

– Display OLED 0.96″
– Cor do display: branca
– ESP32
– Dual-core Tensilica LX6
– Clock de até 240 MHz
– Memória interna de 520kB (SRAM)
– Módulo 802.11 b/g/n WiFi integrado
– Conexão Wifi 2.4Ghz
– Dual Bluetooth (clássico e BLE)
– Tensão de operação entre 2,2 V e 3,6 V
– Entrada Micro USB (para programação e energização)
– 36 pinos
– Conversor analógico-digital (ADC)
– Interfaces SPI, UART, I2C, PWM
– Conversor digital-analógico (DAC)
– Conector mini-JST 2 vias
– Dimensões: 52 mm x 25 mm x 6 mm

Na Figura 2 tem-se os pinos do Módulo ESP32 LoRa 868/915MHz com OLED 0.96″ e as respectivas implementações de cada um, com ênfase nas que serão utilizadas no decorrer deste desenvolvimento:

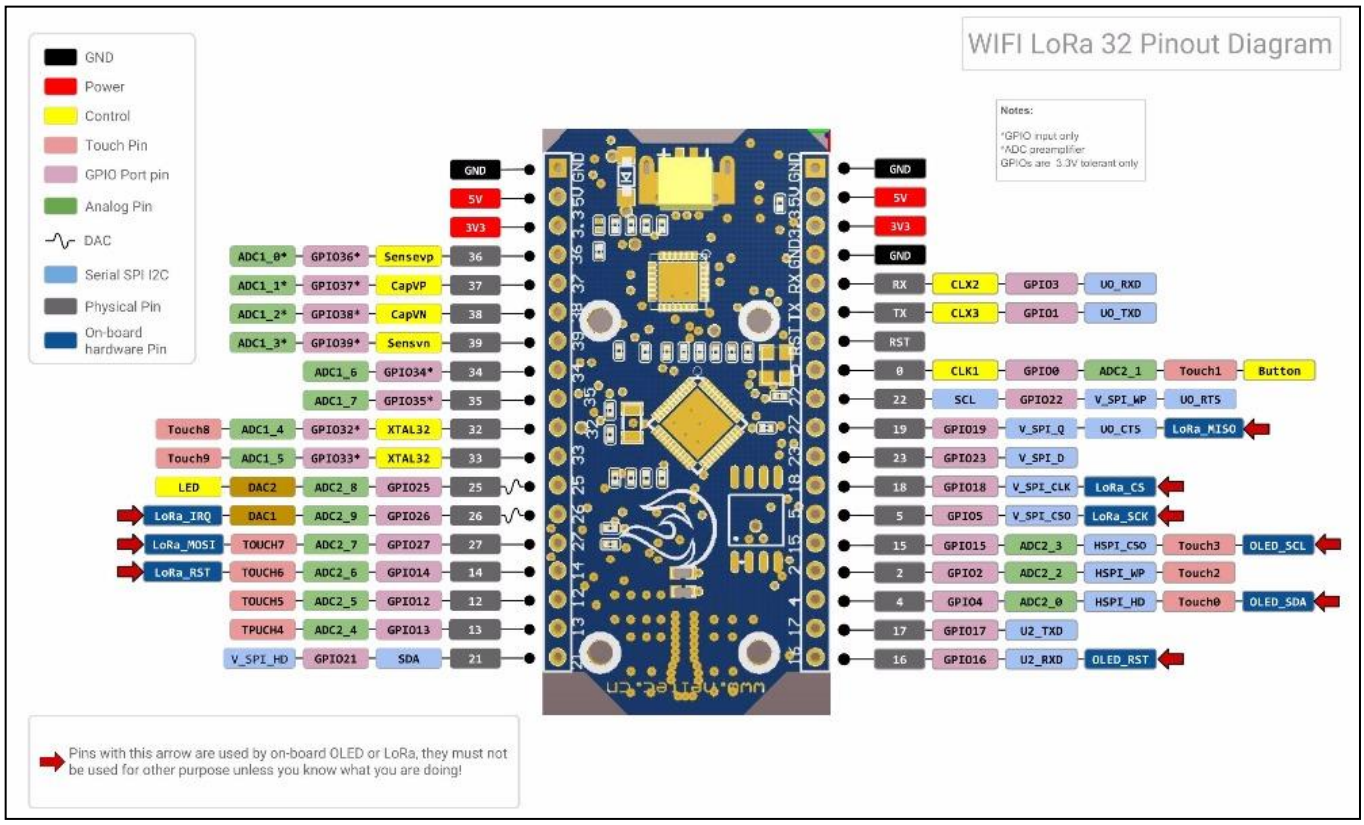

Figura 2 - *Pinos do Módulo ESP32 LoRa 868/915MHz com OLED 0.96".*

Para este teste, o material necessário é:

1 - Módulo ESP32 LoRa 868/915MHz com OLED 0.96″;

1 - Uma antena SMA;

1 – Cabo micro-USB (o mesmo servirá para programação e alimentação).

4. Configurando o ambiente de teste e OLED

Conforme a Figura 3, demonstra o a ambiente para o teste do firmware no módulo utilizado é a Arduino IDE. Este é um programa onde se faz códigos e a transparência para uma placa Arduino. Hoje em dia com a sua popularidade, há vários microcontroladores e componentes que são compatíveis com esta IDE, basta apenas adicionar as suas respectivas bibliotecas.

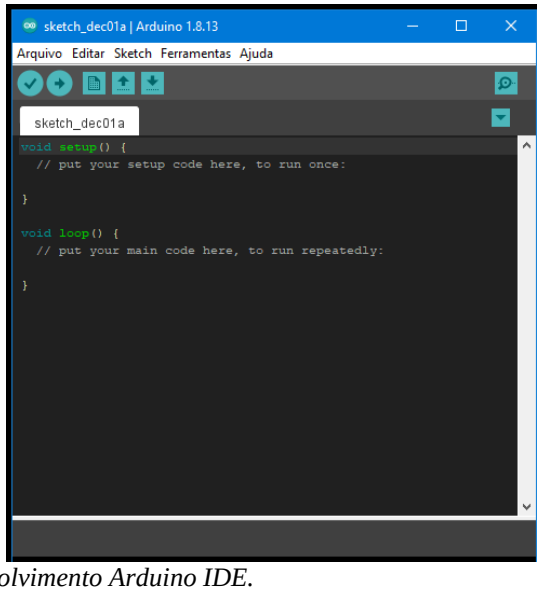

Figura 3 – *Ambiente de desenvolvimento Arduino IDE.*

Antes de fazer efetivamente o teste utilizando o Módulo ESP32 LoRa 868/915MHz, instala-se as bibliotecas mais atuais e necessárias a programação do mesmo, inicialmente a básica do ESP32, a da rádio LoRa e por fim a do display, onde são encontradas na própria Arduino IDE ou em algum site do fabricante do componente.

Logo que instaladas a biblioteca, basta inclui-las com os seguintes código:

```
#include <LoRa.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Como a biblioteca considera que o chip LoRa está ligado em pinos diferentes dos que estão ligados no módulo WiFi inclui-se no programa os seguintes defines:

```
#define SCK_LORA 5
#define MISO_LORA 19
#define MOSI_LORA 27
#define RESET_PIN_LORA 14
#define SS_PIN_LORA 18
```

Como este microcontrolador contem OLED 0.96″, faz-se bom uso desta tecnologia a favor do teste, uma vez que para ler os estados da porta de um cofre em função do sensor magnético, basta ver na própria tela do display. Para isto utiliza-se as seguintes definições:

Definições do OLED

```
#define OLED_SDA_PIN    4
#define OLED_SCL_PIN    15
#define SCREEN_WIDTH    128
#define SCREEN_HEIGHT   64
#define OLED_ADDR       0x3C
#define OLED_RESET      16
```

Offset de linhas no display OLED

```
#define OLED_LINE1      0
#define OLED_LINE2      10
#define OLED_LINE3      20
#define OLED_LINE4      30
#define OLED_LINE5      40
#define OLED_LINE6      50
```

Define-se também as seguintes constantes para o ganho e da frequência trabalhada na rádio LoRa:

```
#define GANHOLORA        20  /* dBm */
#define FREQ             915E6  /* 915MHz de frequência */
```

Para o implemento do display há variáveis e objetos globais a serem utilizados, os respectivos são:

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);.
```

Para a inicializar e verificar o display, a função abaixo verifica se há conexão com o display e caso haja, ela limpa a tela e configura os parâmetros de texto.

```
void inicializacaoDisplay(void){
   if(!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR))
   {
      Serial.println("Falha ao inicializar comunicação com OLED");
   }
   else{
      Serial.println("Comunicação com OLED inicializada com sucesso");

      /* Limpa display e configura tamanho de fonte */
      display.clearDisplay();
      display.setTextSize(1);
      display.setTextColor(WHITE);
   }
}
```

4.1. Void setup()

Todo programa de Arduino tem duas funções, sendo a primeira void setup(), executada apenas quando começa o programa, tendo como objetivos configurar os pinos da placa e estabecer a comunicação serial com um computador. A segunda é a void loop(),uma função que executa os comandos que são colocados nela infinitamente..

No void setup(),display OLED utiliza GPIOs para comunicação I²C diferentes daqueles assumidos por grande parte das bibliotecas disponíveis. Portanto, na Arduino IDE, utiliza-se a chamada “Wire.begin(OLED_SDA_PIN, OLED_SCL_PIN); ” antes da inicialização de qualquer biblioteca de display OLED.

Logo após, tem-se a execução da função mencionada anteriormente para fazer a inicialização do display. Feito isto, utiliza-se os seguintes códigos para escrever a primeira mensagem na tela para aguardar:

```
display.clearDisplay();
display.setCursor(0, OLED_LINE1);
display.print("Aguarde...");
display.display()
```

A comunicação serial e a taxa de transmissão são estabelecidas através dos seguintes comandos:

```
Serial.begin(115200);
while (!Serial);
```

5. Comunicação com o Gatway

Com seguimento no void setup() e como forma de desenvolver o teste da aplicação. Utiliza-se a execução de um comando com a seguinte chamada para verificar a comunicação com LoRa:

```
while(comunicacao_gateway_lora() == false);
```

Antes de descrever a implementação da função e toda a sua lógica, primeiro, analisa-se o teste descrito, vendo que todo o firmware da porta já está desenvolvido, para desenvolve-lo é necessário utilizar a comunicação LoRa ponto-a-ponto receptor para testar as funções do projeto.

A seguir, com toda a biblioteca LoRa instalada na IDE utilizada, agora deve-se estabelecer a comunicação com o Gatway Lora e ficar conectado a todo momento. Para isto, utiliza-se uma função dentro do while que recebe vazio e retorna um valor boleano (TRUE: comunicação OK ou FALSE: Falha na comunicação). Abaixo, tem-se toda a função implementada.

```
bool comunicacao_gateway_lora(void){
   bool status_comunicacao = false;
   Serial.println("Tentando iniciar comunicação com o rádio LoRa...");
   SPI.begin(SCK_LORA, MISO_LORA, MOSI_LORA, SS_PIN_LORA);
   LoRa.setPins(SS_PIN_LORA, RESET_PIN_LORA, LORA_DEFAULT_DIO0_PIN);

   if (!LoRa.begin(FREQ)){
      Serial.println("Tentando iniciar comunicação com o rádio LoRa, aguarde...");

      /* Escrevendo a mensagem para tentar a comunicação em 5 segundos */
      display.clearDisplay();
      display.setCursor(0, OLED_LINE1);
      display.print("Tentando iniciar comunicação com o rádio LoRa, aguarde...");
      display.display();
      delay(5000);
      status_comunicacao = false;
   }else  {
      LoRa.setTxPower(GANHOLORA);
      status_comunicacao = true;

     Serial.println("Comunicação com o rádio LoRa ok!");

      display.clearDisplay();
      display.setCursor(0, OLED_LINE1);
      display.print("Ganho da antena estabelecido: ");
      display.println(GANHOLORA);
      display.setCursor(0, OLED_LINE2);
      display.print("Comunicação com o rádio LoRa ok!");
      display.setCursor(0, OLED_LINE3);
      display.display();
     //coloca no modulo receptor continuo
      LoRa.receive();
   }

   return status_comunicacao;
}
```

No início da função comunicacao_gateway_lora, o valor da variável status_comunicacao já é inicializado como falso, apenas para uma questão de segurança para caso haja algum ruído na inicialização. Logo abaixo, conforme mencionado anteriormente, utiliza-se o código para inicializar o chip LoRa através das funções SPI.begin() e LoRa.setPins().

Na função SPI.begin() altera-se os padrões inserindo os que foram definidos anteriormente, tais como: SCK_LORA, MISO_LORA, MOSI_LORA, SS_PIN_LORA. Já na função LoRa.setPins(), altera-se o pino de seleção de escravo a ser usado, o pino de reinicialização e mantem o padrão DIO0 utilizado.

A estrutura de decisão é utilizada para verificar se a conexão foi realizada com a frequência definida anteriormente, uma vez que a do transmissor e a do receptor tenham que ser a mesma, caso não, ela fica tentando de cinco em cinco segundos até obter sucesso.

Para que isto ocorra, utiliza-se a negação da função LoRa.begin(). Esta função retorna 1 se obter sucesso ou 0 se falhar e recebe como parâmetro a frequência utilizada pela rádio, que neste teste optou por uma frequência de 915MHz, definida em uma constante anteriormente.

Quando obtém sucesso com a comunicação com a antena, implementa-se o ganho pela função LoRa.setTxPower(), onde ela recebe o valor definido anteriormente dado em dBm, coloca-se o modulo para receptor continuo através da função LoRa.receive(); e altera-se o estado da variável status_comunicacao retornando assim, o valor verdadeiro para a função comunicacao_gateway_lora.

6. Void loop()

Abaixo, segue as variáveis declaradas no void loop() que serão utilizadas no programa principal.

```
string byte_recebido;
string ultimo_byte_recebido;
int pacoteRecebido = 0;
int lora_rssi = 0;
```

A variável inteira lora_rssi é usada ao longo do código para receber a função LoRa.packetRssi() e indicar para quem está monitorando. Ela retorna o valor médio do RSSI (Received Signal Strength Indicator – indicador de força do sinal recebido) em dBm.

As demais variáveis serão explicadas nos capítulos seguintes com suas devidas funções

7. Notificação da violação

Dentro do void loop(), em primeiro, são declaradas as variáveis para a implementação do firmware desenvolvido. Dentre elas, encontra-se a variável inteira pacoteRecebido, logo após a sua declaração, ela recebe a função LoRa.parsePacket() que é responsável por checar se um pacote foi recebido. Embora é uma função que não recebe valor, ela retorna o tamanho do pacote recebido em bytes ou 0 se o pacote não foi recebido, por isto o motivo da função pacoteRecebido estar recebendo a declaração inteira.

Esta função, é utilizada em uma lógica de condição, para que quando ela entregar o seu valor 1, já imprima no display a notificação do sensor violado. Abaixo tem-se o código para a parte de violação utilizado, juntamente com o que implementa para imprimir no display a mensagem:

```
if(pacoteRecebido) {

    //Informando que está recebendo informação
    display.clearDisplay();
    display.setCursor(0, OLED_LINE1);
    display.print("RSSI: ");
    display.println(lora_rssi);
    display.setCursor(0, OLED_LINE2);
    display.print("Sensor detectado!");
    display.setCursor(0, OLED_LINE3);
    display.display();
...}
```

8. Ação ao detectar a violação

Tanto para fazer a checagem que o LED vermelho e o buzzer foram ligados durante o processo de violação, é necessário ler que o pacote foi entregue primeiro. Para isto, utiliza-se a seguinte linha de código:

```
while (LoRa.available()) {
    byte_recebido = LoRa.readString();
    ....
    }
```

Na função while, recebe-se outra função, a LoRa.available(). Essa recebe vazio e retorna o número de bytes disponíveis para leitura. Com isso, tem-se um loop infinito dentro dela enquanto está recebendo informação.

Dentro deste conjunto de funções, é feita a leitura da informação através da função LoRa.readString() que recebe vazio e retorna o próximo byte no pacote ou -1 se não tem bytes disponíveis e armazena na variável byte_recebido. A função original é LoRa.read onde pode-se acrescentar o tipo da variável que vai receber. O motivo de estar utilizando uma string, é para poder ler o que o transmissor está enviando quando for violado, onde tem-se o valor de duas variáveis o LED vermelho e o buzzer. Abaixo tem-se o código com a informação recebida para o seu monitoramento no display:

```
lora_rssi = LoRa.packetRssi();
display.clearDisplay();
display.setCursor(0, OLED_LINE1);
display.print("RSSI: ");
display.println(lora_rssi);
display.setCursor(0, OLED_LINE2);
display.print("Dados enviados: ");
display.setCursor(0, OLED_LINE3);
display.println(byte_recebido);
display.display();
```

9. Violação resolvida

Para o teste de verificação se a violação foi resolvida, tem-se a variável ultimo_byte_recebido, localizado durante a leitura da informação. Esse recebe a função LoRa.peekString(), que retorna o último dado recebido, sendo assim o último dado enviado do transmissor indicando que a violação foi resolvida.

Abaixo tem-se o código dessa lógica, onde a leitura é feita enquanto ainda está verificando a informação transmitida, logo que a porta do cofre é fechada, guarda este último comando na variável ultimo_byte_recebido e mostra no display para confirmação se a violação foi resolvida.

```
if(pacoteRecebido) {

    ...
    //verificando a informação que o tx enviou
    while (LoRa.available()) {
    …
    ultimo_byte_recebido = LoRa.peekString();
    }
...
  } else {
    /*quando a violação for resolvida,
    verifica se o led verde acendeu*/
    lora_rssi = LoRa.packetRssi();
    display.clearDisplay();
    display.setCursor(0, OLED_LINE1);
    display.print("RSSI: ");
    display.println(lora_rssi);
    display.setCursor(0, OLED_LINE2);
    display.print("Ultimos dados enviados: ");
    display.setCursor(0, OLED_LINE3);
    display.println(ultimo_byte_recebido);
    display.display();
  }
```

10. Considerações finais

Durante os estudos para o desenvolvimento do teste, notou-se que não foi mencionando sobre o pacote que o transmissor localizado na porta estaria enviando, portanto, para uma complementação do teste como um todo, seria necessário enviar para o rádio LoRa receptor os dados das variáveis que terão de serem lidas. Com base nas pesquisas, encontrou-se as seguintes funções:

```
LoRa.beginPacket();
LoRa.write();
LoRa.endPacket();
```

A função LoRa.beginPacket() inicializa a sequência de envio dos pacotes. A LoRa.write() escreve o dado para o pacote, podendo cada pacote conter até 255 bytes, nela você pode enviar um único byte ou então o dado para escrever no pacote e o seu tamanho como nos exemplos abaixo:

```
LoRa.write(byte);
LoRa.write(buffer, length);
```

Por fim, para sinalizar o termino do envio, utiliza-se a função LoRa.endPacket();

Com estas funções em mãos, considerando que o hardware foi o mesmo utilizado na solução e caso não tenha sido projetado no firmware da porta do cofre, pode-se acrescenta-las para assim, ter o desenvolvimento do teste como um todo. Abaixo segue um possível exemplo de como pode ser implementado:

```
se (sensorAtivado){

   LoRa.beginPacket();
   LoRa.write(Envia mensagem avisando que foi violado o cofre);
   LoRa.endPacket();

      Liga LEDVermelho
      LoRa.beginPacket();
      LoRa.write(Envia mensagem avisando que LED vermelho está ligado);
      LoRa.endPacket();

     Liga Buzzer
      LoRa.beginPacket();
      LoRa.write(Envia mensagem avisando que buzzer está ligado);
      LoRa.endPacket();


   }
```

```
Se não{
   Liga LEDVerde
   LoRa.beginPacket();
   LoRa.write(Envia mensagem avisando que LED verde foi ligado);
   LoRa.endPacket();
}
}
```

As demais definições, verificação com o gateway, frequência, nível da antena e outras configurações realizadas no receptor são as mesmas a serem acrescentadas, diferenciando apenas pelas funções descritas neste capítulo.

11. Demonstração pratica

Para a demonstração pratica, utilizou-se apenas os recursos básicos contidos, uma vez que não foi possível realizar a demonstração com rede LoRa.

Para a conexão com os dois ESP32 WROOM 32 utilizou-se da conexão WiFi, na qual também, todo o firmware e hardware teve que ser desenvolvido para a adaptação deste teste.

O ESP32 responsável pelo controle e envio dos dados da porta, está com sua configuração em modo AP (Acess Point), desta forma ele é configurado como server, que envia dados, e o segundo está como cliente, o qual recebe os dados.

Para a transmissão dos dados de entrada e saída, utiliza-se o mecanismo de comunicação Socket, no qual permite a troca de mensagens entre os processos de aplicação servidor e uma aplicação cliente.

Na demonstração pratica a lógica utilizada para ler os dados tanto em LoRa, quanto no modo WiFi foi a mesma, diferenciando apenas os meios de comunicação utilizados.

Para mais detalhes dos códigos utilizados e conferência do vídeo da demonstração pratica, está disponível no link: https://github.com/luiz-educosta/RI-Especialista-I-HW-SWE

12. Conclusão

Conforme foi informado em um dos quesitos do teste, poderia usar qualquer tipo de ferramenta para poder desenvolve-lo. A escolhida foi uma da família do ESP32 com radio

LoRa devido já ter um pouco mais intimidade com o mesmo e a sua IDE utilizada, porém a junção de ambas as tecnologias foi a primeira vez que se teve contato.

Para o melhor detalhamento e entendimento das funções da biblioteca LoRa instalada, foi necessário estudar um documento que vem em sua instalação, a API.md, o que também agregou para o conhecimento do participante no decorrer do desenvolvimento do teste.

Um ponto negativo foi o de que não houve compilação do código apresentado, uma vez que o mesmo foi realizado através das junções e estudos de exemplos de referências, não verificando assim, toda a lógica desenvolvida no decorrer do mesmo.

Durante o desenvolvimento do teste, houve dificuldade sobre como ler as informações da porta do cofre, uma vez também que não era de conhecimento qual o tipo de informação era transmitido caso fosse violado, com isso, optou-se por uma sugestão de lógica de implementação no item Considerações finais, uma vez que possa também agregar na documentação final e no teste como um todo.

Mesmo já conhecendo o ESP32 e já ter ouvido falar um pouco da tecnologia LoRa, o desafio proporcionou um grande aprendizado sobre os assuntos, uma vez que foi necessário estudar pelas referências encontras, as funções da biblioteca LoRa utilizada, além de também trabalhar com um display OLED, para aprimoramento e como forma de enriquecer o desenvolvimento do teste.

Para a demonstração pratica, foi necessário utilizar dos poucos recursos contidos, implementando uma configuração de transmissão diferente da proposta, abrangendo mais conhecimento de novas formas de trabalhar e de diversos assuntos.

13. O Código

Durante todo o desenvolvimento do relatório, utilizou-se os capítulos anteriores para descrever as definições, funções e toda lógica implementada, mas para melhor visualização, tem-se abaixo todo o código implementado:

```
#include <LoRa.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

/* Definições para comunicação com rádio LoRa */
#define SCK_LORA          5
#define MISO_LORA         19
#define MOSI_LORA         27
#define RESET_PIN_LORA    14
#define SS_PIN_LORA       18

/* Definições do OLED */
#define OLED_SDA_PIN    4
#define OLED_SCL_PIN    15
#define SCREEN_WIDTH    128
#define SCREEN_HEIGHT   64
#define OLED_ADDR       0x3C
#define OLED_RESET      16

/* Offset de linhas no display OLED */
#define OLED_LINE1    0
#define OLED_LINE2    10
#define OLED_LINE3    20
#define OLED_LINE4    30
#define OLED_LINE5    40
#define OLED_LINE6    50

#define GANHOLORA        20  /* dBm */
#define FREQ             915E6  /* 915MHz de frequência */

/* Variaveis e objetos globais */
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

```
void inicializacaoDisplay(void);
bool comunicacao_gateway_lora(void);

bool comunicacao_gateway_lora(void){
   bool status_comunicacao = false;
   Serial.println("Tentando iniciar comunicação com o rádio LoRa...");
   SPI.begin(SCK_LORA, MISO_LORA, MOSI_LORA, SS_PIN_LORA);
   LoRa.setPins(SS_PIN_LORA, RESET_PIN_LORA, LORA_DEFAULT_DIO0_PIN);

   if (!LoRa.begin(FREQ)){
      Serial.println("Tentando iniciar comunicação com o rádio LoRa, aguarde...");

      /* Escrevendo a mensagem para tentar a comunicação em 5 segundos */
      display.clearDisplay();
      display.setCursor(0, OLED_LINE1);
      display.print("Tentando iniciar comunicação com o rádio LoRa, aguarde...");
      display.display();
      delay(5000);
      status_comunicacao = false;
   }else {
      LoRa.setTxPower(GANHOLORA);
      status_comunicacao = true;

      Serial.println("Comunicação com o rádio LoRa ok!");

      display.clearDisplay();
      display.setCursor(0, OLED_LINE1);
      display.print("Ganho da antena estabelecido: ");
      display.println(GANHOLORA);
      display.setCursor(0, OLED_LINE2);
      display.print("Comunicação com o rádio LoRa ok!");
      display.setCursor(0, OLED_LINE3);
      display.display();

      //coloca no modulo receptor continuo
      LoRa.receive();
   }

   return status_comunicacao;
```

```
}

void inicializacaoDisplay(void){
   if(!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR)){
      Serial.println("Falha ao inicializar comunicacao com OLED");
   }else{
      Serial.println("Comunicacao com OLED inicializada com sucesso");

      /* Limpa display e configura tamanho de fonte */
      display.clearDisplay();
      display.setTextSize(1);
      display.setTextColor(WHITE);

   }
}

void setup() {

   /* Configuracao da I²C para o display OLED */
   Wire.begin(OLED_SDA_PIN, OLED_SCL_PIN);

   /* Inicializando o display */
   inicializacaoDisplay();

   /* Escrevendo a mensagem para esperar */
   display.clearDisplay();
   display.setCursor(0, OLED_LINE1);
   display.print("Aguarde...");
   display.display();

   Serial.begin(115200);
   while (!Serial);

   /* Verificando a comunicação com LoRa */
   while(comunicacao_gateway_lora() == false);

}

void loop() {
```

```
/*O pacote é a notificação que o sensor envia via LoRa*/
 string byte_recebido;
 string ultimo_byte_recebido;
 int pacoteRecebido = 0;
 int lora_rssi = 0;

 //notificando que tem informação para receber
 pacoteRecebido = LoRa.parsePacket();

 if(pacoteRecebido) {

    //Informando que esta recebendo informação
    display.clearDisplay();
    display.setCursor(0, OLED_LINE1);
    display.print("RSSI: ");
    display.println(lora_rssi);
    display.setCursor(0, OLED_LINE2);
    display.print("Sensor detectado!");
    display.setCursor(0, OLED_LINE3);
    display.display();

    //verificando a informação que o tx enviou
    while (LoRa.available()) {
    byte_recebido = LoRa.readString();
    ultimo_byte_recebido = LoRa.peek();
    }

    /* Escreve RSSI de recepção e informação recebida */
    lora_rssi = LoRa.packetRssi();
    display.clearDisplay();
    display.setCursor(0, OLED_LINE1);
    display.print("RSSI: ");
    display.println(lora_rssi);
    display.setCursor(0, OLED_LINE2);
    display.print("Dados enviados: ");
    display.setCursor(0, OLED_LINE3);
    display.println(byte_recebido);
    display.display();
 } else {
    /*quando a violação for resolvida,
```

```
        verifica se o led verde acendeu*/
        lora_rssi = LoRa.packetRssi();
        display.clearDisplay();
        display.setCursor(0, OLED_LINE1);
        display.print("RSSI: ");
        display.println(lora_rssi);
        display.setCursor(0, OLED_LINE2);
        display.print("Ultimos dados enviados: ");
        display.setCursor(0, OLED_LINE3);
        display.println(ultimo_byte_recebido);
        display.display();
    }
}
```

14. Referências

- https://www.filipeflop.com/produto/modulo-esp32-lora-868-915mhz-com-oled-0-96/
- https://www.filipeflop.com/blog/comunicacao-lora-ponto-a-ponto-com-modulos-esp32-lora/
- https://www.usinainfo.com.br/blog/esp32-lora-wifi-sx1278/
- https://www.usinainfo.com.br/lora/esp32-lora-ttgo-sx1276-868915mhz-de-longo-alcance-com-display-oled-e-bluetooth-5566.html
- https://www.robotics.org.za/ESP32-DISP
- https://www.newtoncbraga.com.br/index.php/microcontrolador/143-tecnologia/16326-moduloesp32-heltech-mec218
- http://p3r3.com/tutoriais/arduino-e-programacao-2/#:~:text=A%20void%20setup()%20%C3%A9,que%20s%C3%A3o%20colocados%20nela%20infinitamente.
- https://github.com/G6EJD/ESP32_LoRa_Examples
- https://www.fernandok.com/2018/01/esp32-controle-remoto-usando-sockets.html