INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA – IFSP
CAMPUS SÃO PAULO

YHAN CHRISTIAN SOUZA SILVA

SISTEMA IOT PARA MONITORAMENTO DE VIBRAÇÃO BASEADO EM SISTEMA EMBARCADO RTOS E REDE LORA

SÃO PAULO

2023

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA – IFSP
CAMPUS SÃO PAULO

YHAN CHRISTIAN SOUZA SILVA

SISTEMA IOT PARA MONITORAMENTO DE VIBRAÇÃO BASEADO EM SISTEMA EMBARCADO RTOS E REDE LORA

Monografia apresentada ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – Campus São Paulo – como requisito para o título de Especialista em Controle e Automação.

Orientador: Prof. Dr. Gilberto Igarashi

SÃO PAULO

2023

Catalogação na fonte
Biblioteca Francisco Montojos - IFSP Campus São Paulo
Dados fornecidos pelo(a) autor(a)

s586s Silva, Yhan Christian Souza
Sistema iot para monitoramento de vibraÇÃo baseado em sistema embarcado rtos e rede lora / Yhan Christian Souza Silva. São Paulo: [s.n.], 2023.
81 f. il.

Orientador: Prof.º Dr. Gilberto Igarashi

Monografia (Especialização em Automação e Controle) - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, IFSP, 2023.

1. Iot. 2. Lora. 3. Freertos. I. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo II. Título.

CDD 629.8

SERVIÇO PÚBLICO FEDERAL
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
DIRETORIA GERAL/CAMPUS SAO PAULO
Câmpus São Paulo, (11) 2763-7520, Rua Pedro Vicente, 625, CEP 01109-010, São Paulo (SP)

ATA DE DEFESA DE MONOGRAFIA

Na presente data realizou-se a sessão pública de defesa da Monografia intitulada **"Sistema IOT para monitoramento de vibração baseado em Sistema Embarcado RTOS e Rede LoRa"** apresentada pelo aluno **Yhan Christian Souza Silva (SP3053741)** do Curso **ESPECIALIZAÇÃO EM CONTROLE E AUTOMAÇÃO (Câmpus São Paulo)**. Os trabalhos foram iniciados às **20:00** pelo Professor presidente da banca examinadora, constituída pelos seguintes membros:

Membros	IES	Presença	Aprovação/Conceito *(quando exigido)*
Gilberto Igarashi (Orientador)	IFSP	Sim	Aprovado
Ricardo Pires (Examinador Interno)	IFSP	Sim	Aprovado
Joao Batista Brandolin (Examinador Externo)	IFSP	Sim	Aprovado

Observações:

Banca à distância, realizada no dia 23mai2023, às 20:00hs, utilizando recursos de videoconferência através da plataforma Google Meet na defesa do referido aluno de pós-graduação.

O trabalho apresentado teve bom nível técnico e atendeu plenamente seus objetivos.

A banca examinadora, tendo terminado a apresentação do conteúdo da monografia, passou à argüição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] Aprovado [ ] Reprovado Nota (quando exigido): _____

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

SÃO PAULO / SP, 23/05/2023

Gilberto Igarashi

p/ **Ricardo Pires**

p/ **Joao Batista Brandolin**

DEDICATÓRIA

Dedico este presente projeto ao Pai Celestial que permite eu estar vivo e ter forças para sempre prosseguir, a toda minha família, a minha esposa Camila Braz da Costa e minha avó Aparecida de Souza, por todo apoio durante minha trajetória.

Dedico também aos meus colegas de profissão e amigos que sempre me incentivaram na busca de conhecimento e agregam bastante na troca de experiências e discussões relevantes.

Por fim, dedico a todos os pesquisadores que dedicam seu tempo e vida em busca de conhecimento para o avanço da humanidade e tornar o mundo um lugar melhor para todos, mesmo com toda falta de incentivo existente.

AGRADECIMENTOS

Agradeço ao professor e orientador Dr. Gilberto Igarashi por todo suporte dado ao longo do desenvolvimento deste presente trabalho. Seus questionamentos e exemplos de sistemas robustos no mercado me ajudaram na busca de fazer o melhor na solução proposta, além disso, sua paciência e seu conhecimento são exemplos para todos.

Agradeço a todos os professores que ministraram conteúdo no curso de Pós-Graduação, pois todo conhecimento adquirido, direta ou indiretamente me deu a bagagem necessária para aplicar os conceitos teóricos aplicados a prática desenvolvida neste trabalho, todos profissionais com que tive aula são excelentes e sempre foram muito solícitos em minhas dúvidas durante as aulas.

Agradeço também ao professor Fernando Simplício que realiza um excelente trabalho propondo cursos realmente relevantes para o desenvolvimento de sistemas embarcados que foram muito aproveitados neste presente projeto.

E finalmente minha esposa Camila Braz da Costa que sempre esteve ao meu lado me apoiando, sendo uma verdadeira companheira.

“Deixe o futuro dizer a verdade, e avaliar cada um de acordo com seus trabalhos e suas conquistas.”

Nikola Tesla

RESUMO

A Internet das Coisas (IoT) vem revolucionando o mundo cada vez com mais dispositivos conectados, tornando o mundo mais inteligente com mais serviços e aplicações. As LPWAN (*Low Power Wide Area Network)* atendem grande parte dos requisitos de um sistema IoT devido ao baixo consumo de bateria, custo de implementação e capacidade de longo alcance, sendo umas das principais utilizadas a rede LoRa. O projeto tem como objetivo estudar e implementar uma proposta de solução utilizando a rede de comunicação LoRa através da comunicação P2P (*Point to Point)* entre dois dispositivos, sendo um responsável por iniciar a comunicação e outro enviar os dados requisitados, além disso, utiliza-se o sistema operacional de tempo real FreeRTOS e seus recursos para desenvolvimento de um sistema embarcado confiável e robusto, implementando um algoritmo de CRC(*Cyclic Redundancy Check),* além da confirmação de recebimento de dados *acknowledgement*. Também é utilizada uma ferramenta para exibição e análise dos dados, sendo possível criar alertas para valores que estejam fora dos parâmetros desejados. Testes realizados em um ambiente controlado demonstram que o sistema proposto, mostrou-se eficaz e confiável para a coleta e validação através dos algoritmos de verificação de erros e confirmação de recebimento dos dados e por fim o armazenamento em nuvem.

Palavras-chave: IoT, LoRa, FreeRTOS.

ABSTRACT

The Internet of Things (IoT) has been revolutionizing the world with more and more connected devices, making the world smarter with more services and applications. LPWAN (Low Power Wide Area Network) meet most of the requirements of an IoT system due to low battery consumption, implementation cost and long-range capacity, one of the main ones being used is the LoRa network. The project aims to study and implement a proposed solution using the LoRa communication network through P2P (Point to Point) communication between two devices, one being responsible for initiating the communication and the other sending the requested data, in addition, using the real-time operating system FreeRTOS and its resources for developing a reliable and robust embedded system, implementing a CRC (Cyclic Redundancy Check) algorithm, in addition to acknowledgment data receipt confirmation. A tool for displaying and analyzing data is also used, making it possible to create alerts for values that are outside the desired parameters. Tests carried out in a controlled environment demonstrate that the proposed system proved to be effective and reliable for data collection and validation through error checking algorithms and confirmation of receipt of data and finally cloud storage.

Keywords: IoT, LoRa, FreeRTOS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CRC	*Cyclic Redundancy Check*
CSS	*Chirp Spread Spectrum*
FFT	*Fast Fourier Transform*
FIFO	*First In, First Out*
FreeRTOS	*Free Real Time Operating System*
FSK	*Frequency Shift Keying*
GPIO	*General Purpose Input/Output*
I^2C	*Inter-Integrated Circuit*
IoT	*Internet of Things*
ISM	*Industrial, Scientific and Medical*
JSON	*Javascript Object Notation*
LoRa	*Long-Range*
LoRaWAN	*LoRa for Wide Area Networks*
LPWAN	*Low Power Wide Area Network*
MEMS	*Micro-Electromechanical System*
MQTT	*Message Queue Telemetry Transport*
P2P	*Point to Point*
QFN	*Quad-Flat No-leads*
RAM	*Random Access Memory*
RMS	*Root Mean Square*
RTOS	*Real Time Operating System*
SF	*Spread Factor*

SoC	*System on Chip*
SPI	*Serial Peripheral Interface*
Wi-Fi	*Wireless Fidelity*
M2M	*Machine to Machine*

SUMÁRIO

1. INTRODUÇÃO

O IoT (Internet of Things – Internet das Coisas) se trata de uma rede de dispositivos físicos conectados à internet. Estes dispositivos podem ser máquinas, sensores, atuadores, ou outros equipamentos compostos por interfaces de comunicação, unidades de processamento e armazenamento de dados, o que permite a troca de informações entre si, constituindo um novo paradigma tecnológico (CENTENARO et al., 2015). Estimativas apontam um grande potencial econômico que pode atingir cerca de 50 bilhões de sensores conectados em rede, e cerca de 3 a 11 trilhões de dólares de potencial de mercado em diferentes segmentos. (J. MANYIKA et al, 2015).

As LPWAN (*Low Power Wide Area Network*) são tecnologias que atendem grande parte dos requisitos de dispositivos IoT. Estas redes operam na banda ISM (*Industrial, Scientific and Medical*) e fornecem comunicação sem fio em longas distâncias com baixo consumo de energia, entretanto com baixa taxa de transmissão de dados, sendo bastante empregadas em sensores ou dispositivos que enviam pouca informação. Dentre as tecnologias existentes duas estão ganhando cada vez mais espaço: Sigfox e LoRa.

A tecnologia Sigfox é uma solução proprietária que fornece toda infraestrutura de rede necessária para a comunicação LPWAN. Ela possibilita que dispositivos finais sejam conectados à rede, desde que certas condições comerciais sejam acordadas.

A tecnologia LoRa (*Long Range*) oferece uma solução de camada física que proporciona um excelente alcance de transmissão, flexibilidade no ajuste da sensibilidade do receptor, além de ser otimizada para o baixo consumo de energia. (LORA ALLIANCE, 2018). Enquanto a camada física da tecnologia LoRa é uma solução proprietária da empresa Semtech, as demais camadas são mantidas em aberto, facilitando o desenvolvimento das mais variadas aplicações. O protocolo LoRaWAN (*LoRa for Wide Area Networks*) é um dos que mais se destaca, permitindo por exemplo a criptografia de dados e a definição de diferentes classes de implementação para um melhor gerenciamento de consumo de energia nos dispositivos. Suas diretrizes de desenvolvimento são

mantidas pela LoRa Alliance, uma associação de empresas responsáveis pela sua padronização (CENTENARO et al., 2015).

A proposta deste trabalho é desenvolver uma solução com a tecnologia de transmissão sem fio LoRa utilizando a comunicação P2P (*Point to Point*) entre dois dispositivos em uma aplicação IoT para monitoramento de dados coletados através de um acelerômetro, contemplando no sistema embarcado o RTOS (*Real Time Operating System*) FreeRTOS com objetivo de estudar seus componentes, sua implementação na programação de um sistema embarcado de 32 bits.

Como principais requisitos do sistema, foram selecionadas placas de desenvolvimento de baixo custo utilizando o SoC (*System on Chip*) ESP32, sensor acelerômetro com tecnologia MEMS (*Micro-Electromechanical System*), além disso foi desejado garantir a flexibilidade de inclusão de mais nós na rede, tornando o sistema adequado para o monitoramento de vibração de equipamentos industriais.

O presente trabalho está sob licença FreeBSD (*Berkeley Software Distribution*), onde o autor permite o uso de código-fonte, mas não tem a responsabilidade em qualquer dano causado pelo uso e/ou modificação do hardware/software (OPEN SOURCE INITIATIVE, 2018).

1.1. Objetivos

Este trabalho possui os seguintes objetivos:

- Desenvolver um sistema embarcado para aquisição de dados de um sensor e transmitir os dados a partir da tecnologia de transmissão por rádio frequência LoRa (*Long-Range*) para outro dispositivo;
- Desenvolver a comunicação ponto a ponto via LoRa com validação de dados com o uso de CRC (*Cyclic Redundancy Check*) e sistema de confirmação de dados *acknowledgement;*
- Utilizar o sistema operacional de tempo real FreeRTOS e seus recursos para desenvolvimento do sistema proposto;

- Construção de um protótipo para testes em campo do sistema proposto, comparar e discutir os resultados obtidos através dos testes em campo da comunicação ponto a ponto.

1.2. Justificativa

As justificativas para o presente trabalho são estudar a tecnologia de transmissão sem fio LoRa e suas principais aplicações (com foco no monitoramento de máquinas industriais), investigar as funcionalidades de um RTOS (neste caso o FreeRTOS) em um sistema embarcado, e explorar as potencialidades de armazenamento e tratamento de dados coletados do sistema monitorado em uma plataforma hospedada em nuvem.

O uso de um RTOS em um sistema embarcado garante maior reutilização de código, maior modularidade, melhor capacidade de manutenção e extensibilidade em comparação ao desenvolvimento de *firmwares* em *bare-metal*.

Adotar um RTOS garante melhor aproveitamento dos recursos disponíveis do hardware para o desenvolvimento com o ESP-32, utilizado neste presente projeto que possui arquitetura de 32 bits e dois núcleos de processamento.

O uso da tecnologia LoRa para monitoramento de sensores nos mais diversos segmentos já é realidade. Como por exemplo, pode-se citar o sensor WISE-2410, da empresa Advantech que coleta dados de um acelerômetro e calcula oito parâmetros de aceleração. (ADVANTECH, 2020).

O uso de um servidor em nuvem para armazenamento dos dados coletados e transmitidos pelo sistema proposto através do protocolo de comunicação MQTT (*Message Queue Telemetry Transport*), torna possível a realização do processamento de métricas através de plataforma em nuvem. Para este projeto foi escolhida a plataforma Grafana.

Portanto, o projeto proposto contempla o desenvolvimento de pesquisa técnica sobre o uso da rede LoRa, sua aplicação em um sistema embarcado programado com um RTOS, o armazenamento dos dados na nuvem e a análise de dados, sendo possível aplicar o sistema desenvolvido no monitoramento remoto de equipamentos rotativos.

1.3. Organização do trabalho

O presente trabalho foi organizado em 5 capítulos.

No capítulo 1, são apresentados o problema de pesquisa, os objetivos propostos, a relevância do tema, a motivação para o desenvolvimento deste trabalho e, por fim a organização do texto.

No capítulo 2, são apresentados trabalhos referentes ao uso da comunicação sem fio LoRa e técnicas aplicadas para a aquisição de dados e diferentes tecnologias aplicadas ao monitoramento de vibração com sensor acelerômetro.

A fundamentação teórica na qual são apresentados os tópicos e técnicas relevantes para o presente trabalho, é baseada nos sistemas operacionais de tempo real com o foco no FreeRTOS, na tecnologia de transmissão sem fio LoRa e suas regulamentações, no protocolo de comunicação MQTT e, por fim, nos protocolos I^2C e SPI utilizados para comunicação entre os módulos e a placa de desenvolvimento deste projeto.

No capítulo 3, são apresentados os materiais, as ferramentas e os métodos utilizados para aquisição, processamento e armazenamento dos dados do sistema proposto.

No capítulo 4, são apresentados os procedimentos experimentais realizados e a análise dos resultados obtidos.

No capítulo 5, são apresentadas as conclusões deste presente estudo e sugestões sobre desenvolvimento de trabalhos futuros para continuidade desta pesquisa.

2. FUNDAMENTAÇÃO TEÓRICA

Neste presente capítulo, são apresentados sistemas de sensoriamento utilizando a tecnologia LoRa e LoRaWan, bem como sistemas de monitoramento de vibração usando um acelerômetro como sensor.

Além dos trabalhos mencionados que norteiam esta pesquisa serão apresentados alguns produtos existentes no mercado, demonstrando a aplicação desta tecnologia voltada ao monitoramento de máquinas industriais.

Finalmente, são caracterizados os principais assuntos envolvidos no desenvolvimento deste projeto, tais como a utilização do sistema operacional de tempo real FreeRTOS e seus principais recursos, apresentação da tecnologia de transmissão sem fio LoRa e suas regulamentações, o protocolo de comunicação MQTT e por fim o protocolo I²C para comunicação com o módulo MPU6050 e com o display OLED de 0.96” e o protocolo SPI para comunicação com o transceptor LoRa SX1296 da Semtech.

2.1. Monitoramento de sistemas

O projeto desenvolvido por (HELAL; et al, 2018) denominado EstAcqua, tem como objetivo integrar hardware, software e sensores de baixo custo para monitoramento de informações de pressão atmosférica, umidade, temperatura externa e subaquática para estações ambientais e oceanográficas com o uso da tecnologia LoRa.

A arquitetura do dispositivo é dividida em quatro partes: um ou mais nós transmissores de dados, receptor de dados, servidor em nuvem para armazenamento dos dados coletados e diferentes interfaces com o usuário. (HELAL; et al, 2018).

Para diminuir as chances de perda de dados, o hardware empregado possui um micro SD de 32GB para armazenar os dados coletados pelos sensores aplicados na solução, ou seja, além do envio de dados para a plataforma em nuvem o hardware conta com armazenamento no dispositivo aumentando a robustez do sistema proposto.

O protótipo demonstrou sucesso em transmitir dados com taxa de perda inferior a 1% em carga útil máxima de 51 bytes para a distância de 2.8km

comportando dados de todos os sensores em um único pacote de dados. (HELAL; et al, 2018).

O projeto desenvolvido por (MEIRELES, 2018) tem como objetivo, avaliar as áreas de risco de desabamento com a implantação de uma rede de baixo consumo de energia e longo alcance, sendo realizada análise comparativa entre os resultados de cobertura do sinal em campo com simulações utilizando modelos de predição.

Uma rede com dois *gateways* e três nós sensores foi implementada na cidade de Ouro Preto, MG e a plataforma TTN (*The Things Network*) foi utilizada como servidor de rede e a plataforma Cayenne foi utilizada para apresentação dos dados coletados para o usuário final.

A dissertação desenvolvida por (FERNANDES, 2020), apresenta um protótipo de um nó sensor com interface LPWAN para monitoramento de vibração, com requisitos compatíveis com áreas de risco, mais especificamente áreas de deslizamento de terras, barragens e estruturas de engenharia, sendo um desdobramento do trabalho proposto por (MEIRELES, 2018).

Analisando os produtos disponíveis no mercado, a empresa Advantech possui um sensor, denominado WISE-2410 que permite a transmissão dos dados via LoRa ou também utilizando o protocolo LoRaWan sendo aplicado para o monitoramento de oito características de vibração (ADVANTECH, 2020):

- Velocidade RMS (*Root Mean Square*);
- Aceleração (RMS e pico);
- Deslocamento;
- Curtose;
- Fator de crista;
- Distorção;
- Desvio padrão.

Com o processador ARM Cortex-M4 este sensor consegue realizar os cálculos complexos RMS na borda e pode se comunicar diretamente com um serviço em nuvem ou transmitir os dados para um *gateway* LoRaWan.

2.2. FreeRTOS

Um sistema operacional de tempo real é um sistema operacional que garante que determinadas tarefas sejam concluídas dentro de limites de tempo determinados, ou seja, todos os prazos estipulados devem ser cumpridos, independente das circunstâncias, o que atribui aos RTOS (*Real-Time Operating System*) a característica de possuir previsibilidade (Rammig et al., 2009). Dentre as principais vantagens de se utilizar um sistema operacional de tempo real em um sistema embarcado, pode-se listar:

- Confiabilidade;
- Portabilidade;
- Reaproveitamento de código;
- Segurança.

O FreeRTOS é um sistema operacional de código aberto em tempo real para microcontroladores que facilita a programação, a implantação, a segurança, a conexão e o gerenciamento de dispositivos de borda pequenos com baixo consumo de energia. Distribuído gratuitamente sob a licença de código aberto do MIT, o FreeRTOS inclui um *kernel* e um conjunto crescente de bibliotecas de software adequadas para o uso em vários setores e aplicações. (FREERTOS.., 2021).

O FreeRTOS é ideal para aplicações em tempo real que utilizam microcontroladores ou pequenos microprocessadores. Este tipo de aplicação inclui a necessidade de requisitos de tempo real de hardware e software.

O FreeRTOS é um *kernel* de tempo real (escalonador) que permite que a aplicação desenvolvida seja organizada como um conjunto de *threads* de execução independentes, denominadas no sistema como tarefas. O escalonador decide qual tarefa deve ser executada de acordo com a prioridade atribuída a cada tarefa. A definição da prioridade da tarefa cabe ao desenvolvedor da aplicação. (BARRY, 2016).

2.2.1. Tarefas

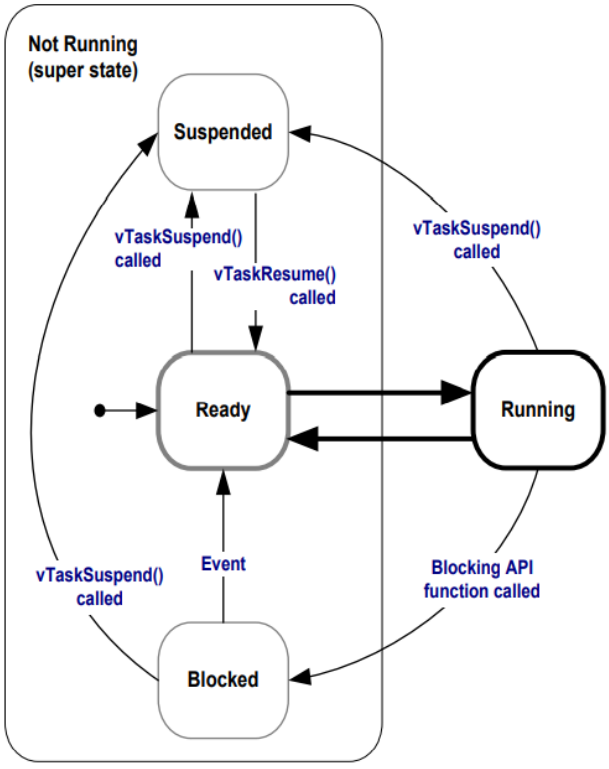
As tarefas no FreeRTOS são pequenos programas independentes dentro do sistema embarcado que são executadas após a sinalização de algum evento, sendo responsabilidade do escalonador administrar as tarefas que irão obter o uso da CPU.

Apenas a tarefas no estado *Running* estão sendo executadas no momento, enquanto as demais tarefas encontram-se no superestado *Not Running*. Neste superestado as tarefas podem estar em um dos 3 estados a seguir:

- *Ready* (Pronto);
- *Blocked* (Bloqueado);
- *Suspended* (Suspenso).

A Figura 1 tem como objetivo apresentar uma simplificação dos estados das tarefas no FreeRTOS, incluindo as divisões do superestado *Not Running*.

Figura 1- Ciclo de vida das tarefas no FreeRTOS



Fonte: (BARRY, 2016).

Uma tarefa que esteja no estado *Blocked* aguarda um evento que pode ser: temporal, no caso de tarefas periódicas como: recebimento de dados em uma fila, liberação de um semáforo, notificação de outra tarefa, ou a combinação dos itens citados acima. Como por exemplo, dados recebidos de uma fila em conjunto com a liberação de um semáforo. (BARRY, 2016).

Tarefas no estado *Suspended* não estão disponíveis para o escalonador e somente existem transições para este estado através de chamadas explícitas das funções *vTaskSuspended* e *vTaskResume*.

Geralmente transições para o estado *Suspended* são utilizadas durante o tratamento de interrupções de hardware. (BARRY, 2016).

Tarefas no estado *Ready* estão prontas para entrar em execução, entretanto ainda estão aguardando serem chamadas pelo escalonador. O escalonador sempre definirá a tarefa com maior prioridade para ser executada, ou seja, a que passará para o estado *Running*.

Quando existe mais de uma tarefa de alta prioridade, o escalonador escolherá a que está no estado *Ready* há mais tempo e caso seja configurado pelo desenvolvedor da aplicação haverá revezamento da execução das tarefas a cada *time slice* (intervalo de tempo periódico determinado com base no clock do microcontrolador). (BARRY, 2016).

Sempre deverá haver uma tarefa no estado *Running*. Para que isso ocorra o FreeRTOS cria a tarefa *Idle* quando o escalonador é inicializado. A tarefa *Idle* está sempre habilitada para execução, ou seja, não é possível bloquear ou suspender a tarefa.

Esta tarefa é configurada com a mais baixa prioridade possível, de forma que ocorra transição imediata sempre que houver outra tarefa com maior prioridade disponível para execução. Além disso, a tarefa Idle é responsável por limpar os recursos do *kernel* ocupados por uma tarefa após ela ser removida. (BARRY, 2016).

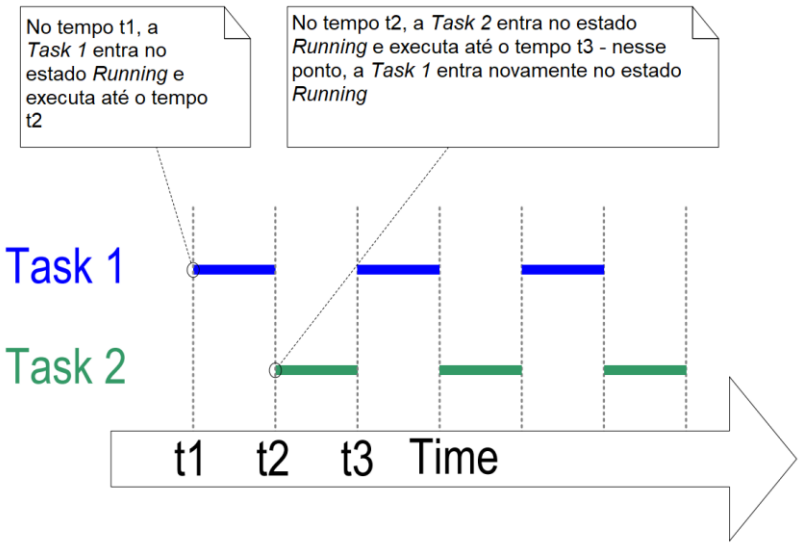
O escalonador determina qual tarefa no estado *Ready* deverá ser executada. A escolha é baseada apenas na prioridade das tarefas. Para cada

tick de clock do processador, o escalonador decide qual tarefa deverá ser executada. (BARRY, 2016).

A prioridade é definida pelo desenvolvedor no momento de criação da tarefa. O FreeRTOS não possui gerenciamento automático de prioridades. Quanto menor o valor, menor a prioridade da tarefa. A prioridade mais baixa é o valor zero (0), que é reservado para a tarefa *Idle*. O valor máximo da prioridade é definido na constante configMAX_PRIORITIES, constante contida na biblioteca FreeRTOSConfig.h. Para a tarefa de mais alta prioridade é atribuído o valor de configMAX_PRIORITIES – 1. (BARRY, 2016).

Em tarefas com a mesma prioridade, o escalonador do FreeRTOS realiza o revezamento do tempo de execução entre cada tarefa. Ele aloca a mesma fração de tempo para cada tarefa, usando como base o algoritmo *Round Robin* em conjunto com *Time Slicing,* explicados mais adiante nesta seção. Isso pode ser observado na Figura 2.

Figura 2 - Tarefas de prioridades iguais dividindo uso do CPU



Fonte: Adaptado de (BARRY, 2016).

O escalonador é o grande responsável por administrar quais tarefas serão executadas pelo sistema operacional. O algoritmo geralmente utilizado para o escalonamento de tarefas é o *Round Robin*, pois é um dos mais antigos e simples, além de ser imune a problemas de *starvation*, que se trata de tarefas que jamais são executadas por terem prioridade inferior às demais.

No FreeRTOS, a definição do tempo entre as tarefas é realizada através do *Time Slicing*, onde cada tarefa recebe uma fração mínima de tempo dividido, denominado *quantum*. Quando o tempo do *quantum* é esgotado e a tarefa ainda não liberou o uso da CPU, ocorre uma troca de contexto forçada e o escalonador decide qual será a próxima tarefa a ser executada. (BARRY, 2016).

2.2.2. Filas

No FreeRTOS, as filas são utilizadas para fornecer um mecanismo de comunicação entre processos, como entre tarefas, interrupções e vice-versa. Esse recurso tem como objetivo garantir a integridade dos dados trafegados e permite a sincronização entre processos.

Na maioria dos casos, elas são usadas como buffers FIFO (*First In, First Out*) seguros para tarefas com novos dados sendo enviados para o final da fila. (Os dados também podem ser enviados para a frente da fila.) (FREERTOS.., 2021). Para indexação dos elementos em uma fila, são utilizados dois ponteiros:

- *Head* (Início): aponta para a posição da fila que contém o elemento a ser removido;
- *Tail* (Final): aponta para a posição da fila em que novos elementos devem ser inseridos.

Há duas maneiras de inserir dados em uma fila no FreeRTOS. A primeira é copiando byte a byte o dado diretamente para dentro da fila, enquanto a segunda consiste em utilizar um ponteiro como referência para o dado, criando uma fila que armazena ponteiros para o dado. No entanto, é importante destacar que a recomendação do FreeRTOS é passar sempre os dados por cópia e não por ponteiros, para garantir a integridade dos dados e evitar problemas de compartilhamento de memória e segurança. (BARRY, 2016).

A passagem via cópia apresenta algumas vantagens como:

- É possível enviar variáveis da pilha diretamente pela fila, mesmo que a variável não exista mais na função onde foi declarada;

- Os dados podem ser enviados para uma fila sem primeiro alocar um buffer para armazenar os dados e, em seguida, copiar os dados no buffer alocado;
- A tarefa de envio pode reutilizar imediatamente a variável ou buffer que foi enviado para a fila;
- Em um sistema com proteção de memória, a RAM (*Random Access Memory*) que uma tarefa pode acessar é restrita. Se uma fila passa os dados com o uso de ponteiros, a tarefa que envia e a que recebe poderão acessar a mesma área na qual os dados foram armazenados.

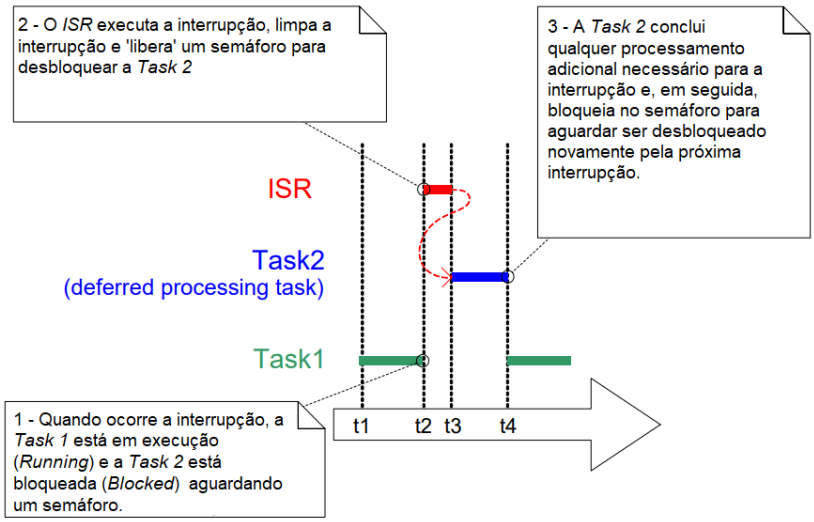
Se a quantidade de dados armazenados na fila for grande, é recomendável utilizar ponteiros para os dados em vez de copiar os próprios dados byte a byte para dentro e fora da fila. A transferência de ponteiros é mais eficiente em termos de tempo de processamento e quantidade de RAM necessária para criar a fila. No entanto, é importante ter cuidado ao enfileirar ponteiros, garantindo que o dado referenciado pelo ponteiro ainda exista na memória quando a tarefa o acessar. Caso contrário, ocorrerá uma falha de segmentação ou outra violação de memória. (BARRY, 2016).

2.2.3. Semáforos

O uso do semáforo é essencial para compartilhamento de recursos exclusivos entre tarefas distintas, como por exemplo, uma porta serial ou um GPIO (*General Purpose Input/Output*). Os semáforos trabalham de uma forma simples e que se assemelha aos semáforos de trânsito, controlando o fluxo de veículos nas vias urbanas.

A Figura 3 exemplifica a utilização de um semáforo binário, onde a *Task2* aguarda o semáforo ser liberado para iniciar seu processamento, ela está no estado *Blocked*, permitindo a execução de outras tarefas:

Figura 3 - Task2 em espera pelo semáforo



Fonte: Adaptado de (BARRY, 2016).

Os semáforos contam com duas operações básicas. Quando está liberado para uso e obtêm o mesmo (*take*) e depois de utilizar o recurso, o semáforo é liberado (*give*) para outra tarefa poder utilizá-lo, que são descritos a seguir:

- *Take*: Se o semáforo tiver um valor maior ou igual a 1, ou seja, indicando que está disponível, a tarefa poderá obtê-lo para uso, após obter o semáforo o valor dele é decrementado em 1.
- *Give*: Após execução da tarefa que obteve o semáforo, ela libera o semáforo para outras tarefas poderem utilizá-lo.

O *kernel* do FreeRTOS fornece semáforos binários, semáforos de contagem e mutexes para fins de sincronização e exclusão mútua entre tarefas. (FREERTOS.., 2021). Segue uma breve explicação dos tipos de semáforos:

- **Binário:** O semáforo mais simples, possui uma única variável que assume valores um (1) permitindo que tarefas o obtenham e (0) quando não é possível obtê-lo;
- **Mutex:** O Mutex é uma variação do semáforo binário que implementa a herança de prioridade. Ele é utilizado no FreeRTOS para minimizar o problema de inversão de prioridades, garantindo

que a tarefa que está usando o semáforo tenha a maior prioridade entre todas as outras que estão tentando obter o mesmo semáforo. Quando uma tarefa interrompe a outra para tentar obter o semáforo, o Mutex aumenta temporariamente a prioridade da tarefa que está usando o recurso para que ela possa concluir o seu uso sem ser interrompida novamente. Isso evita atrasos desnecessários e melhora o desempenho do sistema como um todo;

- **Contagem:** Conta com um vetor de valores, similar a um vetor. Este semáforo é utilizado geralmente para contabilizar eventos, a cada *give* o valor do semáforo é incrementando e a cada *take* o valor do semáforo é decrementado. O valor da contagem é a diferença entre o número de eventos que ocorreram e o número que foi processado. Esse tipo de semáforo é frequentemente utilizado no tratamento de interrupções para evitar conflitos entre interrupções e outros semáforos.

Os semáforos são uma ferramenta essencial para o desenvolvimento de sistemas multitarefa, permitindo o compartilhamento seguro de recursos entre tarefas distintas. Ao utilizar semáforos, o programador pode ter a certeza de que o acesso aos recursos está sendo gerenciado pelo próprio sistema operacional, aumentando a segurança da aplicação.

É importante ressaltar que o uso adequado de semáforos é essencial para garantir a integridade e confiabilidade do sistema em que estão sendo empregados.

2.2.4. Grupo de eventos

Os grupos de eventos são uma poderosa ferramenta disponível no FreeRTOS para facilitar a comunicação de eventos entre as tarefas, aumentando a eficiência e a segurança do sistema. (BARRY, 2016).

Ao contrário de semáforos e filas:

- Os grupos de eventos permitem que uma tarefa aguarde no estado *Blocked* por uma combinação de um ou mais eventos;

- Grupos de eventos desbloqueiam todas as tarefas que estavam esperando pelo mesmo evento, ou combinação de eventos, quando o evento ocorre.

Os grupos de eventos possuem características únicas que os tornam muito úteis na sincronização de várias tarefas, permitindo a transmissão de eventos sinalizados para mais de uma tarefa e mantendo as tarefas no estado *Blocked* por uma ou mais combinações de eventos.

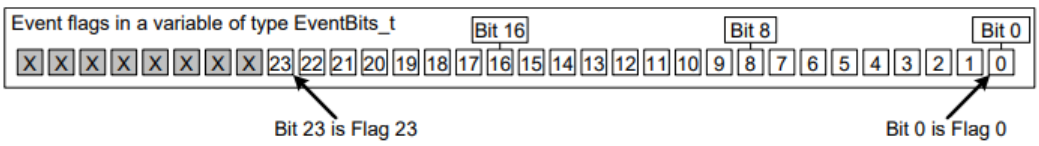
Além disso, os grupos de eventos podem ajudar a reduzir o uso de RAM em um aplicativo, substituindo muitas instâncias de semáforos binários por um único grupo de eventos. Dessa forma, os grupos de eventos são uma alternativa eficiente e econômica para a implementação de sincronização de eventos em um sistema. (BARRY, 2016).

Um evento "*flag*" é um valor booleano (0) ou (1), a fim de indicar se determinada ação ocorreu ou não, como por exemplo, a ativação de um GPIO, a leitura de um botão etc.

Um sinalizador de evento permite que o estado seja armazenado em um único bit e o estado de todos os sinalizadores de eventos em um grupo de eventos é armazenado em uma variável do tipo *EventBits_t* que pode ter 16 ou 32 bits. Por esse motivo, os sinalizadores de evento também são conhecidos como "bits" de evento. Se um bit for definido como 1 em uma variável EventBits_t, o evento representado por esse bit ocorreu. Se um bit for definido como 0 na variável *EventBits_t*, o evento representado por esse bit ainda não ocorreu. (BARRY, 2016).

A Figura 4 demonstra o mapeamento de bits em uma variável *EventBits_t*, pode-se observar que 8 bits são reservados pelo sistema operacional:

Figura 4 - Sinalizador de evento para mapeamento de bits em uma variável do tipo EventBits_t



Fonte: (BARRY, 2016).

O tamanho da variável *EventBits_t* pode ser configurado no arquivo *FreeRTOSConfig.h* alterando a definição *configUSE_16_BIT_TICKS*, sendo: (BARRY, 2016).

- Se configUSE_16_BIT_TICKS for 1, cada grupo de eventos contém 8 bits de evento utilizáveis.
- Se configUSE_16_BIT_TICKS for 0, cada grupo de eventos contém 24 bits de evento utilizáveis.

Um exemplo prático de como os grupos de eventos podem ser utilizados é na implementação da pilha TCP/IP com o uso do FreeRTOS. Ao utilizar os grupos de eventos, o projeto é simplificado e os recursos utilizados são minimizados. Isso ocorre porque os grupos de eventos permitem que várias tarefas aguardem o mesmo evento ou combinação de eventos, desbloqueando todas as tarefas quando o evento ocorre. Dessa forma, é possível sincronizar várias tarefas sem a necessidade de criar múltiplos semáforos, reduzindo a quantidade de recursos utilizados e simplificando o código.

2.3. LoRa

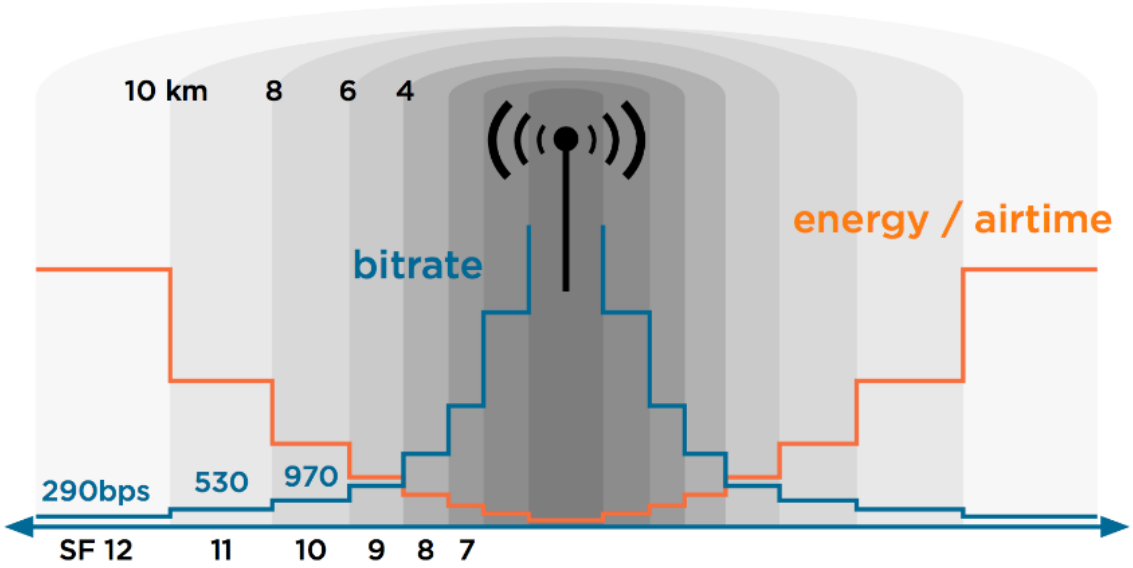
A tecnologia LoRa é uma forma de comunicação sem fio que utiliza a tecnologia de modulação no CSS (*Chirp Spread Spectrum*) para permitir uma transmissão de dados de longo alcance com baixo consumo de energia. (SEMTECH, 2015).

LoRa utiliza modulação de frequência modificada (chaveamento de mudança de frequência) que permite que os dados sejam transmitidos a distâncias da ordem de dezenas de quilômetros. Patenteada pela Semtech usada no nível físico, sendo assim, enquadra-se na primeira camada do modelo OSI, esta modulação opera na banda de frequência ISM definida pela área geográfica em que os sensores operam. (LAVRIC, 2019). Em suma, esta tecnologia está englobada na camada física do modelo OSI, sendo responsável pelos parâmetros físicos de comunicação entre dispositivos e a modulação do sinal.

A tecnologia LoRa é capaz de suportar taxas de dados variáveis, o que permite ajustar o enlace, a robustez ou o consumo de energia, mantendo uma

largura de banda constante, como pode ser visto na Figura 5 (VANGELISTA; ZANELLA; ZORZI, 2015):

Figura 5 - Alcance vs Taxa de transmissão



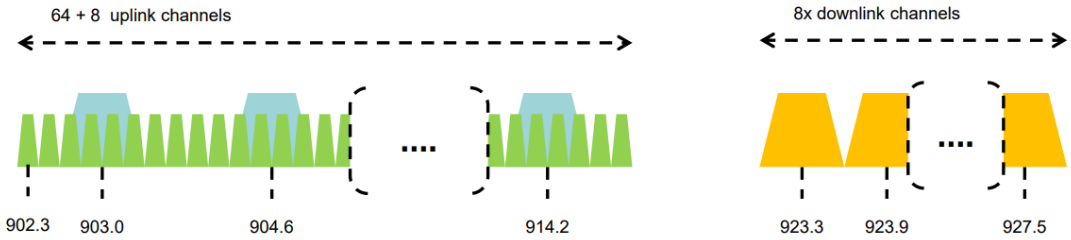
Fonte: (SHUDA; RIX; BOOYSEN, 2018).

Existe uma relação inversa entre a taxa de transmissão e o alcance em que a tecnologia LoRa pode operar. Isso significa que é possível obter um alcance maior, mas a quantidade de dados transmitidos será menor. Essa relação pode ser observada na Figura 5. No próximo tópico, serão abordados os parâmetros de SF (*Spread Factor*) *e data rate*, que são importantes para ajustar a taxa de transmissão e o alcance de acordo com as necessidades de cada aplicação.

A modulação LoRa permite comunicação a longas distâncias (em áreas urbanas 3-4km de alcance, e em áreas rurais, até 12km ou mais), com consumo mínimo de energia. (EMBARCADOS, 2016).

O LoRa foi projetado para funcionar nas bandas centradas nas frequências de 169MHz, 433MHz e 915MHz nos EUA, entretanto na Europa funciona na banda centrada em 868MHz. No Brasil a faixa de frequência está entre 915 e 928MHz, seguindo o padrão australiano (ANATEL, 2018), conforme demostrado na Figura 6:

Figura 6 - Faixa ISM de frequências



Fonte: (EMBARCADOS, 2016)

Dispositivos LoRa podem comunicar-se somente com outros dispositivos LoRa. Para a comunicação com outros dispositivos, existe o protocolo LoRaWan que nada mais é que um protocolo de comunicação da subcamada de acesso ao meio (MAC) que utiliza a camada física LoRa, entretanto o foco deste presente trabalho é somente na aplicação do rádio LoRa e não em toda arquitetura LoRaWan, por conta disso, este tópico não será abordado.

2.3.1. Parâmetros da camada física

O LoRa utiliza toda a largura de banda do canal de transmissão, tornando-se robusto a ruídos e desvios de frequência causados pelo efeito Doppler, sendo uma excelente opção para transmissões móveis, onde este efeito é relevante.

A largura de banda do LoRa pode ser programada para três valores diferentes: 125kHz, 250kHz e 500kHz. Além disso, a largura de banda pode sofrer um desvio de até 20% sem afetar a decodificação, permitindo o uso de cristais de menor precisão e, consequentemente, reduzindo o custo do sistema.

O fator de propagação no LoRa define o espalhamento espectral e estabelece uma relação entre a taxa de bits e a taxa de chirps. A especificação do LoRa define seis valores distintos de SF (SF7 a SF12), permitindo a formação de canais ortogonais que possibilitam a transmissão de vários sinais simultaneamente sem interferências (SEMTECH, 2015). No entanto, um valor maior de SF aumenta a sensibilidade do limiar de recepção em termos de potência, mas também aumenta o tempo de propagação no ar e reduz a taxa de transmissão (VANGELISTA; ZANELLA; ZORZI, 2015).

Para expressar a relação entre o SF e a taxa de transmissão, a seguir serão apresentadas algumas equações. A taxa de bits de dados Rb desejada pode ser expressa conforme a equação (1):

$$Rb = SF.\frac{1}{\left[\frac{2^{SF}}{BW}\right]}\ bits/s \tag{1}$$

Sendo o BW a largura de banda programada na aplicação LoRa, onde pode assumir três valores, conforme abordado anteriormente.

O CR (*Code Rate*) define quantos bits são utilizados para redundância da mensagem, sendo um parâmetro que varia de 1 a 4, conforme documentação do protocolo. A taxa de codificação Tc é definida conforme a equação (2).

$$Tc = \frac{4}{4 + CR} \tag{2}$$

Também é especificada uma taxa de transmissão Rb teórica, definida em função do fator de espelhamento espectral SF e da taxa de código Tc, conforme a equação (3).

$$Rb = SF.\frac{Tc}{\left[\frac{2^{SF}}{BW}\right]}\ bits/s \tag{3}$$

A Tabela 1 demonstra valores teóricos de taxa de transmissão para diferentes fatores:

Tabela 1 - Taxa teórica de transmissão (bits/s)

Largura de Banda (kHz)	SF	TC	Taxa teórica de transmissão (bits/s)
125	7	4/5	5468,75
125	9	4/5	1757,8125
125	12	4/5	292,96875
250	7	4/5	10937,5
250	9	4/5	3515,625
250	12	4/5	585,9375
500	7	4/5	21875
500	9	4/5	7031,25
500	12	4/5	1171,875

Fonte: Elaborado pelo autor.

Ao interpretar a tabela, percebe-se que quanto maior SF menor é a taxa de transmissão de *bits/s*, ou seja, é uma relação inversamente proporcional, conforme descrito no decorrer deste capítulo.

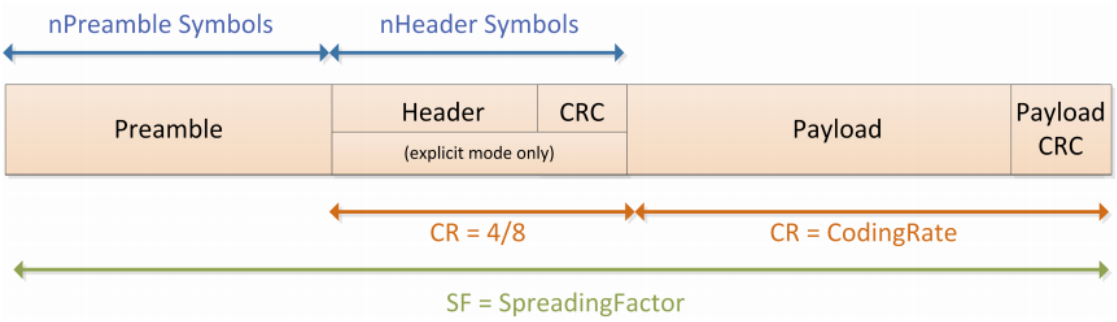
Em suma, para alcançar uma comunicação confiável e eficiente em sistemas LoRa, é essencial ajustar adequadamente os parâmetros de largura de banda (BW), fator de espalhamento (SF) e taxa de código (CR), que determinam, respectivamente, a eficiência espectral, a sensibilidade à interferência e a robustez da transmissão de dados.

2.3.2. Formato do quadro LoRa

O LoRa permite dois tipos de pacote para transmissão: explícito e implícito. O pacote explícito inclui um cabeçalho curto que contém informações sobre o número de bytes, taxa de codificação e se o CRC é usado no pacote. (SEMTECH, 2020).

O formato do quadro LoRa conforme mostrado na Figura 7, é formado por: preâmbulo, cabeçalho, carga útil e carga útil CRC.

Figura 7 - Estrutura de pacote LoRa



Fonte: (SEMTECH, 2020)

O preâmbulo é usado para sincronizar o receptor com o fluxo de dados de entrada na tecnologia LoRa. Começa com uma sequência de constantes upchirps que cobrem toda a banda de frequência. Nas últimas variações do preâmbulo é codificada a palavra de sincronização, que diferencia as redes LoRa que utilizam a mesma banda de frequência.

Em uma transmissão, os dispositivos são configurados com uma determinada palavra de sincronização. Se após a decodificação, a palavra de sincronização não corresponder a sua configuração, este dispositivo automaticamente deixará de ouvir a transmissão. O preâmbulo pode ser gerado com duração total entre 10,25 e 65539,25 símbolos. (AUGUSTIN et al., 2016).

O cabeçalho do pacote LoRa é utilizado no modo explícito de comunicação para indicar o tamanho da carga útil e limitá-lo a 255 bytes. Além disso, ele possui uma taxa de código para o final da transmissão e um CRC para descartar pacotes inválidos, a opção de utilização do cabeçalho é dada quando o receptor já conhece as informações antecipadamente, o que reduz o tempo de transmissão. No entanto, é necessário configurar manualmente o comprimento da carga, a taxa de codificação de erro e a presença do CRC da carga útil em ambos os lados do link de rádio. (AUGUSTIN et al., 2016).

Para pacotes com duração potencialmente longa em fatores de espalhamento elevados, é dada a opção de melhorar a robustez da transmissão para variações na frequência ao longo da duração da transmissão e recepção do pacote.

O uso deste recurso aumenta a robustez do link LoRa nessas baixas taxas de dados efetivas e é obrigatório quando a duração do símbolo excede 16 ms. Além disso, os rádios envolvidos na comunicação (transmissor e receptor) devem ter a mesma configuração para *LowDataRateOptimize*. (SEMTECH, 2020).

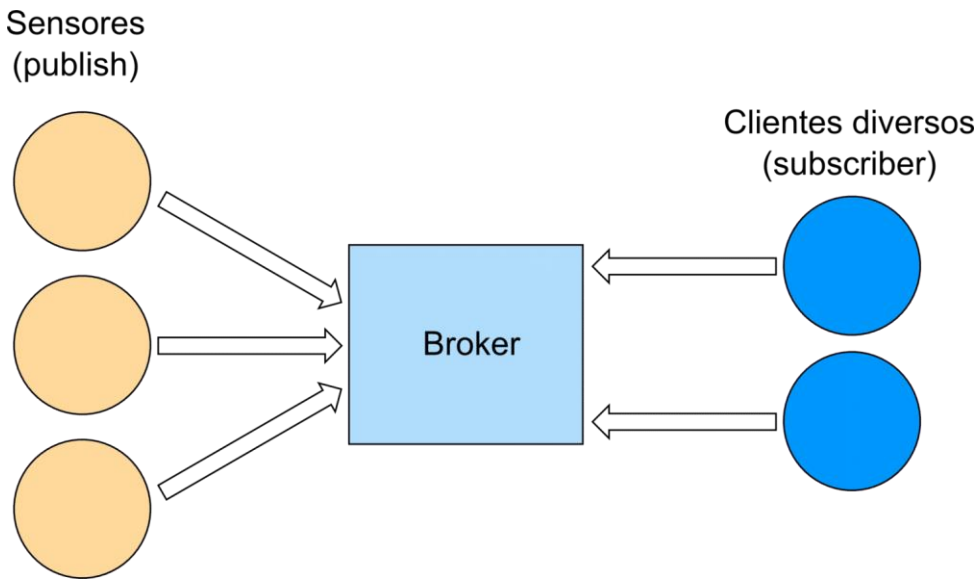
A carga útil do pacote é um campo de comprimento variável que contém os dados reais codificados na taxa de erro, conforme especificado no cabeçalho no modo explícito ou nas configurações do registro no modo implícito. Um CRC opcional pode ser anexado. (SEMTECH, 2020).

2.4. MQTT

O MQTT (*Message Queue Telemetry Transport*) é um protocolo de mensagens da camada de aplicação, projetado para baixo consumo de banda de rede e recursos de hardware, desenvolvido na década de 1990. Segundo OASIS (2014), a estrutura do protocolo MQTT utiliza o padrão de comunicação cliente-servidor, onde um ou mais clientes se conectam a um servidor, denominado de broker.

O protocolo segue o modelo de “publicar-assinar“, onde o p*ublisher* envia dados para o *Broker* e os s*ubscribers* recebem estes dados. Publishers e S*ubscribers* não interagem entre si. Eles se comunicam via Broker, tornando a comunicação assíncrona. A função do *Broker* é receber os dados, fazer a filtragem e enviá-los aos *publishers,* que estiverem registrados em determinado bloco. A estrutura na Figura 8 demonstra uma visão global do protocolo MQTT:

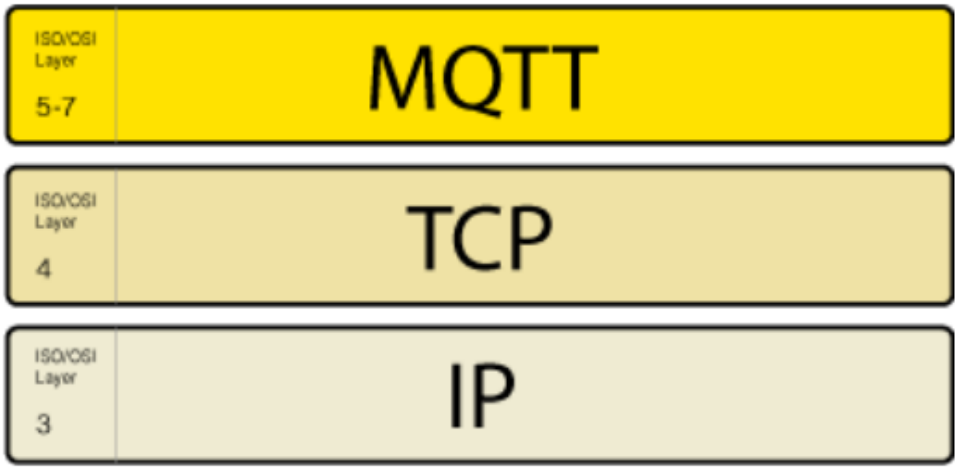
Figura 8 - Estrutura geral protocolo MQTT



Fonte: (EMBARCADOS, 2015)

O MQTT é um protocolo de transporte leve, M2M (*Machine to Machine*) otimizado para redes TCP/IP não confiáveis ou de alta latência. É um protocolo que fica nas camadas 5, 6 e 7 do modelo OSI e na camada de aplicação do modelo TCP/IP, conforme pode-se observar na Figura 9:

Figura 9 - Protocolo MQTT no modelo TCP/IP



Fonte: (HIVEMQ, 2022)

Devido a simplicidade de implementação e por ser amplamente utilizado em projetos de IoT, o protocolo MQTT foi selecionado para o projeto proposto nesta monografia.

2.5. SPI

O protocolo de comunicação SPI (*Serial Peripheral Interface*) trata-se de uma comunicação serial síncrona para pequenas distâncias desenvolvido pela Motorola na década de 1970. (LEENS, 2009).

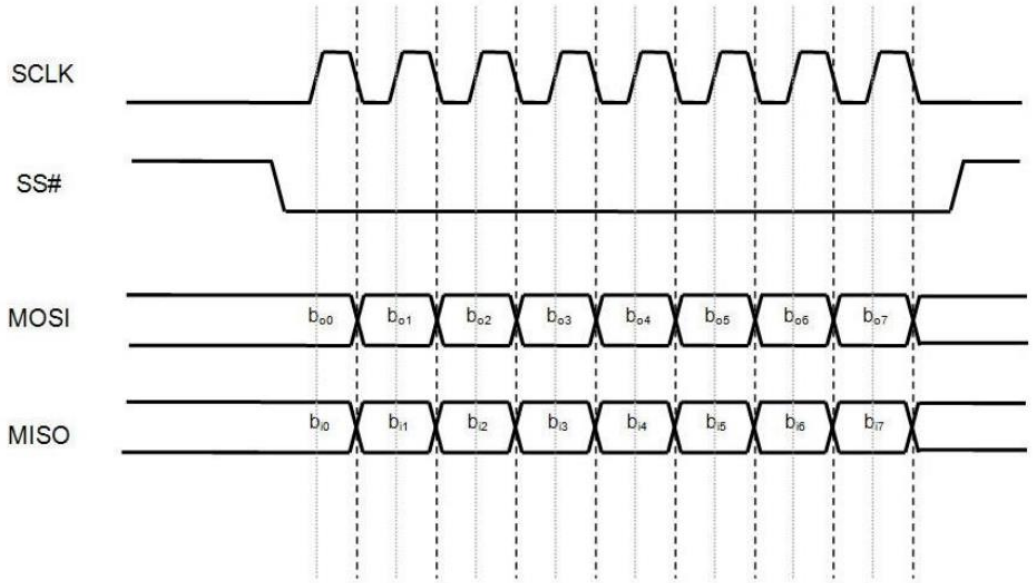
O protocolo SPI utiliza quatro pinos de sinal para realizar a conexão entre um dispositivo mestre e um escravo, sendo:

- **SCLK:** (*Serial clock*), os sinais do barramento são sincronizados com este sinal;
- **SS:** (*Slave Selection*), seleciona o dispositivo escravo com que o mestre deseja realizar a comunicação;
- **MOSI:** (*Master-Out Slave-In*), linha de dados do mestre para o escravo;
- **MISO:** (*Master-In Slave-Out*), linha de dados do escravo para o mestre.

O barramento SPI permite a utilização de apenas um único mestre, mas não limita o número de escravos a ele conectados. Por haver somente um único dispositivo mestre no barramento qualquer comunicação é iniciada por ele.

Quando um dispositivo mestre deseja enviar dados e/ou receber dados de um dispositivo escravo, é selecionado o dispositivo através do terminal SS, é ativado o *clock* e os dados são enviados através da linha MOSI, enquanto os dados retornados pelo escravo são na linha MISO. A Figura 10 demonstra um diagrama de comunicação SPI.

Figura 10 - Diagrama de comunicação SPI



Fonte: (LEENS, 2009)

O protocolo SPI permite que os dispositivos enviem e recebam dados simultaneamente, portanto trata-se de um protocolo *full-duplex*. Além disso, não há definição de taxa máxima de transferência de dados, também não há mecanismo de *acknowledgement* para confirmar os dados recebidos e não há controle de fluxo. Por fim, o protocolo SPI não descreve as características físicas como nível de tensão ou uma norma para padronizar o meio entre os dispositivos. (LEENS, 2009).

2.6. I²C

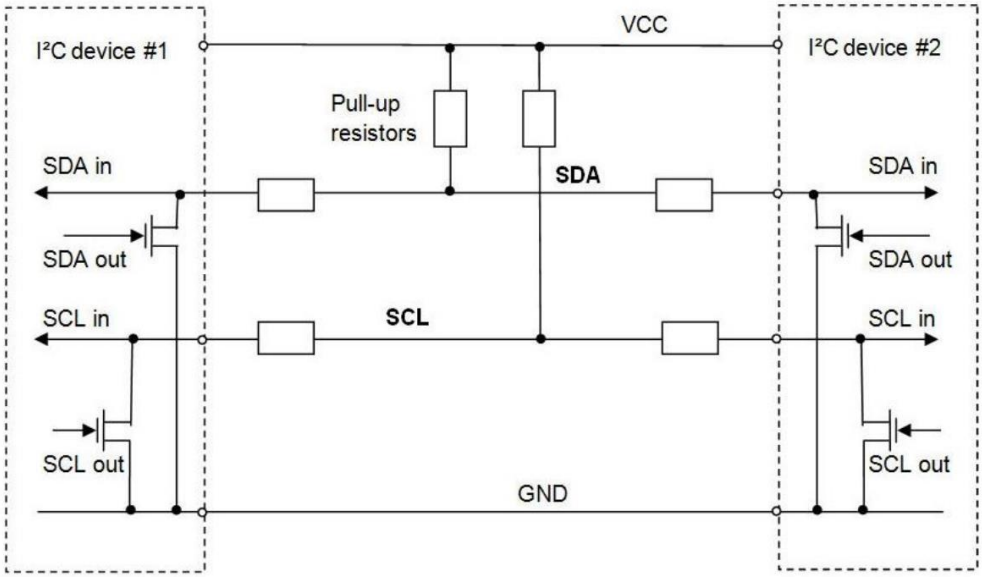
O protocolo de comunicação I2C (*Inter-Integrated Circuit*) foi desenvolvido pela Philips na década de 1990. Ele é muito utilizado para conectar periféricos de baixa velocidade a placas-mãe, microcontroladores etc.

O barramento I2C é composto apenas por dois sinais, SCL (*Serial Clock*) que transporta o *clock* e SDA (*Serial Data*) que transporta os dados em ambos os sentidos, caracterizando assim o protocolo como *half-duplex*.

Como ambas as saídas SCL e SDA são bidirecionais, os dispositivos I2C necessitam utilizar saídas do tipo dreno aberto, além de cada linha necessitar de um resistor de *pull-up* externo. (LEENS, 2009).

O protocolo I2C suporta múltiplos dispositivos mestre e múltiplos dispositivos escravos no mesmo barramento. A conexão física de um dispositivo I2C pode ser visualizada conforme a Figura 11:

Figura 11 - Conexão de dispositivos a um barramento I2C



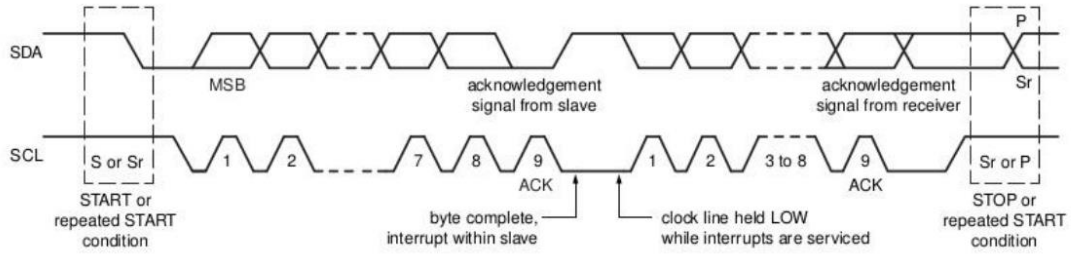
Fonte: (LEENS, 2009)

A comunicação é sempre iniciada pelo dispositivo mestre controlando a linha de *clock* e cada escravo possui um endereço físico único no barramento, podendo ter a presença de até 128 escravos endereçados. (LEENS, 2009).

O protocolo I2C prevê alguns sinais de controle utilizados antes, durante e depois de qualquer transação. Antes do dispositivo mestre iniciar uma comunicação ele deve verificar se o barramento está disponível, após confirmar que o barramento está disponível o mestre envia um sinal de controle denominado *start bit* que indica aos demais dispositivos que a palavra seguinte contém o endereço do dispositivo escravo.

O bit menos significativo do byte que contém o endereço do dispositivo I2C sinaliza o tipo de transação solicitada pelo mestre. A Figura 12 mostra o diagrama de uma transferência de dados via barramento I2C.

Figura 12 - Transferência de dados via barramento I2C



Fonte: (NXP SEMICONDUCTORS, 2021)

A comunicação entre dispositivos I2C é orientada a palavras de oito bits, para cada palavra de 8 bits transmitida um sinal de confirmação, denominado *acknowledgement bit* deverá ser retornado pelo dispositivo receptor.

Os dados que trafegam na linha de SDA sempre são amostrados na borda de subida da linha de *clock* SCL e durante a transmissão de dados a linha SDA somente é atualizada quando a linha de *clock* estiver em nível lógico zero. Os sinais de controle não respeitam estas regras, porém só podem ser emitidos por dispositivos mestres no barramento.

3. MATERIAIS E MÉTODOS

Neste capítulo, serão descritos os materiais e ferramentas empregados no desenvolvimento do protótipo do módulo transmissor e receptor, além da metodologia adotada para coletar os dados do sensor MPU6050, transmiti-los via LoRa, processá-los e publicá-los no broker MQTT em um servidor em nuvem. Por fim, também será abordada a exibição dos dados e os cálculos de estatísticas por meio do Grafana.

3.1. Requisitos do sistema

O projeto desta monografia envolve dois dispositivos: um módulo transmissor, que atua como *Gateway*, solicitando ao módulo receptor o envio dos dados do sensor MPU6050. Após receber esses dados por meio da tecnologia LoRa, o módulo transmissor os processa, valida e publica em um broker MQTT. Por fim, os dados são armazenados em um servidor na nuvem. O diagrama na Figura 13 demonstra a arquitetura do projeto proposto:

Figura 13 - Arquitetura proposta do projeto

Fonte: Elaborado pelo autor.

Para fazer o uso do LoRa com confiabilidade, foram utilizados algoritmos disponibilizados pela escola Microgênios e adaptados à realidade deste projeto, ele foi inspirado em alguns dos padrões utilizados no protocolo Modbus RTU. O protótipo desenvolvido conta com os seguintes requisitos:

- Sistema transmissor-receptor, ou seja, o receptor só enviará dados se for requisitado;
- Utilização de algoritmo CRC para validação dos pacotes;
- Sistema de confirmação de dados *acknowledgement*;
- Utilização dos recursos do FreeRTOS no firmware desenvolvido.

3.2. Materiais e recursos utilizados

Os materiais utilizados para desenvolvimento do trabalho proposto são de acordo com a Tabela 2.

Tabela 2 - Relação de materiais e custo estimado

Item	Descrição	Qtde	Valor Unitário	Valor Total
1	Placa WiFi LoRa 32 - ESP32 / LoRa / Display OLED	2	R$ 189,00	R$ 378,00
2	Acelerômetro e Giroscópio 3 Eixos 6 DOF - MPU6050	1	R$ 17,90	R$ 17,90
3	Caixa armazenamento - Impressão 3D PLA 1,75mm	2	R$ 25,95	R$ 51,89
4	Carregador portátil de bateria	2	R$ 49,90	R$ 99,80
		TOTAL:		R$ 547,59

Fonte: Elaborado pelo autor.

A Tabela 2 demonstrou a lista de materiais utilizados para a construção dos protótipos do módulo transmissor e do módulo receptor e os custos envolvidos, além disso foram empregados os seguintes recursos:

- **Servidor em Nuvem:** uso do Amazon AWS para envio dos dados coletados pelo dispositivo transmissor através do protocolo de comunicação MQTT;
- **Análise de Dados**: O Grafana que se trata de uma solução de código aberto que permite executar análise de dados, sendo

possível monitorar grande quantidade de informações e obter métricas.

3.3. Módulo LoRa Heltec ESP-32

O módulo Wi-Fi LoRa ESP32 V2 da Heltec é uma solução de desenvolvimento para projetos de IoT que oferece três formas de comunicação sem fio: Wi-Fi, Bluetooth e LoRa. O módulo utiliza o SoC ESP32 e se comunica com o transceptor LoRa SX1276 via SPI. Além disso, ele possui uma memória flash W25Q32FV, um conversor USB-Serial CP2102, um controlador de bateria MCP73831 e um display OLED de 0,96 polegadas, como mostrado na Figura 14: (HELTEC, 2019).

Figura 14 - Visualização Placa: Módulo Heltec ESP-32

Fonte: (HELTEC, 2019).

A Figura 15 apresenta as principais especificações técnicas do kit de desenvolvimento.

Figura 15 - Especificações Técnicas: Módulo Heltec ESP32

Resource	Parameter		
Master Chip	ESP32(240MHz Tensilica LX6 dual-core + 1 ULP, 600 DMIPS)		
Wireless Communication	Wi-Fi	Bluetooth	LoRa
	802.11 b/g/n (802.11n up to 150 Mbps)	Bluetooth V4.2 BR/EDR and Bluetooth LE specification	Node-to-node communication or LoRaWAN
LoRa Chip	SX1276/SX1278		
LoRaWAN Area	hardware version	Support frequency	
	LF	EU433	
		CN470	
	HF	IN865	
		EU868	
		US915	
		AU915	
		KR920	
		AS923	
LoRa Maximum Output Power	19dB ± 1dB		
Hardware Resource	UART x 3; SPI x 2; I2C x 2; I2S x 1; 12-bits ADC input x 18; 8-bits DAC output x 2; GPIO x 22, GPI x 6		
FLASH	8MB(64M-bits) SPI FLASH		
RAM	520KB internal SRAM		
Interface	Micro USB x 1; LoRa Antenna interface(IPEX) x 1; 18 x 2.54 pin x 2		
Maximum Size (Including protruding parts such as switch and battery compartment)	51 x 25.5 x 10.6 mm		
USB to Serial Chip	CP2102		
Battery	3.7V Lithium (SH1.25 x 2 socket)		
Solar Energy	x		
Battery Detection Circuit	√		
External Device Power Control (Vext)	√		
Low Power	Deep Sleep 800μA		
Display Size	0.96-inch OLED		
Working Temperature	-40~80℃		

Fonte: (HELTEC, 2019).

A Figura 16 demonstra as características elétricas de consumo estimado pelo módulo:

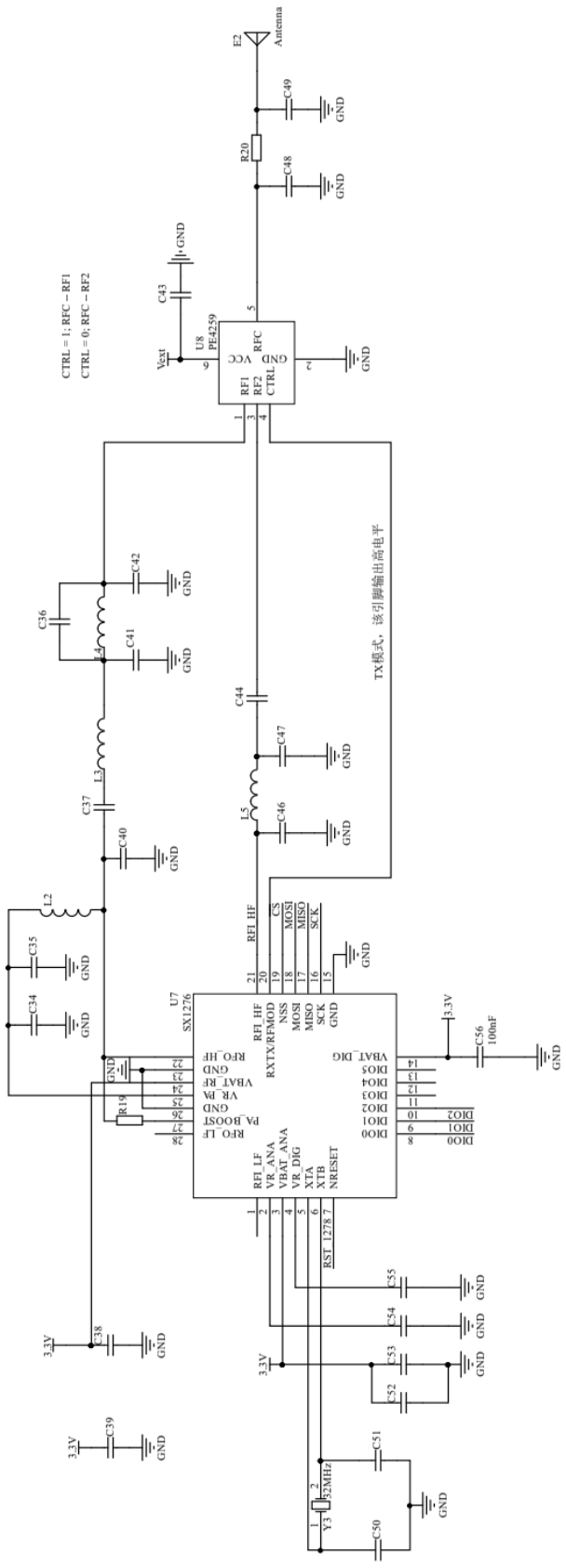
Figura 16 - Características Elétricas: Módulo Heltec ESP32

Electrical Features	Condition	Minimum	Typica	Maximum
Power Supply	USB powered (≥500mA)	4.7V	5V	6V
	Lithium powered (≥250mA)	3.3V	3.7V	4.2V
	3.3V (pin) powered (≥150mA)	2.7V	3.3V	3.5V
	5V (pin) powered (≥500mA)	4.7V	5V	6V
Power Consumption(mA)	WIFI Scan		115mA	
	WIFI AP		135mA	
	LoRa 10dB output		50mA	
	LoRa 12dB output		60mA	
	LoRa 15dB output		110mA	
	LoRa 20dB output		130mA	
Output	3.3V pin output			500mA
	5V pin output (USB powered only)		Equal to the input current	
	External device power control (Vext 3.3V)			350mA

Fonte: (HELTEC, 2019).

Este módulo possui alguns pinos que utilizam a comunicação SPI para comunicação com o chip LoRa SX1276, conforme demonstra diagrama esquemático na Figura 17:

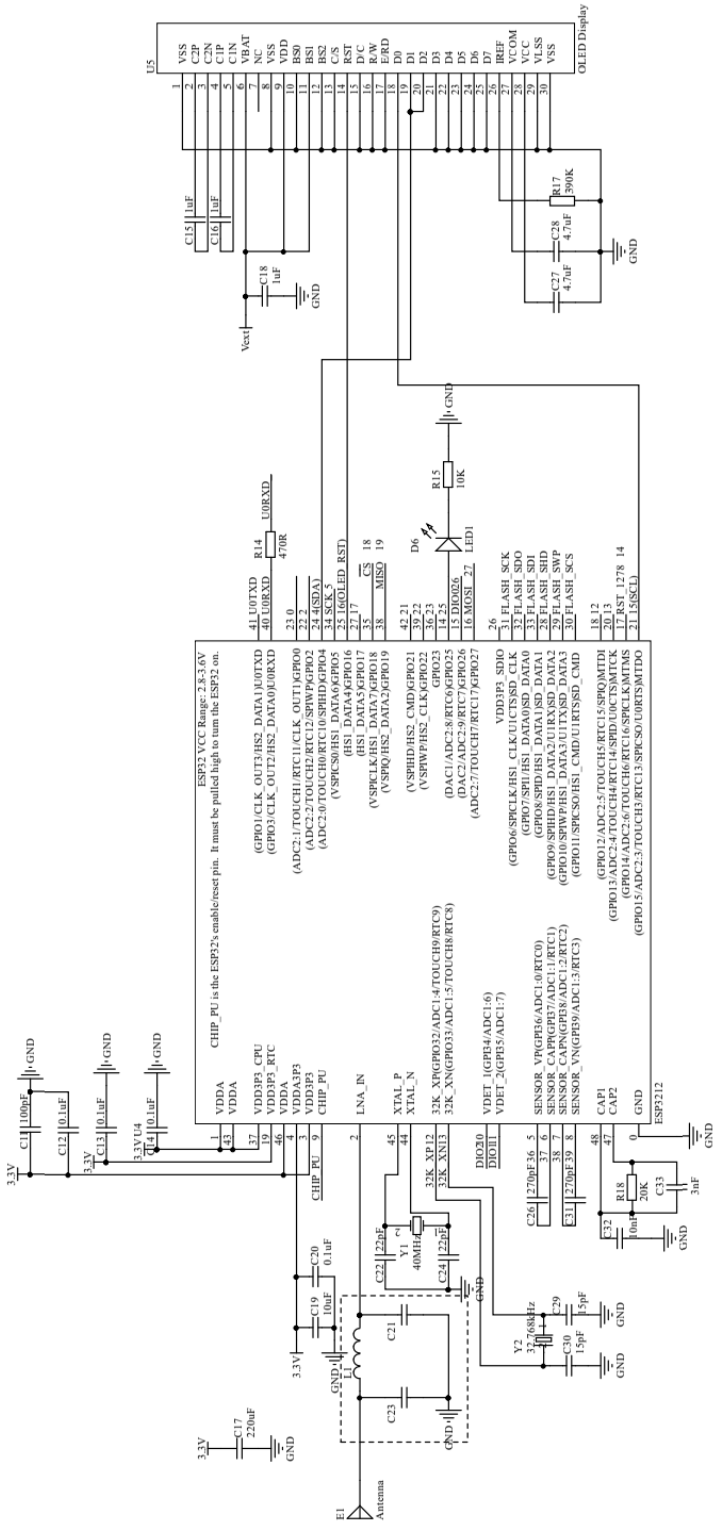
Figura 17 - SX1276: Conexão SPI ao ESP32



Fonte: (HELTEC, 2019).

Além disso, o módulo Wi-Fi LoRa ESP32 V2 da Heltec utiliza uma das interfaces I²C para comunicação do display OLED de 0,96 polegadas, pode-se observar no esquemático, de acordo com a Figura 18:

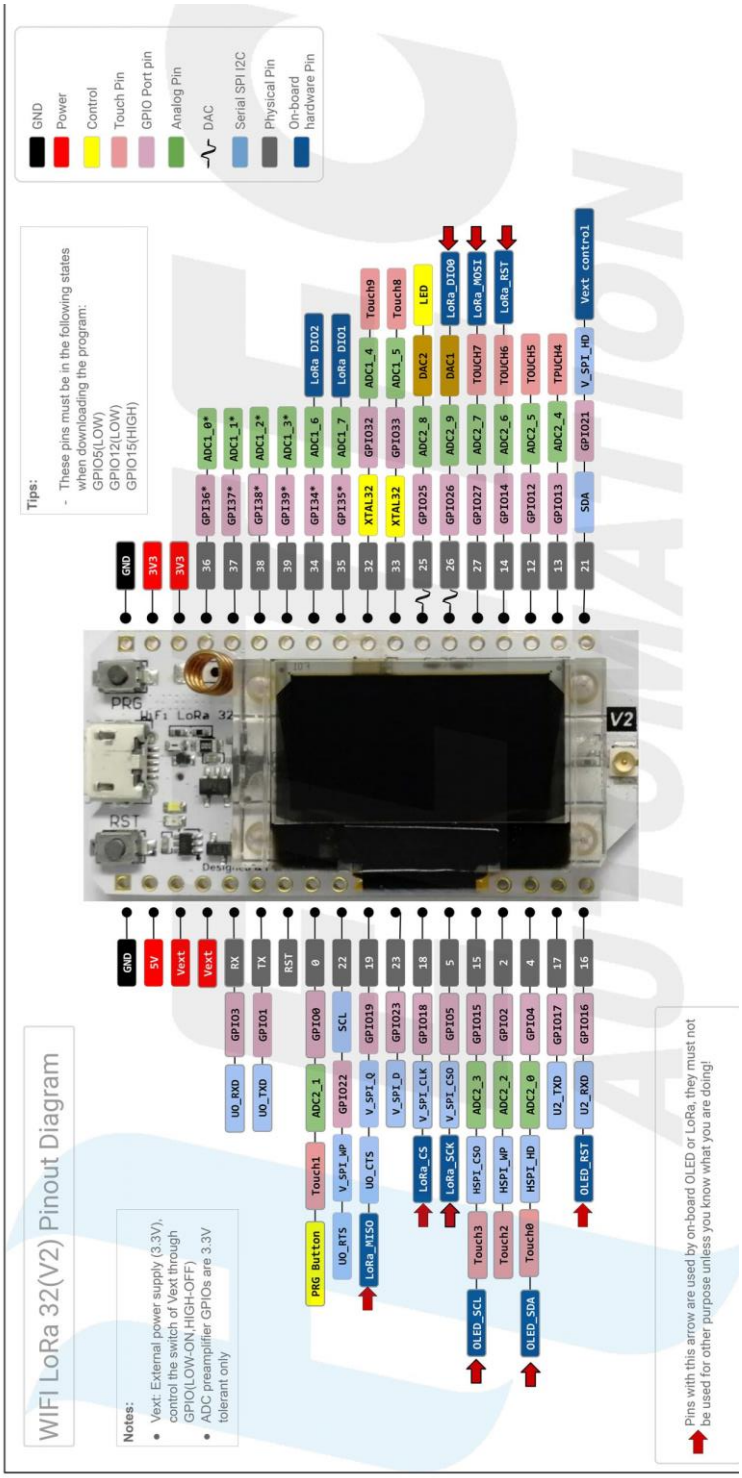
Figura 18 - Display OLED: Conexão I²C ao ESP32



Fonte: (HELTEC, 2019).

A fim de evitar erros de conexão, os pinos destinados a funções específicas são destacados na pinagem do módulo com uma flecha vermelha e há uma nota sugerindo que estes pinos não sejam utilizados para outras finalidades, conforme Figura 19.

Figura 19 – Diagrama Pinagem: Módulo Heltec ESP32



Fonte: (HELTEC, 2019).

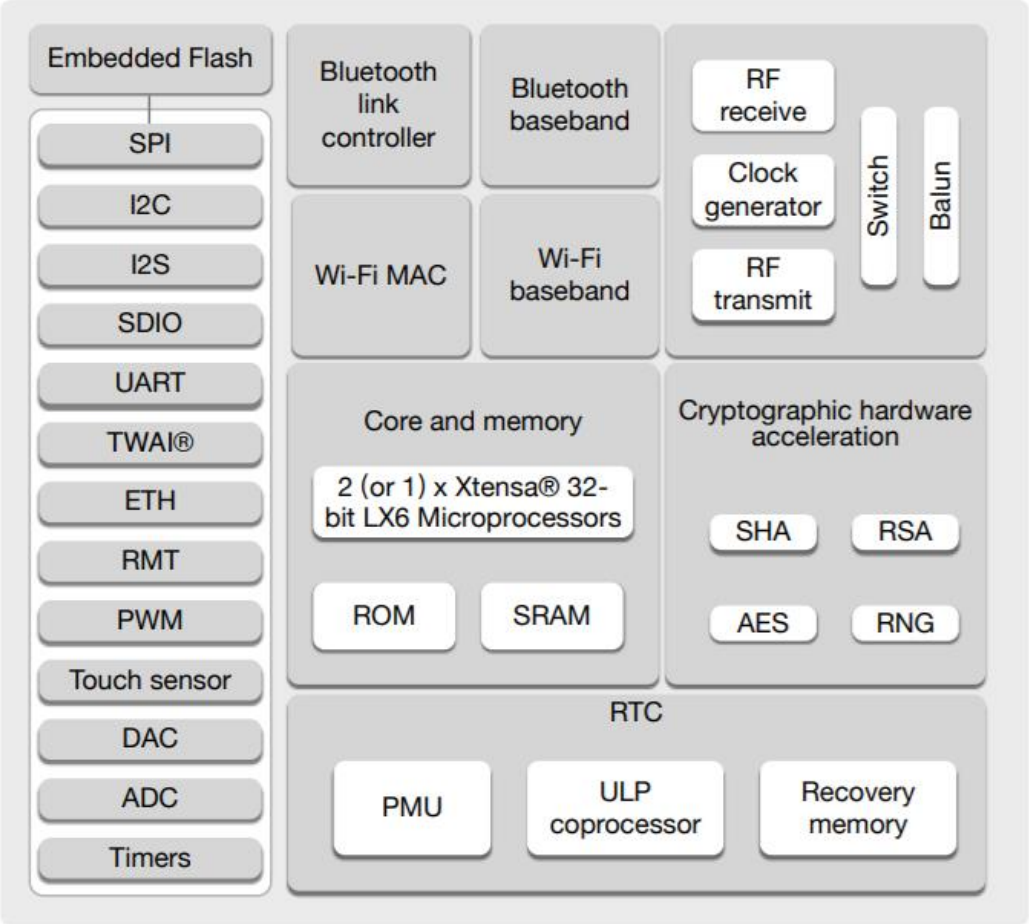
3.3.1. ESP-32

O ESP32 é um chip fabricado pela empresa Espressif Systems que possui alta performance, baixo consumo de energia, conexão Wireless Fidelity (Wi-Fi) padrão 802.11 b/g/n e Bluetooth.

O ESP32 chip possui 4 MB de memória flash, CPU dual-core, 520 Kbytes de SRAM, 34 pinos de GPIO, dos quais 18 podem ser utilizados como conversor analógico digital de 12 bits e possui tensão de operação é de 3.3V. (ESPRESSIF SYSTEMS, 2021).

A Figura 20 demonstra o diagrama de blocos do ESP32:

Figura 20 - Diagrama de blocos ESP32



Fonte: (ESPRESSIF SYSTEMS, 2021).

O ESP32 é um SoC encapsulado com as dimensões de 6 × 6 e 5 x 5 mm de tamanho em encapsulamentos tipo QFN (*Quad-Flat No-leads*), reunindo

bastantes recursos tornando-o uma opção de baixo custo e bastante interessante para projetos de IoT. Suas principais características são descritas a seguir:

- **CPU e memória:**
  - Processador principal: Microprocessador Tensilica Xtensa 32-bit LX6 com 2 núcleos;
  - Frequência de clock: 240MHz;
  - Performance: até 600 DMIPS[1];
  - Coprocessador ULP (*Ultra Low Power*): utilizado para algumas tarefas visando baixo consumo;
  - ROM: 448Kbytes;
  - SRAM: 520Kbytes;
  - RTC: 16Kbytes;
  - Suporte a até 16 MB de memória externa.
- **Conectividade sem fio:**
  - Wi-Fi: 802.11 b/g/n/e/i (802.11n @ 2.4 GHz até 150 Mbit/s);
  - Bluetooth: v4.2 BR/EDR e Bluetooth Low Energy (BLE).
- **Periféricos:**
  - 34 pinos de GPIO;
  - 18 canais de ADC com resolução de até 12bits;
  - 2 canais de DAC de 8 bits;
  - 10 GPIOs com suporte a touch capacitivo;
  - 2 interfaces I²C;
  - 2 interfaces UART;
  - Controlador CAN;
  - 4 interfaces SPI;
  - 2 interfaces I²S;
  - 16 canais de PWM.
- **Segurança e criptografia:**
  - Boot seguro;
  - Criptografia de flash;

[1] DMIPS significa "Dhrystone MIPS" trata-se de um padrão para comparar o desempenho de diferentes microcontroladores / microprocessadores em diferentes conjuntos de instruções.

- Aceleração de criptografia em hardware usando: AES, SHA-2, RSA, ECC e RNG.

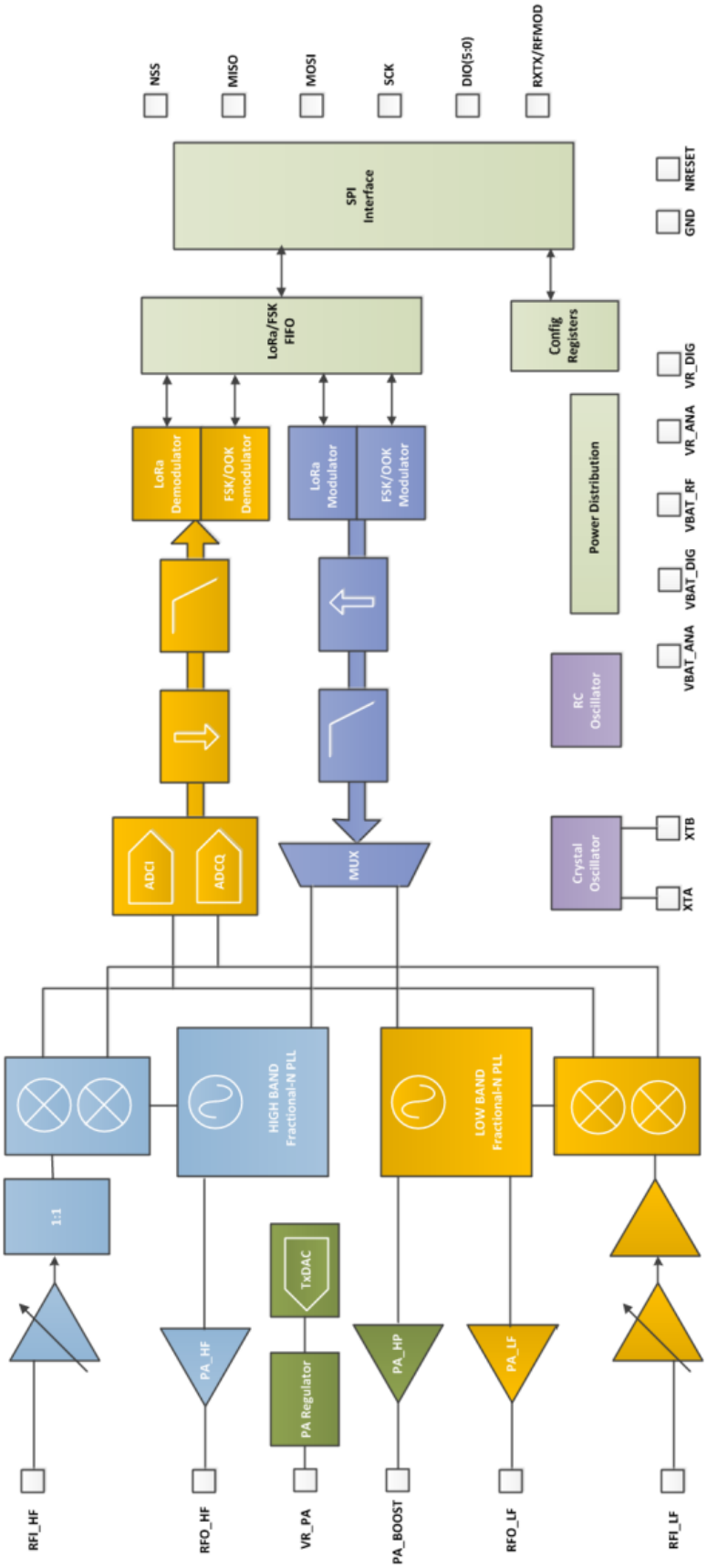
Conforme abordado no datasheet do chip fabricado pela Espressif Systems, este é um dispositivo destinado a diversas aplicações em IoT, desde cidades inteligentes até automação industrial. (ESPRESSIF SYSTEMS, 2021).

3.3.2. LoRa SX1276

O transceptor LoRa SX1276, desenvolvido pela Semtech, é um dispositivo half-duplex que oferece comunicação de espectro com propagação de alcance ultralongo e alta imunidade a interferências, além de minimizar o consumo de corrente. Usando a técnica de modulação LoRa patenteada pela empresa, o transceptor SX1276 pode atingir uma sensibilidade de recepção de até -148dBm, combinada com um amplificador de potência de +20dBm para transmissão integrado, tornando-se extremamente útil para aplicações que exigem alcance e robustez. A tecnologia LoRa apresenta vantagens significativas em relação às técnicas de modulação convencionais, proporcionando baixo consumo de energia e imunidade contra interferências. (SEMTECH, 2020).

Além disso o chip SX1276 permite o envio de pacotes de até 256 bytes com verificação de redundância cíclica (CRC) e possui taxa de bits programável de até 300kbps. A Figura 21 demonstra o diagrama do controlador SX1276:

Figura 21 - Diagrama de blocos SX1276



Fonte: (SEMTECH, 2020).

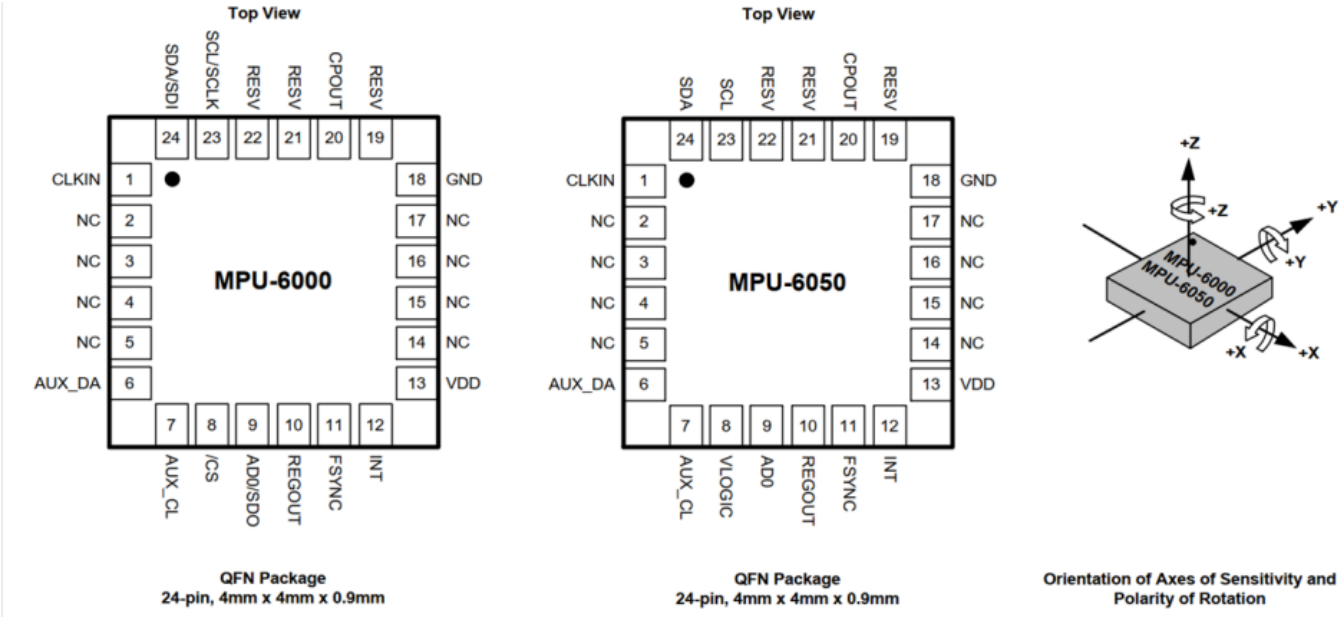
O transceptor SX1276 apresenta duas opções de referência de temporização: um oscilador RC e um cristal oscilador de 32MHz, oferecendo flexibilidade para escolher a melhor opção de acordo com as necessidades da aplicação. Além disso, para comunicação com um microcontrolador externo, o SX1276 utiliza o protocolo SPI para leitura e escrita de seus registradores.

3.4. Sensor MPU6050

O circuito integrado MPU6050 possui integrado um acelerômetro de 3 eixos, um giroscópio de 3 eixos e temperatura, além disso, possui o recurso DMP (*Digital Motion Processor*), responsável por realizar cálculos complexos, sendo utilizado em diversas aplicações como: navegação, sistemas de reconhecimento de gestos, sistemas de monitoramento de vibração. (INVENSENSE, 2013).

A pinagem pode ser visualizada conforme Figura 22 e as principais características deste CI, podem ser visualizadas na Tabela 3 a seguir:

Figura 22 - Vista de topo e orientação MPU-6050.



Fonte: (INVENSENSE, 2013).

Tabela 3 - Especificações MPU6050

Especificação	**Faixa de Valores**
Tensão de Operação	3.3 – 5V
Comunicação	I²C
Faixa do Giroscópio	±250, 500, 1000, 2000°/s
Faixa do Acelerômetro	±2, ±4, ±8, ±16g;
Faixa do Sensor de Temperatura	-40 a 85ºC
Resolução conversor AD	16 bits

Fonte: Adaptado de (INVENSENSE, 2013).

O MPU6050 é composto por um giroscópio de 3 eixos, que pode operar a uma taxa de amostragem de até 8 kHz e um acelerômetro, também de 3 eixos, que pode operar a uma taxa de amostragem de até 1 kHz. Todos os parâmetros podem ser configuráveis, como o índice de sensibilidade, configurações de calibração, filtros, taxas de saída e faixas de operação podem ser encontradas em sua folha de dados (INVENSENSE, 2013).

Um buffer FIFO de 1024 bytes no chip ajuda a reduzir o consumo de energia no sistema para ler os dados do sensor e, em seguida, entrar em um modo de baixo consumo à medida que coleta mais dados, suportando uma variedade de aplicações avançadas baseados em movimento (INVENSENSE, 2013).

3.5. Protótipo

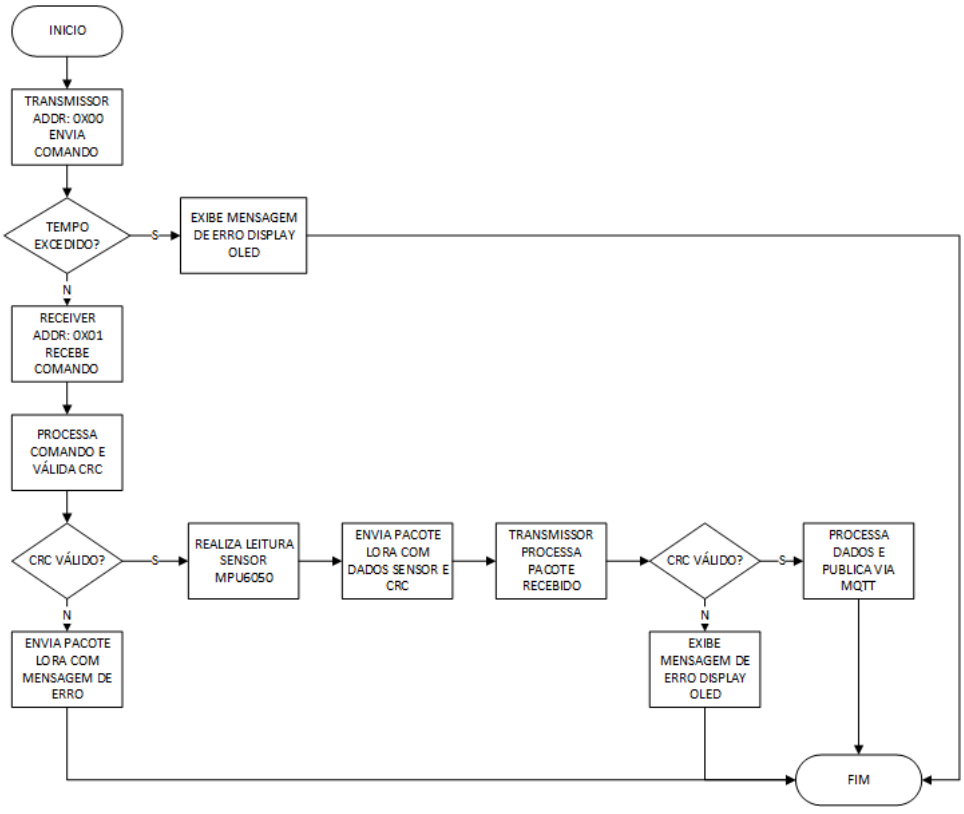
O projeto desenvolvido utiliza dois módulos Wi-Fi LoRa ESP32 V2, exemplificados em seção anterior e atendem aos requisitos estabelecidos na seção 3.1. A Figura 23 a demonstra os módulos utilizados:

Figura 23 -Módulos LoRa (Receptor e Transmissor)

Fonte: Elaborado pelo autor.

O fluxograma de funcionamento geral é apresentado na Figura 24:

Figura 24 - Fluxograma geral projeto



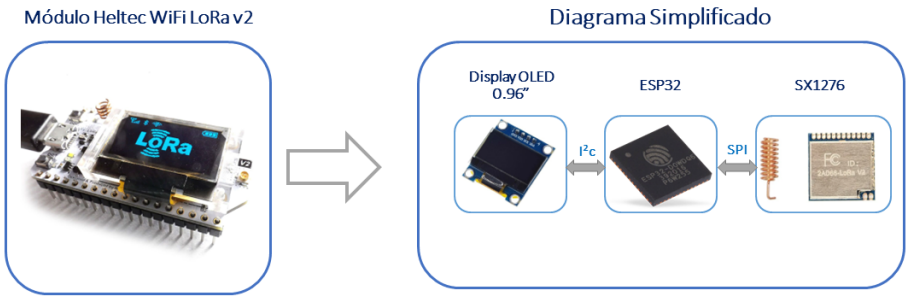
Fonte: Elaborado pelo autor.

O processo de comunicação entre os módulos é iniciado pelo módulo transmissor, que envia um pacote LoRa para o módulo receptor. Ao receber o pacote, o módulo receptor realiza a validação do CRC do LoRa e realiza a amostragem dos dados do acelerômetro para calcular os parâmetros necessários para a análise de vibração. Em seguida, o pacote é retornado ao módulo transmissor com confirmação de *acknowledgement*.

Uma vez recebido o pacote com os dados do acelerômetro, o módulo transmissor inicia a comunicação com o servidor em nuvem por meio do protocolo de comunicação MQTT, publicando os dados no tópico previamente definido.

O diagrama de blocos da placa Wi-Fi LoRa ESP32 V2 utilizada para o sistema desenvolvido como transmissor pode ser observada na Figura 25:

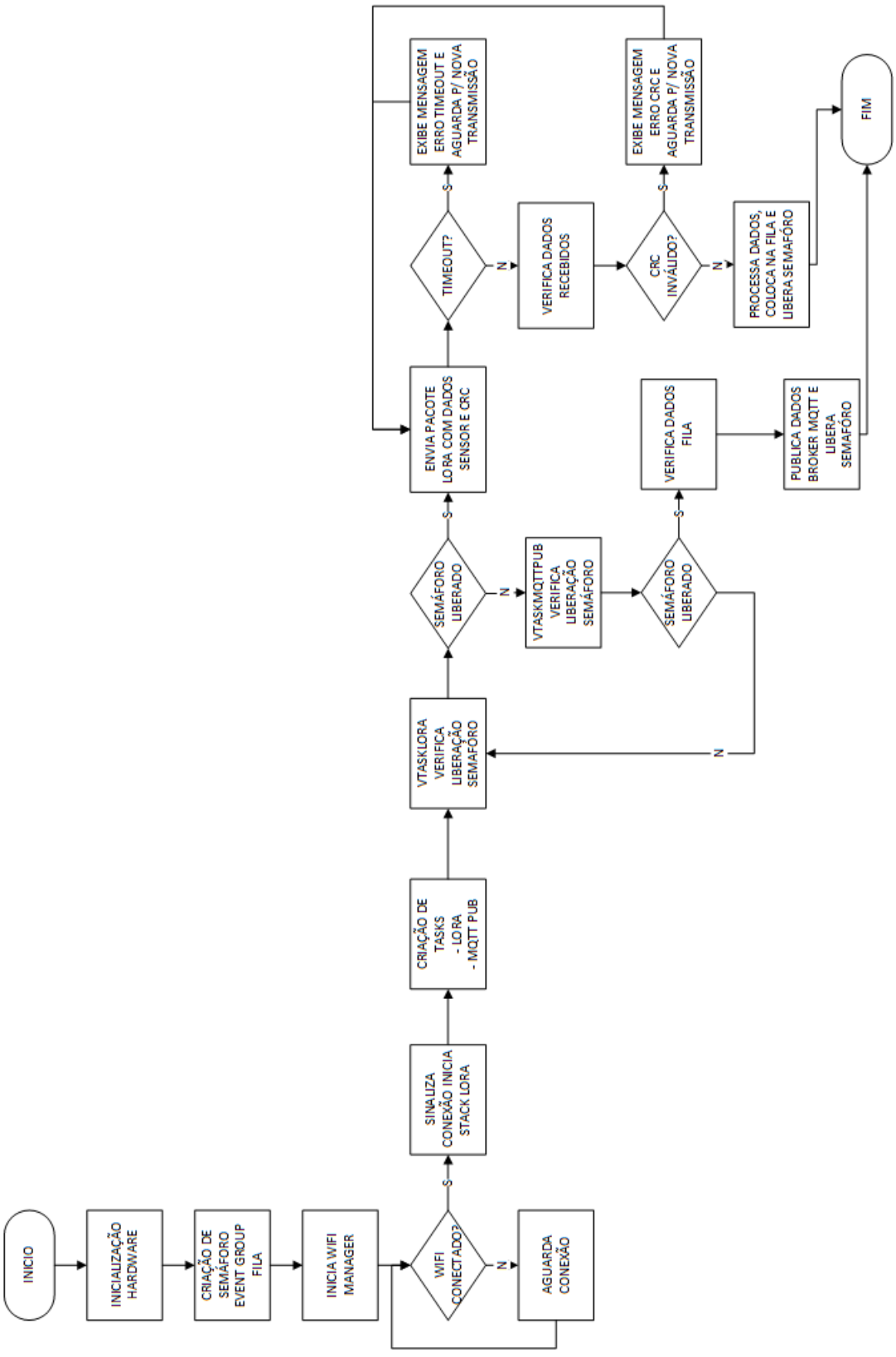
Figura 25 - Diagrama módulo transmissor



Fonte: Elaborado pelo autor.

O módulo transmissor é o responsável por iniciar a comunicação via LoRa e enviar os dados através do protocolo MQTT. A cada dez minutos, a comunicação é reiniciada. O fluxograma do firmware do módulo transmissor está representado na Figura 26:

Figura 26- Fluxograma firmware - Transmissor



Fonte: Elaborado pelo autor.

Após a inicialização de hardware, são utilizados os recursos do FreeRTOS para sinalizar a conexão Wi-Fi, o envio de dados via MQTT e a fila para comunicação entre as tarefas que utilizarão os dados recebidos.

A Figura 27 demonstra a função *app_main()* responsável pela inicialização de hardware e recursos do sistema operacional.

Figura 27 - Inicialização - Transmissor

```
/* app_main function body ------------------------------------------------------*/

void app_main(void)
{
    /*!< Inicia ESP32 e exibe algumas informações */
    esp32_start();

    /*!< Inicia display  OLED 0.96"*/
    ssd1306_start();

    /*!< Cria grupo de evento para sinalizar conexão WiFi */
    wifi_connected_event_group = xEventGroupCreate();
    if (wifi_connected_event_group == NULL)
    {
        ESP_LOGE("ERROR", "*** wifi_connected_event_group Create error ***\n");
        while (true);
    }

    /*!< Cria semaforo Mutex para sinalizar envio dados MQTT*/
    mqtt_connected_mutex = xSemaphoreCreateMutex();

    if (mqtt_connected_mutex == NULL)
    {
        ESP_LOGE("ERROR", "*** mqtt_connected_mutex Create error ***\n");
    }

    /*!< Cria fila para armazenamento dos dados lidos sensor*/
    sensor_data_queue = xQueueCreate(10, sizeof(char *));

    if (sensor_data_queue == NULL)
    {
        ESP_LOGE("ERROR", "*** sensor_data_queue Create error ***\n");
    }

    /*!< Cria fila para contador de pacotes enviados*/
    count_pkg_queue = xQueueCreate(5, sizeof(uint32_t));

    if (count_pkg_queue == NULL)
    {
        ESP_LOGE("ERROR", "*** count_pkg_queue Create error ***\n");
    }
```

Fonte: Elaborado pelo autor.

Durante a inicialização são realizadas verificações de hardware e exibição de algumas informações do ESP32 no depurador.

Posteriormente é criado o *EventGroup* responsável pela sinalização da conexão Wi-Fi, o semáforo e as filas utilizadas para controle de execução das tarefas gerenciadas pelo FreeRTOS.

Logo em seguida é inicializada a conexão Wi-Fi, enquanto não há conexão o sistema fica aguardando o bit do *EventGroup WI-FI_CONNECTED* ser verdadeiro, após confirmação de conexão é realizada a inicialização do rádio LoRa, com os parâmetros de frequência, habilitação do CRC e sinalização da comunicação através de interrupção externa.

A Figura 28 demonstra a conexão com a rede e início do rádio LoRa do dispositivo:

Figura 28 - Inicialização Wi-Fi e LoRa - Transmissor

```
    /*!< Inicia conexão WiFi manager */
    wifi_manager_start();

    /*!< register a callback as an example to how you can integrate your code with the wifi manager */
    wifi_manager_set_callback(WM_EVENT_STA_GOT_IP, &cb_connection_ok);

    /**
     * Aguarda conexão WiFi para processar Task
     */
    xEventGroupWaitBits(wifi_connected_event_group, WIFI_CONNECTED, false, true, portMAX_DELAY);

    /*!< Inicia LoRa e seta frequência 915MHz*/
    lora_init();
    lora_set_frequency(915E6);

    /*!< Habilita CRC*/
    lora_enable_crc();
    /*!< Habilita a recepção LoRa via Interrupção Externa;*/
    lora_enable_irq();

    /*!<Criação de Tasks*/

    /*!< Cria a task de transmissão LoRa*/
    if (xTaskCreate(vLoRaTxTask, "vLoRaTxTask", configMINIMAL_STACK_SIZE + 8192, NULL, 6, NULL) != pdTRUE)
    {
        ESP_LOGE("ERROR", "*** vLoRaTxTask error ***\n");
    }

    /*!< Cria a task de  MQTT para publicar dados*/
    if (xTaskCreate(vMQTTPublishTask, "vMQTTPublishTask", configMINIMAL_STACK_SIZE + 10240, NULL, 5, NULL) != pdTRUE)
    {
        ESP_LOGE("ERROR", "*** vMQTTPublishTask error ***\n");
    }
}
```

Fonte: Elaborado pelo autor.

São criadas as duas tarefas do módulo transmissor, sendo a *vLoRaTxTask* e a vMQTTPublishTask.

A tarefa *vLoRaTxTask* é responsável pelo início da comunicação LoRa iniciando a comunicação com módulos receptores da rede, no caso do projeto, há apenas um módulo receptor, aguarda a recepção dos dados requisitados, realizar a validação de CRC, colocar na fila para consumo de outra tarefa e por fim liberar o semáforo para o uso de outra tarefa. A Figura *29* demonstra a rotina de execução da tarefa:

Figura 29 - Implementação da vLoraTxTask - Transmissor

```
static void vLoRaTxTask(void *pvParameter)
{
    uint8_t protocol[160];
    static uint16_t count = 0;
    for (;;)
    {
        if (xSemaphoreTake(mqtt_connected_mutex, portMAX_DELAY))
        {
            /**
             * Protocolo;
             * <id_node_sender><id_node_receiver><command><payload_size><payload><crc>
             */
            ++count;
            protocol[0] = MASTER_NODE_ADDRESS;
            protocol[1] = LORA_TOTAL_NODES;
            protocol[2] = CMD_READ_MPU6050;
            protocol[3] = sizeof(count);
            protocol[4] = (UCHAR)(count & 0xFF);
            protocol[5] = (UCHAR)((count >> 8) & 0xFF);

            /**
             * Calcula o CRC do pacote;
             */
            USHORT usCRC = usLORACRC16(protocol, 6);
            protocol[6] = (UCHAR)(usCRC & 0xFF);
            protocol[7] = (UCHAR)((usCRC >> 8) & 0xFF);

            /**
             * Transmite protocol via LoRa;
             */
            lora_send_packet(protocol, 8);
```

Fonte: Elaborado pelo autor.

A variável *count* é responsável por sinalizar ao módulo receptor qual é o número do pacote que será transmitido, a implementação deste contador tem como objetivo identificar a quantidade de pacotes enviados e contabilizar possíveis perdas de pacotes.

A Figura 30 demonstra a chamada da função *lora_received_data()* que realiza a verificação da resposta recebida e tratamento de mensagens de erro:

Figura 30 - Chamada da função lora_received_data - Transmissor

```
            /**
             * Atualiza Display informando o pacote enviado
             *
             */
            ssd1306_clear();
            disp_connected();
            disp_packages(count, -1);

            vTaskDelay(10 / portTICK_RATE_MS);

            /**
             * Chama a função que irá receber os dados do acelerômetro, enviado pelo receptor;
             */
            lora_received_data();

            xSemaphoreGive(mqtt_connected_mutex);
            vTaskDelay(10E3 / portTICK_RATE_MS);
        }
    }
}
```

Fonte: Elaborado pelo autor.

A função responsável por receber os dados requisitados executa algumas verificações importantes. Primeiramente, é realizada a checagem do algoritmo de detecção de erros CRC. Caso seja detectado algum erro, uma mensagem é exibida no display OLED do módulo transmissor para alertar sobre a falha. Além disso, é verificado o comando recebido, e se tudo estiver correto, o dado é processado e armazenado na fila correspondente.

Quando o processo de recepção é concluído, uma mensagem de *acknowledgement* é exibida no display para confirmar que os dados foram recebidos e para indicar a confirmação do pacote transmitido pelo módulo receptor. Essa confirmação visual ajuda a identificar se houve perda de pacotes na transmissão.

A tarefa *vMQTT_PublishTask* é responsável pela comunicação com o *broker* MQTT e realizar a publicação dos dados recebidos do módulo receptor. A Figura 31 demonstra o corpo da tarefa:

Figura 31 - Implementação da vMQTT_PublishTask - Transmissor

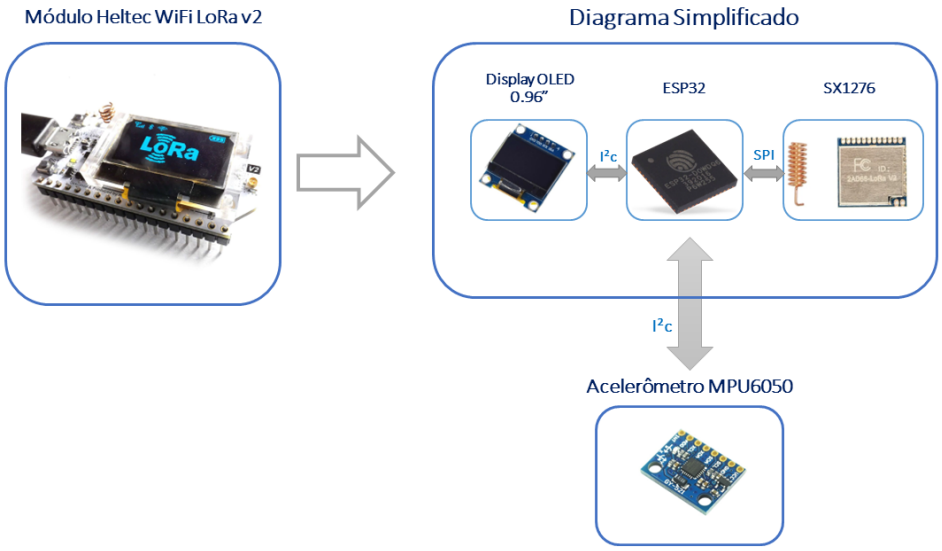
```c
static void vMQTTPublishTask(void *pvParameter)
{
    /*!< Inicio MQTT configurando Broker, User, ETC*/
    mqtt_app_start();

    char *pcDataReceived;

    for (;;)
    {
        if (xSemaphoreTake(mqtt_connected_mutex, portMAX_DELAY))
        {

            if (xQueueReceive(sensor_data_queue, &pcDataReceived, 0) == pdPASS)
            {
                ESP_LOGI(TAG, "Task MQTT Data: %s", pcDataReceived);
                vTaskDelay(10 / portTICK_RATE_MS);
                esp_mqtt_client_publish(client, MQTT_TOPIC, pcDataReceived, 0, 1, 0);
                free(pcDataReceived);
            }

            xSemaphoreGive(mqtt_connected_mutex);
            vTaskDelay(10 / portTICK_RATE_MS);
        }
    }
}
```

Fonte: Elaborado pelo autor.

O sistema transmissor realiza a comunicação com o módulo receptor através do protocolo LoRa e processa os dados recebidos. A verificação do algoritmo CRC é realizada para garantir a integridade dos dados. O comando recebido é então processado e alocado na fila, sendo exibida uma mensagem de *acknowledgement* no display OLED do módulo transmissor para confirmar a recepção dos dados e a confirmação do pacote transmitido pelo módulo receptor. Esse processo é realizado a cada 10 minutos, permitindo a transmissão regular dos dados via MQTT.

O sistema embarcado do receptor consta com um sensor externo, no caso o acelerômetro MPU6050 que se comunica com o ESP32 do módulo Wi-Fi LoRa ESP32 V2 da Heltec através do protocolo I²C, conforme diagrama da Figura 32:

Figura 32 - Diagrama módulo receptor



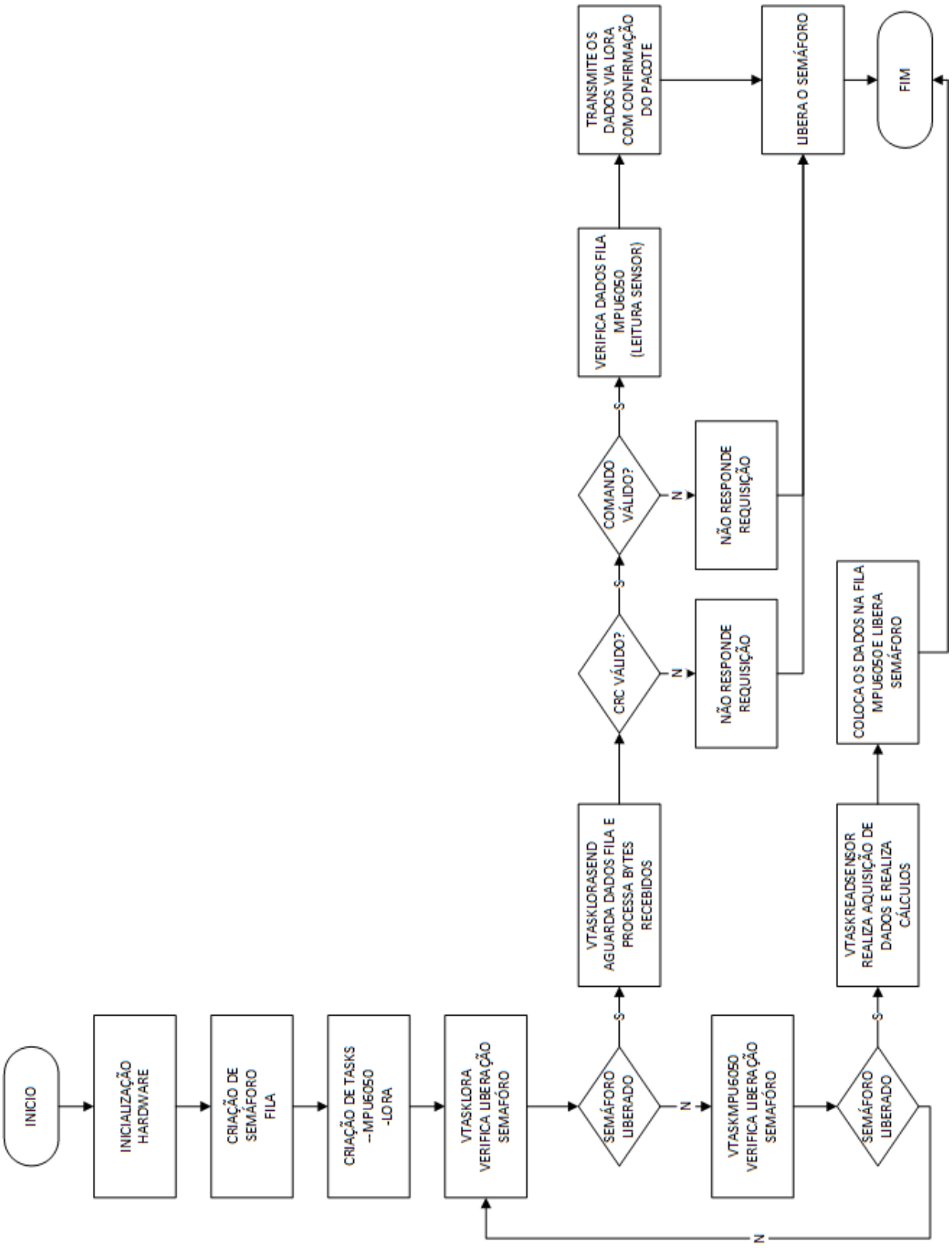
Fonte: Elaborado pelo autor.

O módulo receptor aguarda sempre a chegada de um comando do módulo transmissor antes de iniciar a comunicação. Caso o comando seja recebido com sucesso e o CRC seja válido, é verificado se o semáforo está liberado para prosseguir. Em seguida, são coletados dados do acelerômetro por meio de uma amostragem de 1024 leituras e, a partir desses dados, são calculados os parâmetros para análise de vibração, incluindo aceleração RMS, aceleração pico e velocidade RMS, além da medição da temperatura. No entanto, se o CRC ou o comando recebido forem inválidos, a requisição será ignorada.

A tarefa responsável por enviar a resposta à requisição realizada pelo transmissor, aguarda a liberação do semáforo, consome os dados armazenados na fila, realiza a conversão para o formato JSON (*Javascript Object Notation*) e transmite via LoRa ao módulo transmissor, visto que este formato é bastante comum, optou-se por ele neste projeto.

O fluxograma do firmware do sistema receptor é descrito conforme a Figura 33:

Figura 33 - Fluxograma firmware - Receptor



Fonte: Elaborado pelo autor.

Após a inicialização do hardware, duas tarefas são criadas: a *vTaskReadSensor*, responsável pela amostragem do sensor, cálculo dos parâmetros de análise de vibração e armazenamento dos dados em uma fila; e a *vTaskLoraSend*, que consome os dados da fila e responde às requisições do módulo transmissor. A *vTaskReadSensor* aguarda a liberação do semáforo para executar a amostragem do sensor, enquanto a *vTaskLoraSend* espera pela liberação do semáforo para consumir os dados da fila e transmiti-los via LoRa. A inicialização realizada na função *app_main()* é ilustrada conforme Figura 34:

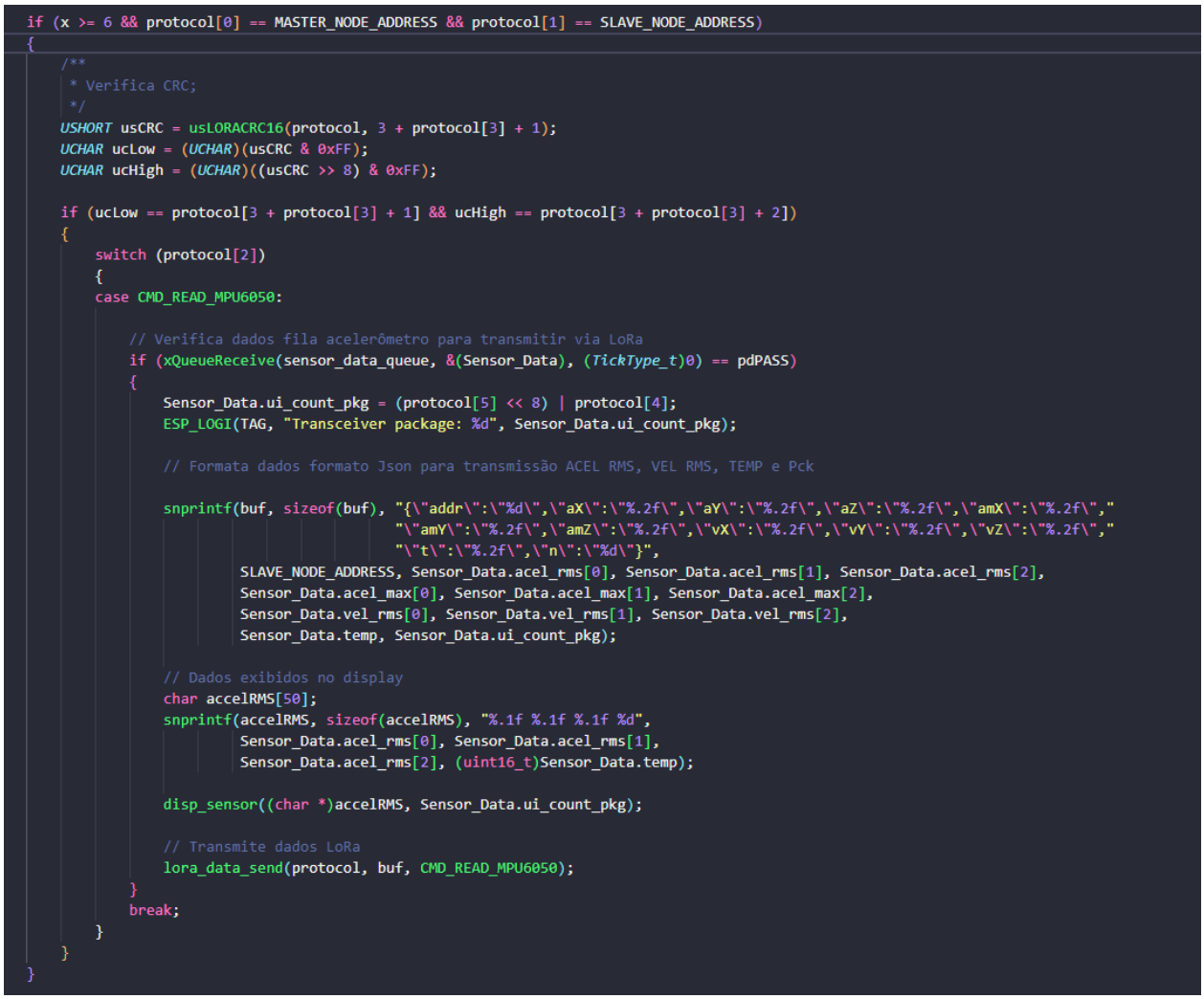
Figura 34 - Inicialização Módulo Receptor

```
/* app_main function body ------------------------------------------------------*/

void app_main(void)
{
    /*!< Inicia ESP32 e exibe algumas informações */
    esp32_start();

    /*!< Inicia display  OLED 0.96"*/
    ssd1306_start();

    /*!< Inicia sensor acelerômetro MPU6050*/
    i2c_sensor_mpu6050_init();

    /*!< Inicia LoRa e seta frequência 915MHz*/
    lora_init();
    lora_set_frequency(915E6);

    /*!< Habilita CRC*/
    lora_enable_crc();
    /*!< Habilita a recepção LoRa via Interrupção Externa;*/
    lora_enable_irq();

    /*!<Criação de Semafóro Mutex controle Tasks*/

    sensor_data_mutex = xSemaphoreCreateMutex();

    if (sensor_data_mutex == NULL)
    {
        ESP_LOGE("ERROR", "*** sensor_data_mutex Create error ***\n");
    }

    /*!<Criação de Fila para armazenar dados da struct*/
    sensor_data_queue = xQueueCreate(10, sizeof(sensor_data_t));

    if (sensor_data_queue == NULL)
    {
        ESP_LOGE("ERROR", "*** sensor_data_queue Create error ***\n");
    }

    /*!<Criação de Tasks*/

    if (xTaskCreate(vTaskReadSensor, "vTaskReadSensor", configMINIMAL_STACK_SIZE + 8192, NULL, 5, NULL) != pdTRUE)
    {
        ESP_LOGE("ERROR", "*** vTaskReadSensor error ***\n");
    }

    if (xTaskCreate(vTaskLoRaSend, "vTaskLoRaSend", configMINIMAL_STACK_SIZE + 8192, NULL, 5, NULL) != pdTRUE)
    {
        ESP_LOGE("ERROR", "*** vTaskLoRaSend error ***\n");
    }
}
```

Fonte: Elaborado pelo autor.

A tarefa *vTaskSendLora* espera pela liberação do semáforo e, em seguida, verifica se há dados armazenados na fila. Caso haja, a tarefa consome os dados e converte-os em uma *string* no formato JSON. Em seguida, a tarefa monta o pacote de transmissão para o módulo transmissor por meio da execução da função *lora_data_send()*. Por fim, a tarefa libera o semáforo para que possa ser utilizado por outra tarefa. A Figura 35 demonstra parte da implementação da tarefa:

Figura 35 - Implementação da vTaskSendLora - Receptor



```
if (x >= 6 && protocol[0] == MASTER_NODE_ADDRESS && protocol[1] == SLAVE_NODE_ADDRESS)
{
    /**
     * Verifica CRC;
     */
    USHORT usCRC = usLORACRC16(protocol, 3 + protocol[3] + 1);
    UCHAR ucLow = (UCHAR)(usCRC & 0xFF);
    UCHAR ucHigh = (UCHAR)((usCRC >> 8) & 0xFF);

    if (ucLow == protocol[3 + protocol[3] + 1] && ucHigh == protocol[3 + protocol[3] + 2])
    {
        switch (protocol[2])
        {
        case CMD_READ_MPU6050:

            // Verifica dados fila acelerômetro para transmitir via LoRa
            if (xQueueReceive(sensor_data_queue, &(Sensor_Data), (TickType_t)0) == pdPASS)
            {
                Sensor_Data.ui_count_pkg = (protocol[5] << 8) | protocol[4];
                ESP_LOGI(TAG, "Transceiver package: %d", Sensor_Data.ui_count_pkg);

                // Formata dados formato Json para transmissão ACEL RMS, VEL RMS, TEMP e Pck

                snprintf(buf, sizeof(buf), "{\"addr\":\"%d\",\"aX\":\"%.2f\",\"aY\":\"%.2f\",\"aZ\":\"%.2f\",\"amX\":\"%.2f\","
                                           "\"amY\":\"%.2f\",\"amZ\":\"%.2f\",\"vX\":\"%.2f\",\"vY\":\"%.2f\",\"vZ\":\"%.2f\","
                                           "\"t\":\"%.2f\",\"n\":\"%d\"}",
                        SLAVE_NODE_ADDRESS, Sensor_Data.acel_rms[0], Sensor_Data.acel_rms[1], Sensor_Data.acel_rms[2],
                        Sensor_Data.acel_max[0], Sensor_Data.acel_max[1], Sensor_Data.acel_max[2],
                        Sensor_Data.vel_rms[0], Sensor_Data.vel_rms[1], Sensor_Data.vel_rms[2],
                        Sensor_Data.temp, Sensor_Data.ui_count_pkg);

                // Dados exibidos no display
                char accelRMS[50];
                snprintf(accelRMS, sizeof(accelRMS), "%.1f %.1f %.1f %d",
                        Sensor_Data.acel_rms[0], Sensor_Data.acel_rms[1],
                        Sensor_Data.acel_rms[2], (uint16_t)Sensor_Data.temp);

                disp_sensor((char *)accelRMS, Sensor_Data.ui_count_pkg);

                // Transmite dados LoRa
                lora_data_send(protocol, buf, CMD_READ_MPU6050);
            }
            break;
        }
    }
}
```

Fonte: Elaborado pelo autor.

A tarefa *vTaskReadSensor* é responsável pela aquisição de dados do sensor, que incluem a amostragem de 1024 leituras do acelerômetro, o cálculo da aceleração média e do valor de pico, bem como a determinação dos valores

de aceleração e velocidade RMS. Após a realização dessas etapas, os parâmetros são armazenados em uma fila para posterior transmissão via LoRa.

Além disso, é importante destacar que a tarefa verifica a disponibilidade do semáforo antes de iniciar a amostragem, e, caso o semáforo esteja liberado, a aquisição dos dados é iniciada. Ao final do processo, o semáforo é liberado para que outras tarefas possam utilizá-lo. A Figura 36 demonstra parte da implementação da tarefa:

Figura 36 - Implementação da vTaskReadSensor - Receptor

```
for (uint16_t i = 0; i < SAMPLES; i++)
{

    mpu6050_acce_value_t MPU6050_Acce_Data;
    mpu6050_temp_value_t MPU6050_Temp_Data;

    mpu6050_get_acce(mpu6050, &MPU6050_Acce_Data);
    mpu6050_get_temp(mpu6050, &MPU6050_Temp_Data);

    Sensor_Data.acel_max[0] = max(MPU6050_Acce_Data.acce_x, Sensor_Data.acel_max[0]);
    Sensor_Data.acel_max[1] = max(MPU6050_Acce_Data.acce_y, Sensor_Data.acel_max[1]);
    Sensor_Data.acel_max[2] = max(MPU6050_Acce_Data.acce_z, Sensor_Data.acel_max[2]);

    Sensor_Data.acel_min[0] = min(MPU6050_Acce_Data.acce_x, Sensor_Data.acel_min[0]);
    Sensor_Data.acel_min[1] = min(MPU6050_Acce_Data.acce_y, Sensor_Data.acel_min[1]);
    Sensor_Data.acel_min[2] = min(MPU6050_Acce_Data.acce_z, Sensor_Data.acel_min[2]);

    Sensor_Data.acel_avg[0] += MPU6050_Acce_Data.acce_x;
    Sensor_Data.acel_avg[1] += MPU6050_Acce_Data.acce_y;
    Sensor_Data.acel_avg[2] += MPU6050_Acce_Data.acce_z;
    Sensor_Data.temp += MPU6050_Temp_Data.temp;
}

/*!< Calculo de acel (média), RMS, vel e vel RMS*/

Sensor_Data.acel_avg[0] = Sensor_Data.acel_avg[0] / SAMPLES;
Sensor_Data.acel_avg[1] = Sensor_Data.acel_avg[1] / SAMPLES;
Sensor_Data.acel_avg[2] = Sensor_Data.acel_avg[2] / SAMPLES;
Sensor_Data.temp = Sensor_Data.temp / SAMPLES;

Sensor_Data.acel_rms[0] = Sensor_Data.acel_avg[0] * M_SQRT1_2;
Sensor_Data.acel_rms[1] = Sensor_Data.acel_avg[1] * M_SQRT1_2;
Sensor_Data.acel_rms[2] = Sensor_Data.acel_avg[2] * M_SQRT1_2;

Sensor_Data.vel_rms[0] = (Sensor_Data.acel_avg[0] * CALC_SAMPLE_TIME) * M_SQRT1_2;
Sensor_Data.vel_rms[1] = (Sensor_Data.acel_avg[1] * CALC_SAMPLE_TIME) * M_SQRT1_2;
Sensor_Data.vel_rms[2] = (Sensor_Data.acel_avg[2] * CALC_SAMPLE_TIME) * M_SQRT1_2;
```

Fonte: Elaborado pelo autor.

O projeto segue a estrutura estabelecida pelos requisitos descritos na seção 3.1. É importante ressaltar que o módulo receptor só transmitirá dados após a solicitação do transmissor. Para aumentar a robustez e confiabilidade da comunicação entre os módulos, foram inseridas validações no projeto.

4. PROCEDIMENTOS EXPERIMENTAIS

No capítulo presente, são descritos os testes realizados em ambiente controlado para verificar a efetividade do sistema, bem como os testes de distância de comunicação. Os testes foram realizados considerando que um dos equipamentos estava em ambiente interno, permitindo avaliar a capacidade de transmissão do sistema em diferentes condições.

Para validar o desempenho do sistema, foram realizados testes preliminares que obtiveram resultados satisfatórios. Nesses testes, foi possível observar que o sistema se comportou de acordo com a modelagem proposta na seção 3.1.

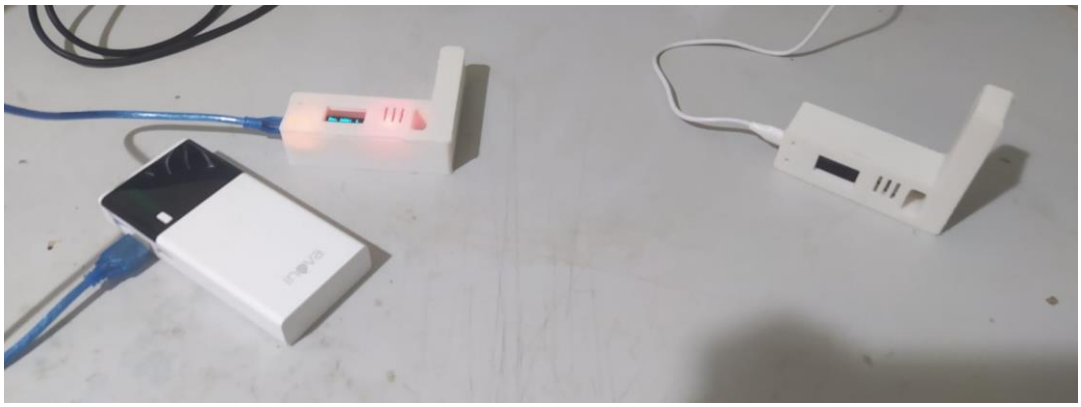
Todos os testes e coleta de dados foram realizados em bancada, adicionalmente foram realizados alguns testes de distância de comunicação entre os módulos, dentro do condomínio em que o autor reside também parcialmente em ambiente externo.

4.1. Estratégias de coleta e transmissão de dados

Para realizar a coleta de dados, o módulo transmissor foi mantido em ambiente interno, devido à necessidade de estar sempre conectado à rede Wi-Fi, já que sua função é receber as informações requisitadas e transmiti-las para o broker MQTT.

Para realizar as medições de distância de comunicação entre os módulos, foi utilizado um carregador portátil para alimentar o módulo receptor, possibilitando o deslocamento do equipamento e a realização de medições em diferentes ambientes. O padrão utilizado para as medições foi o SF7, que oferece maior taxa de transmissão de dados, porém, possui menor alcance em comparação a outros valores de SF. É importante destacar que não houve alteração do SF durante o projeto, mantendo-se sempre o SF7, conforme abordado na seção 2.3.1. Os módulos em bancada podem ser visualizados na Figura 37:

Figura 37 - Módulos LoRa Heltec em bancada



Fonte: Elaborado pelo autor.

O módulo transmissor é alimentado por um carregador de 5V através da porta mini USB e o módulo receptor utiliza os 5V fornecidos pelo carregador portátil através da porta mini USB.

4.2. Experimentos em laboratório e em campo

Foram realizados ensaios para validar o sistema proposto, utilizando o cenário de testes definido com base nos recursos disponíveis, conforme mencionado anteriormente. O módulo transmissor foi configurado para conectar-se à rede Wi-Fi, estabelecer conexão com o broker MQTT e iniciar o rádio LoRa, conforme descrito na seção 3.5. Ao conectar-se com sucesso à rede Wi-Fi, o *bit* do *EventGroup WI-FI_CONNECTED* é alterado para verdadeiro, sinalizando a conexão bem-sucedida.

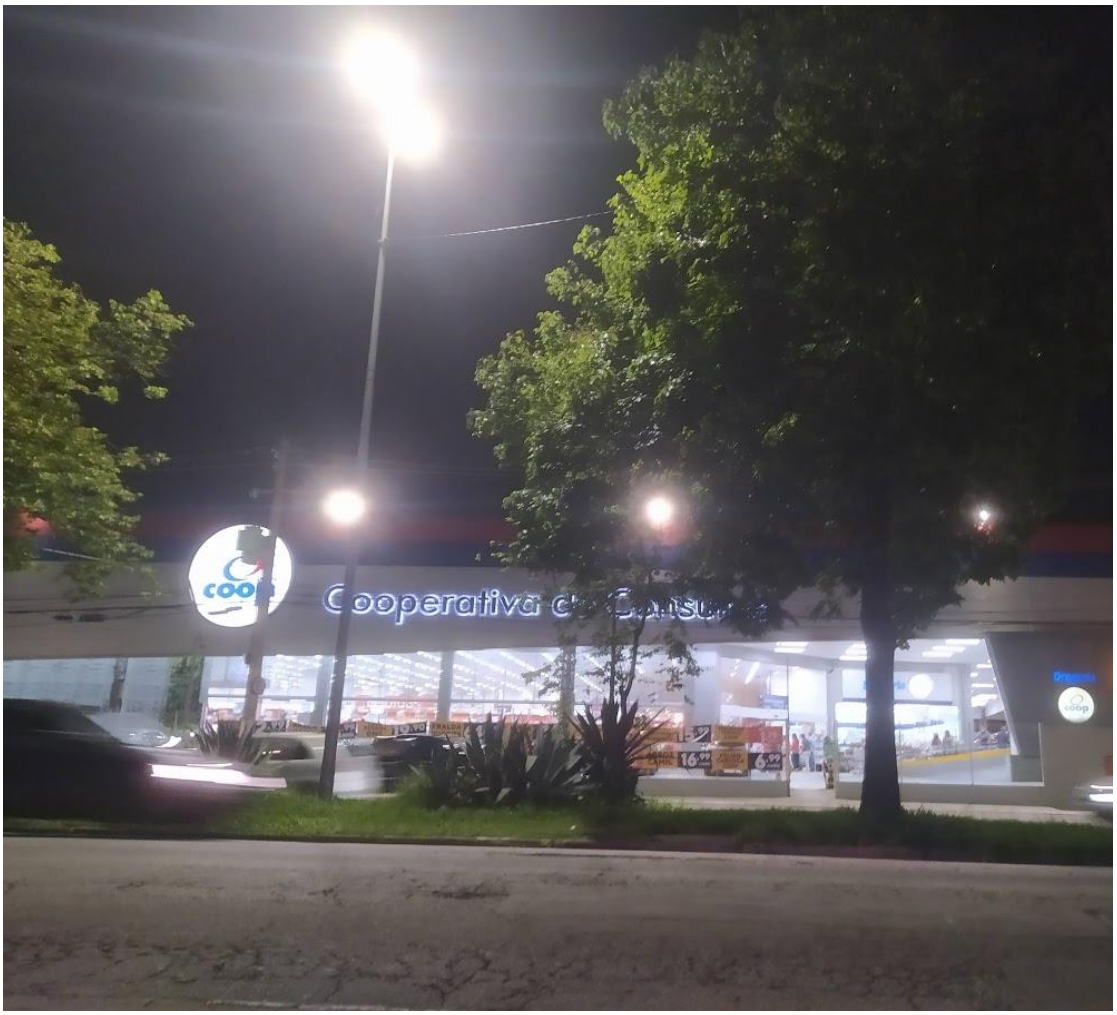
Após estabelecer a conexão com a rede Wi-Fi, o módulo transmissor envia uma solicitação de dados ao módulo receptor, aguarda a resposta, processa as informações e exibe uma confirmação de recebimento de dados, conhecida como *acknowledgement*.

Também foi possível demonstrar o cenário de falha de comunicação, conforme seção 3.5 demonstra em fluxograma, a cada requisição de dados realizada pelo módulo transmissor é aguardado um tempo de 5s. Caso não haja nenhuma resposta, é exibida mensagem de erro de *timeout*.

A fim de avaliar o desempenho do sistema em cenários reais, foram conduzidos testes de transmissão em vários pontos do condomínio onde o autor do projeto reside. Esses testes foram realizados com o módulo receptor em

ambiente externo e o módulo transmissor em ambiente interno, a fim de simular um cenário em que os dispositivos estariam localizados em áreas distintas. Essa abordagem permitiu verificar a capacidade do sistema em superar obstáculos físicos e alcançar uma distância máxima de transmissão, considerando o SF7 que, embora permita uma taxa de transmissão mais elevada, apresenta um alcance menor, conforme descrito na seção 2.3.1. A Figura 38 demonstra a avenida externa ao condomínio:

Figura 38 - Testes realizados: Avenida área externa

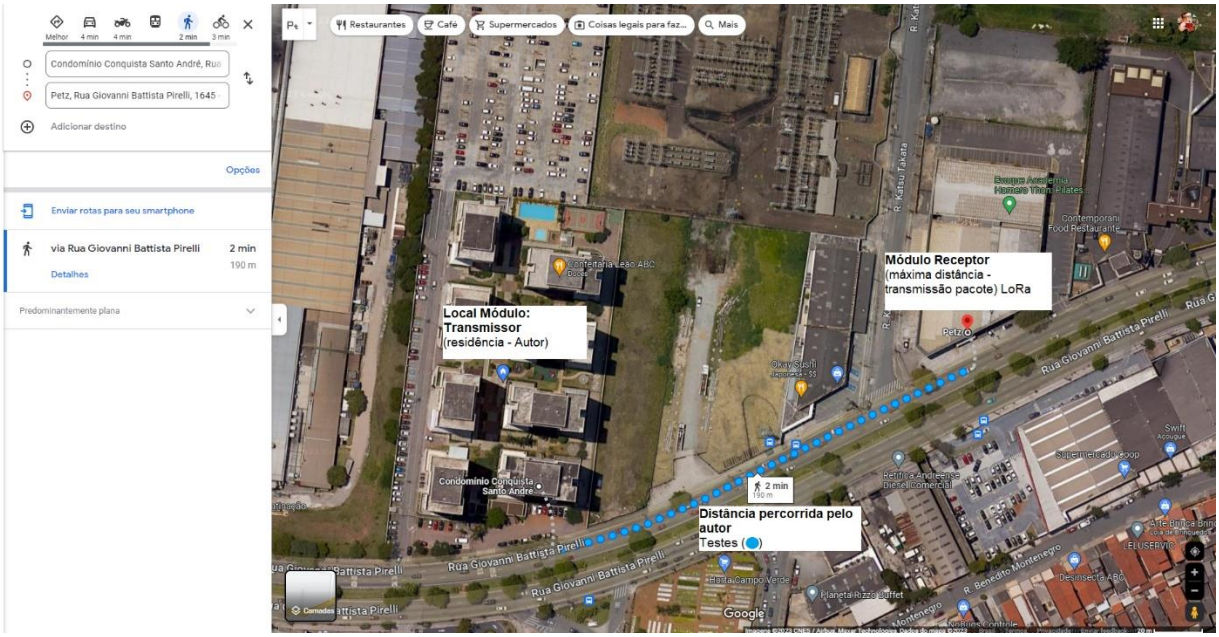


Fonte: Elaborado pelo autor.

A distância máxima alcançada foi de aproximadamente 200 metros, levando em consideração o deslocamento até a portaria do condomínio e o deslocamento em área externa, foi utilizado o SF7 fixo durante todo os testes.

A Figura 39 apresenta a localização aproximada dos dispositivos utilizados durante os testes:

Figura 39 - Teste realizado distância de transmissão (indoor)

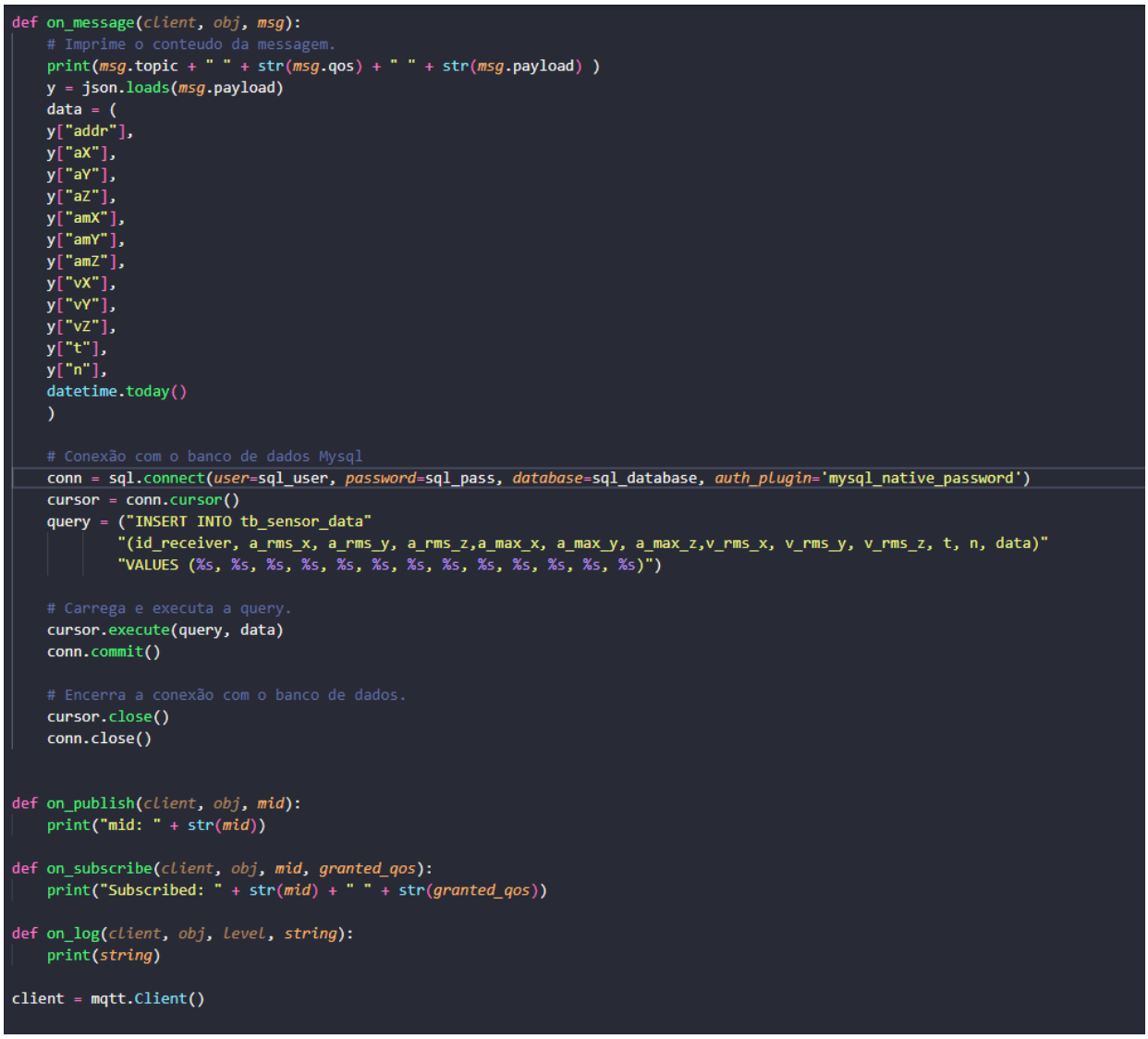


Fonte: Elaborado pelo autor.

Conforme abordado na seção 3.1 ao receber os dados requisitados o módulo transmissor publicará a mensagem no formato JSON através do protocolo MQTT em um servidor em nuvem.

Para receber os dados enviados pelos módulos, o servidor em nuvem foi configurado para utilizar o protocolo MQTT e armazenar as informações em um banco de dados MySQL. Para realizar essa tarefa, foi desenvolvido um script em Python, que possibilitou a integração do sistema com o banco de dados. O script desenvolvido pode ser observado na Figura 40:

Figura 40 - Script Python: Armazena dados no banco MySQL



```python
def on_message(client, obj, msg):
    # Imprime o conteudo da messagem.
    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload) )
    y = json.loads(msg.payload)
    data = (
    y["addr"],
    y["aX"],
    y["aY"],
    y["aZ"],
    y["amX"],
    y["amY"],
    y["amZ"],
    y["vX"],
    y["vY"],
    y["vZ"],
    y["t"],
    y["n"],
    datetime.today()
    )

    # Conexão com o banco de dados Mysql
    conn = sql.connect(user=sql_user, password=sql_pass, database=sql_database, auth_plugin='mysql_native_password')
    cursor = conn.cursor()
    query = ("INSERT INTO tb_sensor_data"
             "(id_receiver, a_rms_x, a_rms_y, a_rms_z,a_max_x, a_max_y, a_max_z,v_rms_x, v_rms_y, v_rms_z, t, n, data)"
             "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)")

    # Carrega e executa a query.
    cursor.execute(query, data)
    conn.commit()

    # Encerra a conexão com o banco de dados.
    cursor.close()
    conn.close()


def on_publish(client, obj, mid):
    print("mid: " + str(mid))

def on_subscribe(client, obj, mid, granted_qos):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))

def on_log(client, obj, level, string):
    print(string)

client = mqtt.Client()
```

Fonte: Elaborado pelo autor.

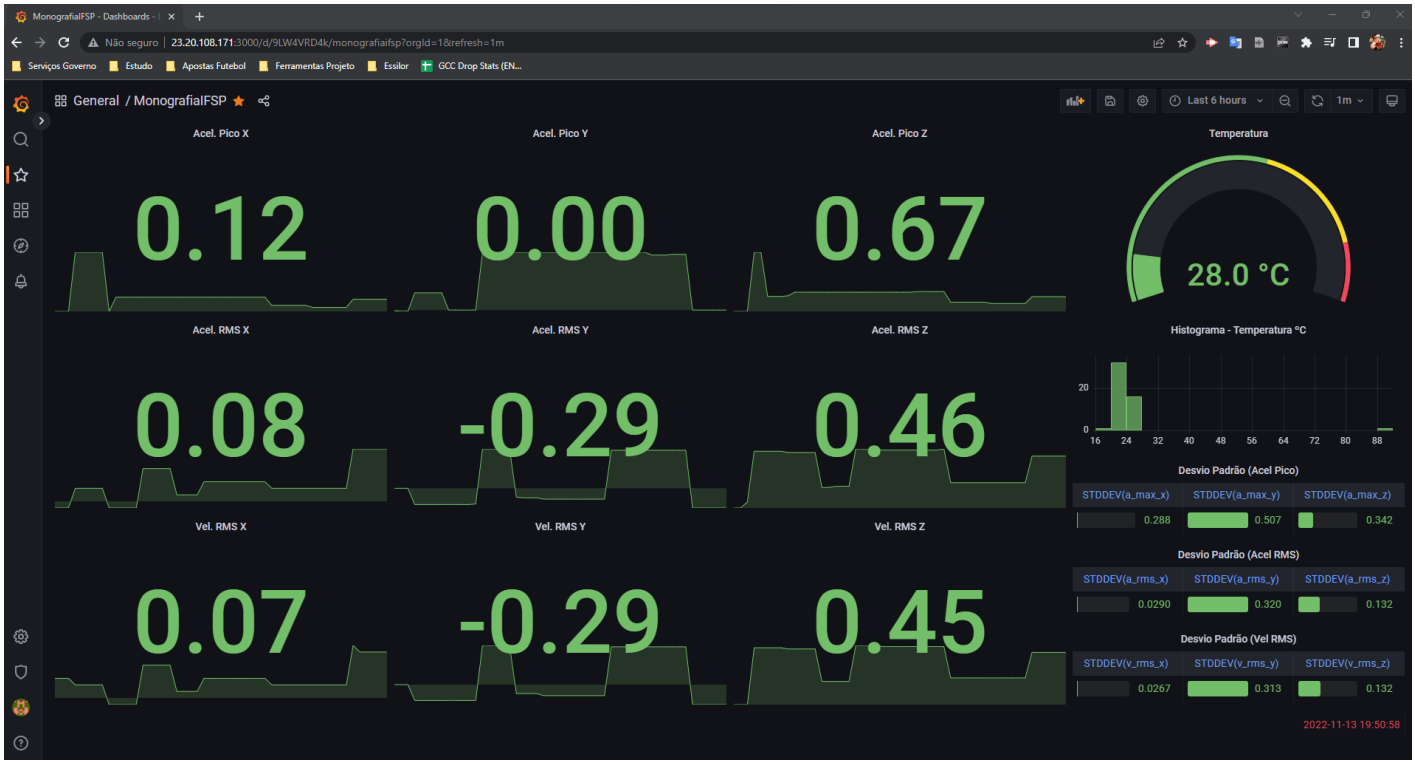
O script é responsável por realizar o processamento dos dados recebidos pelo servidor em nuvem através do protocolo MQTT. Ele faz a verificação do JSON recebido para garantir esteja no formato correto e então realiza a conexão com o banco de dados MySQL para inserir as informações coletadas na tabela *tb_sensor_data*. Esse processo é importante para garantir a integridade e organização dos dados recebidos, facilitando a análise e uso posterior deles.

A plataforma Grafana, mencionada na seção 3.1, foi instalada no servidor em nuvem da Amazon AWS. Dessa forma, é possível acessá-la de qualquer dispositivo que tenha conexão com a internet, tais como smartphones ou

computadores pessoais. Isso permite ao usuário monitorar as informações coletadas pelo sistema em tempo real, bem como visualizar relatórios e gráficos estatísticos de forma simples e intuitiva. Além disso, a plataforma também permite a configuração de alertas para notificar o usuário sobre possíveis eventos indesejados ou anomalias nos dados coletados.

Para exibir os dados coletados em um formato mais visual e acessível, é estabelecida uma conexão entre o Grafana e o banco de dados MySQL. A partir disso, são criados painéis gráficos que apresentam os dados coletados em tempo real, permitindo que os usuários possam monitorar as informações de forma mais eficiente. Além disso, foram realizados cálculos estatísticos básicos para fornecer informações sobre o comportamento do dispositivo, como média, máximo e mínimo dos valores coletados. Tudo isso permite uma melhor compreensão do desempenho do sistema. O painel gráfico tem como objetivo concentrar as informações coletadas pelo acelerômetro, a Figura 41 demonstra o mesmo:

Figura 41 - Painel gráfico: Grafana



Fonte: Elaborado pelo autor.

Foram adicionados ao painel gráfico o histograma de temperatura e o desvio padrão da aceleração RMS para visualizar a distribuição e consistência dos dados coletados. Adicionalmente, é possível configurar alertas na plataforma para notificar os usuários através de e-mail, Telegram ou outras plataformas integradas via API.

Este projeto propõe uma aplicação para leitura de aceleração e velocidade RMS visando o monitoramento de vibração em máquinas rotativas. A plataforma Grafana possibilita a definição de alertas para notificação com base nos limites máximos estipulados pela ISO 10816-1, que estabelece diretrizes gerais para medição e avaliação da vibração mecânica em peças não rotativas. Assim, o sistema proposto é uma ferramenta útil para monitorar a vibração de máquinas, garantindo maior segurança e eficiência na operação.

As zonas típicas de vibração conforme a norma são de acordo com a Figura 42.

Figura 42 - Zonas típicas de acordo com a norma ISO-10816

Norma ISSO 10816 - Classe de Equipamento					
Velocidade Severidade		Intervalo de Velocidades e Classes de Máquinas			
mm/s (RMS)	in/s (Pico)	Classe I Máquina Pequena	Casse II Máquina Média	Classe III Fundação Rígida	Casse IV Fundação Flexível
0,28	0,02	A	A	A	A
0,45	0,03				
0,71	0,04				
1,12	0,06	B			
1,8	0,1		B		
2,8	0,16	C		B	
4,5	0,25		C		B
7,1	0,4	D		C	
11,2	0,62		D		C
18	1			D	
28	1,56				D
45	2,51				
Zona A Verde: valores de vibração operacionais. Zona B Amarelo: Operação contínua sem restrições. Zona C Laranja: Condição é aceitável apenas por um período limitado de tempo. Zona D Vermelho: Valores de vibração perigosos - falha iminente.					

Fonte: Adaptado de ISO 10816-1.

Compreendendo que o foco deste trabalho é a exploração da tecnologia LoRa e a aplicação do sistema embarcado com o uso do FreeRTOS, não foram abordados de forma aprofundada os tópicos relacionados à análise de vibração e suas principais aplicações práticas.

Através do uso da plataforma Grafana, foram criados alertas para temperatura máxima e velocidade excedida, com notificação por e-mail. Isso permite que a aplicação proposta não apenas monitore, mas também emita alertas de valores fora dos limites estabelecidos pela norma.

Os resultados obtidos através dos testes em ambiente controlado foram comparados ao sistema proposto, e é possível afirmar que eles estão de acordo com o que foi projetado anteriormente.

Foram realizados testes em um ambiente controlado para verificar a eficiência dos algoritmos de verificação de erros e confirmação de recebimento de dados, a fim de garantir que as informações coletadas pelo acelerômetro fossem confiáveis.

Além disso, foi utilizado um servidor em nuvem para armazenar os dados coletados e a plataforma Grafana para monitorar e analisar essas informações, permitindo a identificação de anomalias e a criação de alertas para valores fora dos limites estabelecidos pela norma.

Com base nos resultados obtidos nos testes, pode-se afirmar que o sistema proposto é eficaz e confiável para o monitoramento de vibração em máquinas rotativas.

5. CONCLUSÃO

Neste capítulo são apresentadas as considerações finais do trabalho proposto além de sugestões para trabalhos futuros com o uso do FreeRTOS em conjunto com a tecnologia de transmissão de dados sem fio LoRa.

5.1. Considerações finais

Este trabalho apresentou um estudo sobre a tecnologia de transmissão de dados sem fio LoRa e os parâmetros considerados na camada física, além disso foi realizado o estudo e implementação do sistema com o uso do FreeRTOS um sistema operacional de tempo real.

O sistema proposto conforme seção 3.1 trata-se de dois módulos LoRa com comunicação P2P, sendo a comunicação iniciada sempre pelo módulo transmissor enviando um *frame* LoRa e aguardando os dados requisitados pelo módulo receptor, sendo possível incluir na rede mais de um módulo receptor, ainda conta com a validação através de algoritmo CRC e confirmação de recebimento através de *acknowledgement.*

O protótipo desenvolvido mostrou-se promissor e com custo acessível, bem como o uso de um servidor em nuvem para concentrar os dados, armazenar em um banco de dados e utilizar uma ferramenta para a aplicação, que além de exibir os dados coletados, analisa o desvio padrão e o histograma, além de monitorar é possível gerar alertas para não conformidades.

Considerando o objetivo e os resultados alcançados pelos protótipos utilizados neste projeto, pode-se dizer que há aplicação para monitoramento de equipamentos industriais demostrando a versatilidade do uso da tecnologia LoRa.

Após a realização deste trabalho, pode-se afirmar que o sistema P2P proposto é funcional e possui mecanismos que garantem a confiabilidade e integridade dos dados transmitidos. Além disso, foi possível estudar e utilizar recursos de um sistema operacional de tempo real (RTOS) em um sistema embarcado de arquitetura de 32 bits. Outra contribuição importante foi o armazenamento das informações coletadas em um banco de dados, possibilitando a geração de um histórico do comportamento da máquina monitorada ao longo do tempo.

5.2. Sugestões para trabalhos futuros

O objetivo primordial do presente trabalho foi o estudo e desenvolvimento de um sistema embarcado, com a aplicação do FreeRTOS e a utilização da tecnologia LoRa como meio de comunicação entre diferentes dispositivos, visando monitorar máquinas industriais em ambientes controlados.

A continuidade desse trabalho pode ser direcionada para a aplicação do sistema em ambientes não controlados, objetivando compreender as características de vibração das máquinas industriais em operação e implementar o algoritmo para o cálculo da FFT (*Fast Fourier Transform*) a fim de analisar o espectro das frequências de vibração, que é uma técnica muito utilizada para detecção de possíveis falhas.

Também, é possível incorporar o protocolo LoRaWAN como complemento à aplicação proposta. Tal protocolo permitirá aplicar mais recursos de proteção dos dados, tais como criptografia e segurança na transmissão, aumentando a robustez e confiabilidade no quesito segurança da informação. Além disso, a implementação desse protocolo permitirá incluir mais nós na rede e, ainda, aplicar um algoritmo de aprendizado de máquina, compreendendo o comportamento do sistema monitorado com base no histórico e, por meio de predições, antecipar possíveis falhas, tornando possível a aplicação de manutenção preditiva em máquinas industriais.

Este trabalho apresentou uma aplicação promissora do LoRa no monitoramento de máquinas industriais, possibilitando a coleta de informações de vibração por meio de acelerômetros e a transmissão desses dados de forma eficiente e segura. É importante ressaltar que o uso do LoRa não se limita a essa aplicação, já que essa tecnologia é utilizada em diversos setores, como na agricultura, monitoramento ambiental, rastreamento de veículos, entre outros.

O LoRa é uma tecnologia de comunicação de longo alcance que oferece alta eficiência energética, robustez e segurança na transmissão de dados, o que a torna uma escolha atraente para diversas aplicações.

REFERÊNCIAS BIBLIOGRÁFICAS

ADVANTECH. WISE-2410 LoRaWAN Smart Condition Monitoring Sensor. Edition. 1, 2020. Disponível em: < https://advdownload.advantech.com/productfile/Downloadfile1/11ZFBJAP/WISE-2410_User_Manual_Ed.1.Final.pdf> Acesso em: 15 ago. 2022

ANATEL (2018). Faixas de radiofrequências utilizáveis por equipamentos de radiação restrita. Disponível em: < http://www.anatel.gov.br/legislacao/resolucoes/2018/1220-resolucao-705.> Acesso em 06 de fev. 2020.

AUGUSTIN, A. et al. A study of lora: Long range and low power networks for the internet of things. Sensors, v. 16, n. 9, 2016.

BARRY, Richard. Mastering the FreeRTOS™ Real Time Kernel: A Hands-On Tutorial Guide. 161204. ed. [S. l.], 2016. Disponível em: https://freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. Acesso em: 15 nov. 2021.

EMBARCADOS (2016). Conheça a tecnologia LoRa® e o protocolo LoRaWAN™, Disponível em: <https://www.embarcados.com.br/conheca-tecnologia-lora-e-o-protocolo-lorawan/>. Acesso em 06 de fev. 2020.

EMBARCADOS: MQTT – Protocolos para IoT: Marcelo Barros, 2015. Disponível em: <https://www.embarcados.com.br/mqtt-protocolos-para-iot.> Acesso em: 8 mar. 2022.

ESPRESSIF SYSTEMS. ESP32 Series: Datasheet. 3.8, 2021. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 8 jan. 2022.

FERNANDES, F. Internet Das Coisas e Monitoramento De Vibração: Protótipo De Um Nó Sensor. 2020. Dissertação (Mestrado) – Universidade Federal de Ouro Preto. Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração, Ouro Preto, 2020.

FREERTOS. [*S. l.*], 2021. Disponível em: <https://aws.amazon.com/pt/freertos/.> Acesso em: 19 nov. 2021.

HELAL, A. A. et al. Estacqua: Proposta de solução integrada de hardware, software e internet das coisas para monitoramento ambiental. In: Sociedade Brasileira de Computação OPEN LIB. Natal: [s.n.], 2018. ISSN 2595-6205. Disponível em: https://sol.sbc.org.br/index.php/ semish/article/view/3432.

HELTEC. Wi-Fi Kit 32. 2019. Disponível em < https://heltec.org/project/Wi-Fi-lora-32/>: Acesso em: 15 nov. 2021.

HIVEMQ. MQTT Essentials: Introducing MQTT. Disponível em: <http://www.hivemq.com/mqtt-essentials/>. Acesso em: 10 abr. 2022.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO10816-1 - Mechanical vibration: Evaluation of machine vibration by measurements on non-rotating parts. San Diego,CA: Machinery Information Management Open Systems Alliance (MIMOSA), 7-13, 1995.

INVENSENSE. MPU-6000 and MPU-6050 Product Specification. 3.4., 2013. Disponível em: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>. Acesso em: 6 mar. 2022.

J. MANYIKA et al, “The Internet of Things: Mapping the value beyond the hype,” 2015.

KOYANAGI Fernando, ESP8266 e ESP32 com Wi-FiManager. 2018. Disponível em: <https://www.fernandok.com/2018/03/esp8266-e-esp32-com-Wi-Fimanager.html>. Acesso em: 1 jun. 2022.

LAVRIC Alexandru, "LoRa (Long-Range) High-Density Sensors for Internet of Things", Journal of Sensors, vol. 2019, Article ID 3502987, 9 pages, 2019. https://doi.org/10.1155/2019/3502987

LEENS, F. An introduction to I2C and SPI protocols. IEEE Instrumentation and

LORA ALLIANCE. LoRaWAN™ 1.0.3 Specification. 2018. Disponível em: <https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1.0.3.pdf>. Acesso em: 08 mar. 2022.

M. Centenaro, L. Vangelista, A. Zanella and M. Zorzi, "Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios," in IEEE Wireless Communications, vol. 23, no. 5, pp. 60-67, October 2016, doi: 10.1109/MWC.2016.7721743.

MEIRELES, L. Projeto de uma Rede de Internet das Coisas para Monitoramento e Alerta de Emergência em Áreas De Risco. 2018 Dissertação (Mestrado) – Universidade Federal de Ouro Preto. Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração, Ouro Preto, 2018.

MICROGENIOS: Formação em Internet das Coisas. Disponível em: < https://www.microgenios.com.br/formacao-iot-esp32/> Acesso em: 08 jan. 2022.

NXP SEMICONDUCTORS. UM10204 I2C-bus specification and user manual. Rev. 7., October, 2021. Disponível em: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf> Acesso em: 10 abr. 2022

OASIS (2014). MQTT Version 3.1.1. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>. Acesso em: 08 mar. 2022.

OPEN SOURCE INITIATIVE. The 2-Clause BSD License, 24 setembro 2018. Disponivel em: Acesso em: 15 nov. 2022.

RAMMIG, Franz et al. Basic concepts of real time operating systems. In: HARDWARE DEPENDENT Software. [S.l.]: Springer, 2009. p. 15–45

SEMTECH. AN120.22 LoRa Modulation Basics. [S.l.], 2015

SEMTECH. SX1276-7-8-9 Datasheet. Rev. 7., 2020. Disponível em: <https://www.semtech.com/products/wireless-rf/lora-core/sx1276>. Acesso em: 8 maio 2021

Shuda, Joseph & Rix, Arnold & Booysen, M.J. (Thinus). (2018). Module-Level Monitoring Of Solar PV Plants Using Wireless Sensor Networks. Measurement Magazine, v. 12, n. 1, p. 8–13, 2009.

APÊNDICE A – CÓDIGO FONTE

Os arquivos com os códigos fonte utilizados neste projeto proposto são divididos em 3 pastas, sendo:

- **transmissor**: Código fonte do módulo transmissor;
- **receptor**: Código fonte do módulo receptor;
- **scripts**: Código fonte script Python;

Disponível no GitHub do autor:

https://github.com/YhanChristian/EstudoMicrocontrolares/tree/master/ESP-32/monografia_ifsp_lora