UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Renê Couto e Silva

Aplicação de IoT para conversores estáticos de potência

Florianópolis
2022

Renê Couto e Silva

Aplicação de IoT para conversores estáticos de potência

Trabalho de Conclusão do Curso de Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Engenharia Elétrica.
Orientador: Prof. Samir Ahmad Mussa, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Silva, Renê
Aplicação de IoT para conversores estáticos de potência / Renê Silva ; orientador, Samir Ahmad Mussa, 2022.
79 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2022.

Inclui referências.

1. Engenharia Elétrica. 2. IoT. 3. Software Embarcado. I. Ahmad Mussa, Samir. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Renê Couto e Silva

Aplicação de IoT para conversores estáticos de potência

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Engenharia Elétrica” e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 3 de Agosto de 2022.

Documento assinado digitalmente
Miguel Moreto
Data: 09/08/2022 10:22:22-0300
CPF: 948.850.100-63
Verifique as assinaturas em https://v.ufsc.br

Prof. Miguel Moreto, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Documento assinado digitalmente
Samir Ahmad Mussa
Data: 09/08/2022 11:02:47-0300
CPF: 401.044.430-49
Verifique as assinaturas em https://v.ufsc.br

Prof. Samir Ahmad Mussa, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
HARI BRUNO MOHR
Data: 08/08/2022 23:11:43-0300
CPF: 096.407.629-20
Verifique as assinaturas em https://v.ufsc.br

Prof. Hari Bruno Mohr, PhD.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Jose Augusto Arbugeri
Data: 08/08/2022 17:44:41-0300
CPF: 078.092.839-36
Verifique as assinaturas em https://v.ufsc.br

Eng. José Augusto Arbugeri.
REIVAX

Florianópolis, 2022.

Este trabalho é dedicado aos meus colegas de classe e
aos meus queridos pais.

AGRADECIMENTOS

Agradeço a meus pais por terem me encorajado e me apoiado em direção à capacitação acadêmica e profissional, mesmo que longe de casa. Sou muito grato também pelos colegas de curso que tive pelo companheirismo. Também aos amigos que fiz durante o período de faculdade. Agradeço a meu psicólogo Pablo Rubino por me ajudar a manter o foco nas minhas prioridades e me dar ferramentas para realizar os meus objetivos. A todos os professores que tive durante o curso. A todos da secretaria e da coordenação do curso. Infelizmente demorei muito a perceber o valor desse trabalho, e o quanto eu perdi por fugir do que precisava encarar. Agradeço especialmente a meu orientador Samir Ahmad Mussa por todo o apoio e paciência nesse projeto.

“Não quero ficar dando adeus às coisas passando,
eu quero é passar com elas"
(Macalé, 1972)

RESUMO

O projeto tem como objetivo implementar o monitoramento de grandezas elétricas de conversores estáticos de potência controlados por Field-programmable Gate Array (FPGA) ou Digital Signal Processor (DSP). Para isso, essas grandezas devem ser extraídas para algum modo de armazenamento na nuvem de computação e disponibilizadas para visualização. Um microcontrolador com acesso a WiFi e SPI será utilizado para extrair os parâmetros do conversor através da interface SPI e enviá-los para a nuvem utilizando uma rede WiFi. Foram consideradas diferentes abordagens para o método de envio, armazenamento e plotagem desses parâmetros. Um kit de desenvolvimento foi utilizado para simular o FPGA. Também foi implementada a funcionalidade de recebimento de comandos utilizando o protocolo de comunicação MQTT. Foi observado que a implementação ainda possui pontos para melhorar, mas possui desempenho aceitável.

Palavras-chave: IOT. MQTT. ESP. WiFi. SPI.

ABSTRACT

This project aims to implement the monitoring and extraction of electrical measurements of static power converters controlled by FPGA or DSP to some kind of cloud storage and to make those measurements available for visualization. A microcontroller with WiFi an SPI access will be used to extract parâmeters from the FPGA through the SPI interface and send them to the cloud using a WiFi network. Many different approaches were considered to the sending, storing and plotting of the parameters. A development kit was used to simulate the FPGA. A command receiving functionality was also implemented using the MQTT communication protocol. It was observed that the implementation still has room to grow, but delivers an acceptable performance.

Keywords: IOT. MQTT. ESP. WiFi. SPI.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
CSV	Comma-Separated Values
DSP	Digital Signal Processor
FPGA	Field-programmable Gate Array
GCP	Google Cloud Platform
GCS	Google Cloud Storage
IDE	Integrated Development Environment
IIoT	Industrial Internet of Things
IoT	Internet of Things
MISO	Master-In Slave-Out
MOSI	Master-Out Slave-In
PFC	Power-Factor Correction
RDBMS	Relational Database Management System
S3	Simple Storage Service
SCK	System Clock
SPI	Serial Peripheral Interface
SS	Slave Select
TSDB	Time Series Database

SUMÁRIO

1 INTRODUÇÃO

O projeto está destinado a realizar a extração de parâmetros de um conversor estático de potência e armazenamento desses na nuvem de computação para plotagem e análise. Ao decorrer deste, diferentes estratégias foram levantadas para a extração via Serial Peripheral Interface (SPI) e envio para a nuvem através da conexão a uma rede WiFi local. Cada estratégia analisada possui suas peculiaridades, mas o foco principal foi obter um desempenho aceitável para armazenamento de formas de ondas senoidais com cerca de 60 Hz de frequência.

Um grande desafio do software embarcado é monitorar em tempo real suas grandezas e parâmetros. Ter dados sobre o funcionamento de um conversor disponíveis na nuvem ajuda a evidenciar possíveis falhas e anomalia. Ter o conversor também uma forma de interagir a comandos externos possibilita a integração desse com outros possíveis sistemas centralizados.

Tendo em vista que esse trabalho deriva do projeto de Arbugeri (2019), no qual um conversor Power-Factor Correction (PFC) controlado por um FPGA está conectado a um microcontrolador ESP8266, são encontradas aplicações similares, como o uso de Internet of Things (IoT) em paineis de APFC (correção automática de fator de potência) para cargas industriais (BHAGAVATHY; LATHA; THAMIZHMARAN, 2019; BARHATE *et al.*, 2019). No âmbito de monitoramento de sensores, a IoT tem bastante presença também em projetos pessoais. Para realizar a extração de dados do conversor e publicação desses dados na nuvem um microcontrolador (ESP32 ou ESP8266) é utilizado. Para emular o comportamento do conversor, em testes iniciais, sem a necessidade de um conversor estático, foi usado um microcontrolador para simular o comportamento dos sinais.

A natureza do problema ganha uma certa complexidade pela combinação de limitação de memória desses dispositivos, da frequência e precisão em que se deseja extrair os dados e da frequência em que se quer publicar os dados. Logo, avaliou-se a viabilidade de envios para bancos de dados relacionais (Relational Database Management System (RDBMS)), bancos de dados de séries temporais (Time Series Database (TSDB)), armazenamentos de arquivos remotos, como Google Drive, além de armazenamentos de documentos, como Amazon Web Services (AWS) Simple Storage Service (S3) e Google Cloud Platform (GCP) Google Cloud Storage (GCS).

No capítulo 2 há uma breve fundamentação das tecnologias utilizadas. No capítulo 3 há o design inicial do projeto. No capítulo 4, há um detalhamento dos questionamentos referentes à escolha de ferramentas. No capítulo 5, é implementa o envio de dados para armazenamento. No capítulo 6, é implementado um formato de mensagem para comunicação entre o controlador do conversor e o microcontrolador. No capítulo 7, é implementado o recebimento de comandos via MQTT, é configurado um broker

MQTT, um serviço para receber mensagens do broker e inserí-los no armazenamento, e são definidos alguns comandos que o microcontrolador pode receber através do broker. No capítulo 8, especula-se os requisitos para se ter a arquitetura desenvolvida funcionando na nuvem de computação da Amazon. Por fim, no capítulo 9 é feita uma retrospectiva, discussão, perspectivas de futuros trabalhos e uma conclusão desse projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Dado o escopo do projeto, é importante um detalhamento as capacidades das tecnologias que podem ser empregadas para essa solução. No ambiente de IoT há diferentes formas de conectar dispositivos a uma rede sem fio. E a partir dessa conexão, também há diferentes protocolos que podem ser utilizado para a comunicação como o HTTP, MQTT e UDP. Também para armazenar os dados temporais, é comum a utilização bancos de dados de séries temporais, ou TSDB.

2.1 IOT

IoT, ou Internet das Coisas, em português, é um um termo dado a aplicações embarcadas que comunicam entre si e/ou com a internet. Aplicações podem fazer uso de IoT tanto na interação direta com um usuário, quanto num contexto de um sistema que se comunica. IoT pode ser aplicada no contexto de dispositivos inteligentes como tomadas, lâmpadas e outros eletro-eletrônicos domésticos através da disponibilização, através do fabricante, de aplicativos para monitorar o uso ou controlar remotamente seu uso, por exemplo.

Também é possível observar a Internet das Coisas em sistemas maiores como uma casa inteligente, onde se pode ter uma aplicação centralizada como um ponto único para monitorar ou controlar vários dispositivos como lâmpadas, tomadas, câmeras.

Nas aplicações industriais, chamadas de Industrial Internet of Things (IIoT), é possível encontrar exemplos como os já citados (BHAGAVATHY; LATHA; THAMIZHMARAN, 2019; BARHATE *et al.*, 2019), em que medidas de sensores são disponibilizadas para sistemas centralizados e/ou aplicativos celulares. Nesses casos, podem ser utilizado para aumentar a automatização, melhorar o controle do estado do estabelecimento, monitorar anomalias para realizar manutenções preventivas, chegar a conclusões que levam a uma redução de custos.

Nos casos de comunicação com a internet, é comum se utilizar de protocolos HTTP e MQTT através de conexão de rede WiFi. Para comunicação local, porém, principalmente em situações em que a conectividade à WiFi pode ser intermitente e o número de dispositivos conectados pode ser muito alto, o uso de outros tipos de redes sem fio como Zigbee e Z-wave pode se provar útil. Nesses casos o uso de um serviço local como porta de entrada e saída para a comunicação com a nuvem pode ser útil, como num sistema de persianas automatizadas baseadas na iluminação (VARGHESE *et al.*, 2019) ou num sistema de tomadas inteligentes (FERNÁNDEZ-CARAMÉS, 2015).

Outra forma de conexão à internet para dispositivos IoT em seria pela rede 5G. (OUGHTON *et al.*, 2019) aponta que os custos de implementação da infraestrutura necessária para a rede são altos, principalmente para regiões urbanas. Há pesquisa

sobre o uso de IoT de banda estreita em 5G (5G NB-IoT) em casos de uso industrial ou de cidades inteligentes, que permite comunicação entre máquinas e entre máquinas e a internet (MUTEBA; DJOUANI; OLWAL, 2022).

2.2 SPI

Significa Serial Periphereal Interface. É uma forma de comunicação capaz de altas velocidades de transmissão de bits. Funciona com um mestre, dispositivo que inicializa as trocas de mensagens e que configura a frequência de transmissão a ser utilizada, e um número arbitrário de slaves. Para dispositivos master, são utilizados 2 pinos de transmissão de dados (Master-Out Slave-In (MOSI) e Master-In Slave-Out (MISO)), 1 pino de clock (System Clock (SCK)) e, para cada slave conectada, 1 pino de seleção de slave (Slave Select (SS)). Adicionalmente, é possível a utilização de mais de um pino de IO, para maiores taxas de transferências, como em (BARRY; CROWLEY, 2012).

2.3 HTTP

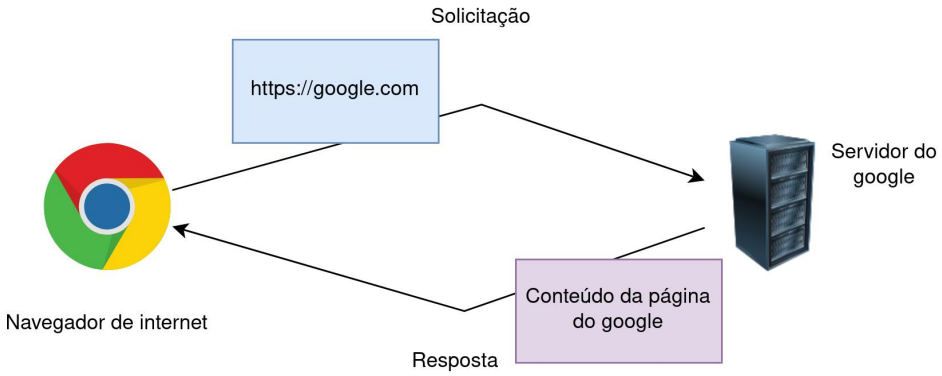
HTTP é um dos protocolos de aplicação mais utilizados na internet. Sua versão criptografada, HTTS, é a principal a ser utilizada para acessar websites e para comunicação entre sistemas na rede. Funciona num padrão de solicitação e resposta, no qual um cliente solicita um recurso ou envia um comando a um servidor, e esse servidor retorna uma resposta. No caso ilustrado na Figura 1 de um website, por exemplo, o Google, o cliente (o navegador de internet Google Chrome, Internet Explorer) faz uma solicitação ao servidor `https://google.com` por um recurso, e o servidor responde com o conteúdo que é o site do Google.

Esse protocolo funciona como uma camada sobre o protocolo TCP/IP. Por padrão não possui criptografia. Isso o faz muito inseguro para comunicação na internet aberta, e então é mais apropriado para redes fechadas confiáveis. Caso criptografia seja desejada, o protocolo HTTPS pode ser utilizado. Porém, https também tem custos adicionais de rede e processamento (NAYLOR *et al.*, 2014).

2.4 MQTT

“ MQTT é um protocolo de transporte de mensagens entre cliente e servidor, publicador/subscrito. É leve, aberto, simples, e projetado para ser de fácil implementação. Essas características o fazem ideal para uso em várias situações, inclusive ambientes restritos como comunicação entre máquinas (M2M) e contextos de IoT em que é necessário uso de pouco espaço para código e/ou largura de banda de rede é escarça. O protocolo executa acima de TCP/IP, ou outros protocolos de rede que provém conexões

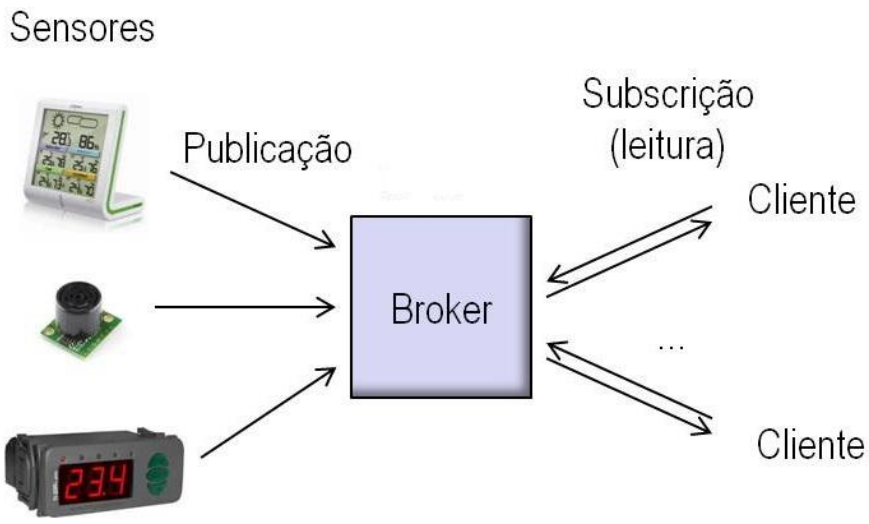
Figura 1 – HTTP funciona entre um cliente e um servidor, o cliente faz uma solicitação ao servidor, e este retorna uma resposta



Fonte: do Autor

ordenadas, sem perdas e bi-direcionais" - Tradução livre da definição do padrão MQTT ("MQTT. . . , 2015). Funciona através da conexão com um "broker", um servidor central, e cada dispositivo conectado pode se publicar mensagens ou subscrever a mensagens em um tópico. Logo, é possível a propagação de uma mensagem para mais de um recebedor, e cada dispositivo conectado ao broker também pode receber comandos, se estiver subscrito a um tópico. Tal funcionamento é ilustrado na Figura 2.

Figura 2 – Diagrama de funcionamento do MQTT. Publicadores enviam mensagens e subscritos recebem essas mensagens



Fonte: Internnet das Coisas: Middlewares e outras coisas - Scientific Figure on ResearchGate. Disponível em https://www.researchgate.net/figure/Figura-23-Esquema-de-funcionamento-do-MQTT_fig3_301298394

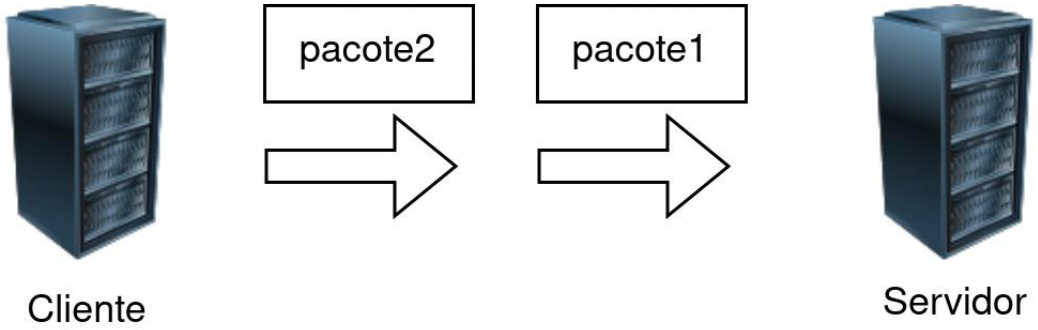
É mais seguro do que UDP e se beneficia de uma conexão de longa duração. Apesar de ser seguro para a informação, componentes no sistema são suscetíveis a ataques DoS (Denial of Service) (VACCARI; AIELLO; CAMBIASO, 2020).

Há protocolos derivados do MQTT que focam no aspecto de segurança, como SMQTT (SINGH *et al.*, 2015). O uso de MQTT em IoT é amplo, e pode ser evidenciado em (KODALI; SORATKAL, 2016).

2.5 UDP

Diferentemente do MQTT e do HTTP, o UDP é um protocolo que não possui conexão, a comunicação é unidirecional. Como ilustrado na Figura 3, os pacotes de dados são enviados sem esperar uma resposta, e não há garantia de que eles chegaram no destino. Tecnicamente, “datagrams” são enviados para um endereço de IP. Artigo mostra como o uso de UDP pode ser inseguro em IoT, e especificamente com a ESP32 (BARYBIN; ZAITSEVA; BRAZHNYI, 2019).

Figura 3 – Diagrama de funcionamento do UDP. Clientes enviam pacotes para servidores sem uma resposta



Fonte: do Autor

2.6 TSDB

TSDB (Time Series Database, ou Banco de dados para Séries Temporais) é um termo dado a banco de dados que são otimizados para lidar com séries temporais, ou seja, dados que contém em sua informação o tempo ou data em que aconteceram.

Algumas características que costumam aparecer nessa classe de banco de dados são: boa eficiência em inserção de dados com informação temporal recente, funções de agregação de dados baseado em um intervalo de tempo, cálculos rápidos entre dados com informação temporal próxima e boa eficiência para extração ou visualização de dados com informação temporal intervalos de tempo determinado.

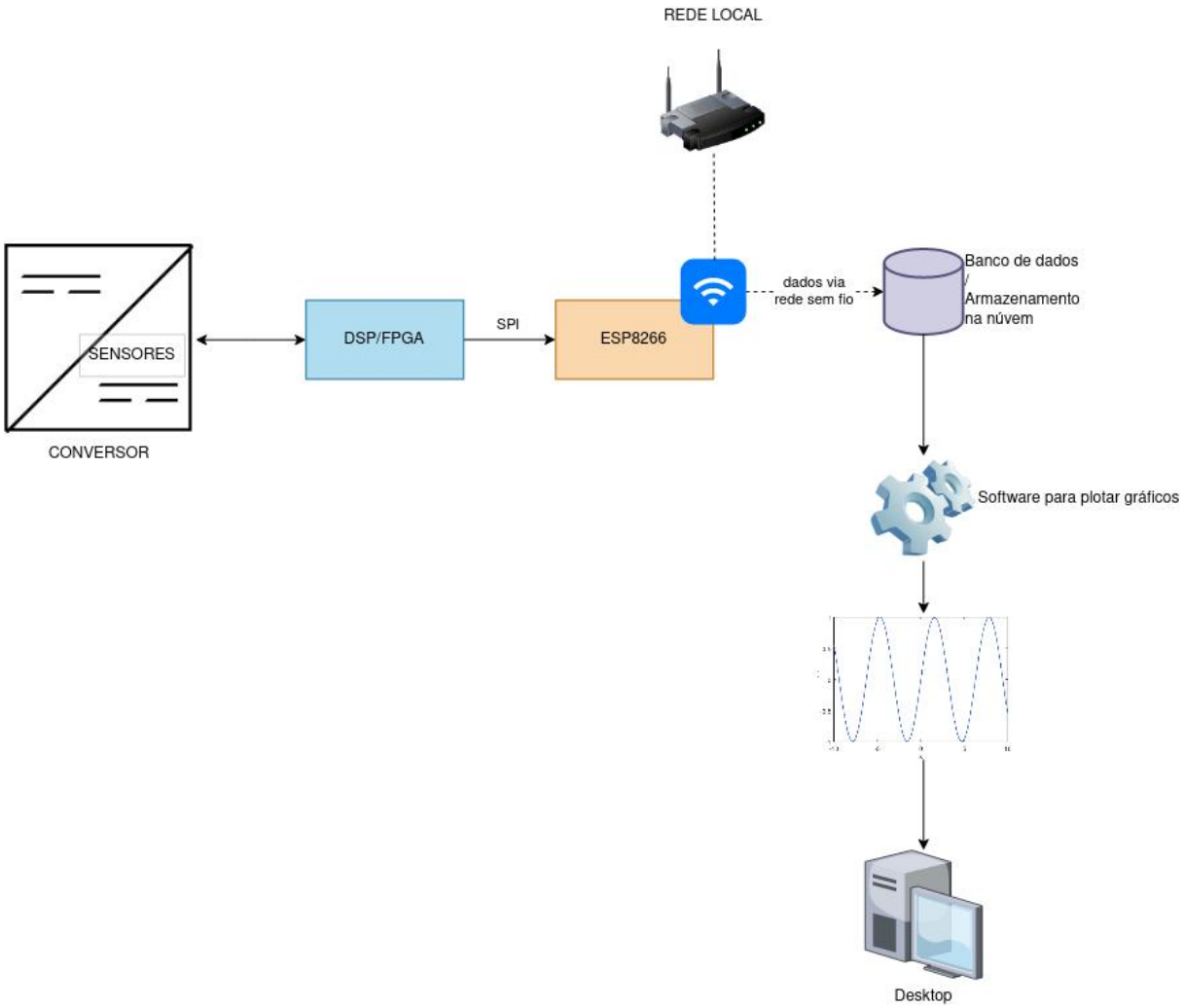
2.7 COMPUTAÇÃO NA NUVEM

Provedores famosos são Amazon Web Services (Amazon), Azure (Microsoft) e Google Cloud Platform (Google). Costumam ter precificação baseada no seu uso, disponibilidade de servidores virtualizados de uso geral de diversos tamanhos disponíveis em segundos, além de vários serviços próprios de uso específico. Ex: Bancos de dados, sistemas de filas, funções como serviço, armazenamento de arquivos (AWS S3).

3 DESIGN INICIAL DA ARQUITETURA

O design inicial da arquitetura foi pensado da seguinte maneira. A ESP se comunica com o FPGA através de SPI e envia os dados via WiFi para algum serviço na nuvem, como ilustrado na Figura 4. Entre as opções de armazenamento na nuvem há o Google Drive, AWS S3 e a utilização de um banco de dados.

Figura 4 – Design inicial da arquitetura. ESP8266 extraindo dados do controlador do conversor(FPGA ou DSP), enviando através da WiFi para algum banco de dados, em rede local ou remota, ou armazenamento na nuvem. Uma solução gráfica de para visualizar os dados armazenados



Fonte: do Autor

3.1 ENVIO PARA GOOGLE DRIVE

O Google Drive é conhecido como uma forma de pessoas e empresas guardarem e compartilharem arquivos. Embora a maioria das pessoas não utilizem, esse

serviço possui uma API Http (GOOGLE, s.d.) favorável ao uso de sistemas de informação.

Utilizando esse sistema como destino dos dados, a ESP deve criar um arquivo a cada quantidade grande de pontos, para que não se tenha milhares de arquivos novos por segundo. O formato utilizado seria provavelmente o Comma-Separated Values (CSV), no qual percebe-se colunas separadas por vírgulas, e fileiras separadas por linhas. Pelo fato de o Google Drive ser um sistema para armazenamento de arquivos, seria necessário escrever um software que baixasse esses arquivos, identificasse os dados, e então plotasse. Isso seria bastante trabalhoso e custoso e também dificultaria a visualização em tempo real de aplicação.

Além disso, o google drive não é muito voltada para sistemas, já que o upload deve ser de um arquivo inteiro. Para ler parte dos dados, como exemplo, baixar dados de uma hora específica no passado, seria necessário baixar todos os arquivos e ir comparando com a data que se deseja acessar. Essa solução teria performance muito abaixo do aceitável para a aplicação proposta.

Para auxiliar na performance de criação de arquivos, haveria a possibilidade de se ter um serviço intermediário recebendo dados da ESP e criando arquivos como necessário, arquivos de tamanho maior do que a memória da ESP permite. Porém ainda permaneceria o problema da plotagem e seria necessário ainda mais trabalho para desenvolver a solução.

3.2 ENVIO PARA AWS S3 OU GCP GCS

AWS S3 e GCP GCS são ambos “object stores”, onde são armazenados arquivos de qualquer formato em um caminho específico.

A diferença principal é que esses serviços são muito mais populares para o desenvolvimento de aplicações, então se encontraria muito mais integrações prontas. E também costumam ser de configuração mais simples.

Haveria também a possibilidade de utilizar o formato CSV. Caso fosse desejado simular uma tabela editável, há formatos como Apache Hudi (APACHE, 2021) . Porém requer o uso de frameworks pesados como Apache Spark (APACHE, 2018) ou Apache Flink (APACHE, 2014) . Caso seja desejada a habilidade de definir uma estrutura para os dados e tê-los comprimidos, a utilização do tipo de arquivo apache Avro (VOHRA, 2016) seria uma opção.

Como meio termo, uma estratégia a ser tomada poderia ser particionar os arquivos escritos por hora do evento. Logo haveria uma “pasta” para cada hora de dado. E se fosse desejado baixar dados entre 2 momentos de tempo, os arquivos de pastas com nomes com as datas inclusas seriam selecionados. A arquitetura então seria como ilustrado na Figura 5. Na qual o serviço “s3-inseridor” é desenvolvido para agregar vários pontos em uma submissão de arquivo.

Figura 5 – Arquitetura de envio para AWS S3 ou GCP GCS. Dados são enviados para o s3-inseridor, que envia para o arquivo apropriado dependendo de quando foi recebido. Uma pasta é criada para cada data, e uma subpasta é criada para cada hora

Fonte: do Autor

4 ESCOLHA DE FERRAMENTAS A SEREM UTILIZADAS

Para esse projeto, opções diferentes de bancos de dados e ferramentas de desenvolvimento em geral foram analisadas. Fatores considerados foram principalmente a popularidade, disponibilidade de uma versão gratuita, e familiaridade.

4.1 BANCOS DE DADOS

Como discutido no capítulo anterior, o uso de Google Drive ou S3 para armazenar dados não é prático. O caso de uso desse projeto é de armazenamento de dados temporais próximos ao tempo atual, então um banco de dados do tipo Time Series seria o que economizaria mais trabalho e daria uma performance mais aceitável.

Porém, pode-se também considerar outros tipos de bancos de dados. Fatores a serem considerados são popularidade, facilidade de gerenciamento, instalação, configuração, e a possibilidade de rodar localmente. Para rodar localmente, a ferramenta utilizada foi Docker, uma aplicação que isola processos diferentes em ‘contêineres‘, análogos a máquinas virtuais minimalistas (RAD; BHATTI; AHMADI, 2017).

4.1.1 Bancos de dados relacionais

Bancos de dados relacionais são caracterizados por possuírem tabelas com estrutura definida explicitamente, suporte a transações, e suporte a colunas de tabelas estarem relacionadas a outras tabelas. Exemplo: uma tabela de vendas ter uma coluna de Id do usuário fazendo referência à coluna Id da tabela usuário. Alguns dos bancos de dados relacionais mais utilizados são o PostgreSQL e o MySQL. Dada a generalidade desse tipo de banco de dados, certamente funcionaria para a inserção e leitura dos dados que são enviados, porém há muitas configurações que teriam que ser feitas para otimizar a escrita e a leitura. Seria necessário a criação de índices para uma rápida leitura, porém quanto mais índices, mais lenta é a escrita.

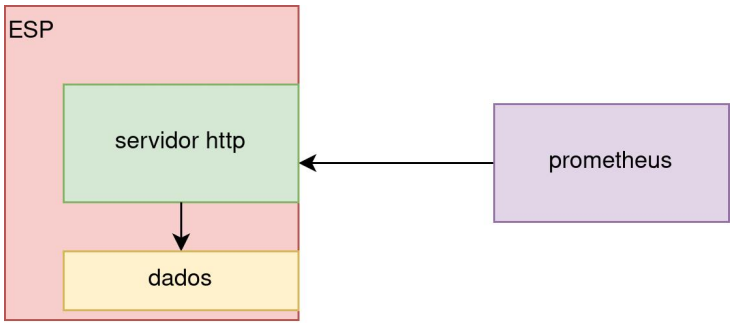
4.1.2 Outros bancos de dados timeseries

4.1.2.1 Prometheus

É um dos bancos de dados time series mais utilizados para métricas de serviços online. É feito primariamente para métricas do tipo "polling", onde o serviço do prometheus faz uma requisição a cada serviço que contém as métricas e armazena essa informações. Isso beneficia o serviço pois o próprio serviço de métricas decide a quantidade de dados que trafega pela rede e em que intervalo consulta cada serviço. Porém, para isso o Prometheus exige que o seu dado esteja agregado em várias formas na qual se deseja consultá-lo. No caso de se querer consultar a métrica de um serviço chamada "número de requisições recebidas", é necessário, no serviço, deixar

disponível: número de requisições recebidas nos últimos N segundos. Se informações sobre o tempo de resposta de um serviço forem desejadas, é necessário deixar disponíveis a média, mediana, p95, p99 de tempo de resposta do serviço nos últimos N segundos, minutos ou horas. Pela natureza de agregação, o Prometheus também não é recomendado para cálculos exatos. Seu foco então fica em métricas aproximas para serviços, onde a disponibilidade do banco de dados de métricas e a baixa influência na rede são muito importantes. Como prometheus precisaria acessar a ESP, um servidor HTTP na ESP seria necessário, como ilustrado na Figura 6.

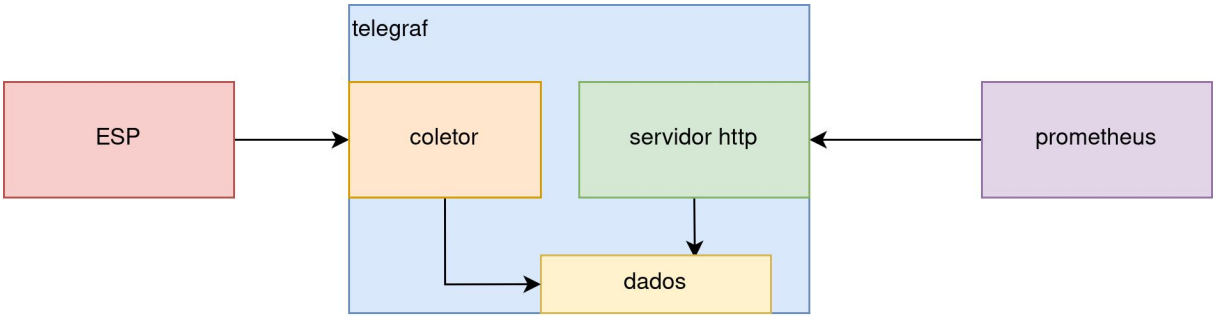
Figura 6 – Serviço de métricas prometheus consultando servidor http rodando na ESP para obter dados



Fonte: do Autor

Outra opção seria ter um servidor intermediário para o qual a ESP publicaria métricas, e o prometheus consultaria esse serviço, como ilustrado na Figura 7. Um serviço notável desse tipo é o telegraf `https://github.com/influxdata/telegraf`.

Figura 7 – Serviço prometheus consultando servidor http telegraf, que recebe dados da ESP



Fonte: do Autor

4.1.2.2 TimescaleDB

Outro banco de dados timeseries, feito como uma extensão instalável para o PostgreSQL. Essa opção também funcionaria, e provavelmente teria resultados similares a utilizar o InfluxDB, outro banco de dados TSDB, porém não queria passar por problemas de instalação de extensões. A empresa Timescale possui opção de rodar o TimescaleDB hospedada na nuvem, o que simplifica a manutenção e configuração. Para rodar localmente, também há a disponibilidade de uma imagem Docker. É menos popular do que o influxDB.

4.1.2.3 VictoriaMetrics

Segundo testes feitos pela própria empresa (VALIALKIN, 2018) , VictoriaMetrics é um dos banco de dados Time Series mais rápido do mercado, principalmente quando há uma grande dimensionalidade nas métricas. Outros pontos bastante fortes desse banco de dados é sua versatilidade. Aceita protocolos diversos tanto para escrita quanto para leitura: queries utilizando a linguagem do prometheus (PromQL), escrita utilizando formato do influxDB, graphite. Aceita HTTP e UDP.

Caso o InfluxDB não fosse rápido o suficiente, testes com o VictoriaMetrics seriam realizados. Provavelmente não necessitaria de muitas alterações já que aceita o mesmo protocolo.

4.1.2.4 InfluxDB

É um banco de dados com boa performance, de código aberto, com imagens Docker disponíveis rodar localmente. Possui boa documentação. Tem um procolo simples, uma interface gráfica aceitável para plotar os dados que estão nele, e nessa interface também há a possibilidade de exportar arquivos CSV com um intervalo de tempo selecionado. Encontra-se vários artigos científicos de aplicações utilizando esse banco de dados, como em sistemas de monitoramento de telescópios (ABARESHI; WILLIAMS; MARSHALL, 2018), e em (MARTINVIITA, 2018) foi comparado com TimescaleDB e se saiu à frente tanto em performance quanto em funcionalidade. Logo esse foi o banco de dados escolhido para armazenar os dados

4.2 AMBIENTE DE DESENVOLVIMENTO

4.2.1 Espressif IDF

Espressif IDF (IoT Development Framework), é um framework da Espressif para desenvolver aplicações IoT para placas da classe ESP32. Conta também com uma extensão para o editor de texto Vs Code. Esse framework garante maior controle sobre as camadas de baixo nível do software. Utiliza a linguagem de programação C prin-

cipalmente, em contrapartida a C/C++ no framework Arduino. Já que esse projeto de TCC começou utilizando a ESP8266, não havia acesso a esse framework. Mais tarde, como houve a necessidade de se utilizar a ESP32, havia a opção de se utilizar esse framework, mas pareceu necessitar de mais aprendizado até conseguir um protótipo, e haveria mais esforço para traduzir o código entre ESP8266 e ESP32.

4.2.2 Arduino

Conhecido por simplificar o desenvolvimento de software embarcado para entusiastas, Arduino é uma plataforma de desenvolvimento de alto nível, contando também com uma IDE com suporte a debug, seleção de placas, monitoramento de porta serial. Tem suporte a inúmeras placas e kits de desenvolvimento, como todos os kits da linha Arduino, além de ESP8266 e ESP32, que foram utilizadas nesse projeto. Possui várias bibliotecas genéricas disponíveis, delegando as especificidades das implementações para as bibliotecas específicas para cada placa de desenvolvimento. Algumas dessas bibliotecas genéricas são SPI, Http, WiFi. Além disso, por ser uma plataforma popular e de código aberto, há também bibliotecas compatíveis com o Arduino não mantidas pelo time oficial, mas por outras pessoas ou entidades independentes. Por exemplo, foi utilizada a biblioteca Ubidots, desenvolvida e mantida pela Ubidots. Também foi utilizada uma biblioteca de timer de hardware de código aberto.

Já que programa-se em C/C++, há menos preocupações de performance. Não é pago o preço de performance de linguagens de mais alto nível como javascript, lua ou python. Caso fosse necessário otimizar mais, provavelmente a nova opção de programação só seria em C, então não houve comprometimento à máxima optimização, mas é fácil encontrar ajuda online ou pessoas que passaram pelos mesmos problemas, exemplos de código, tutoriais. Por essas razões, foi o ambiente de desenvolvimento escolhido inicialmente, composto da IDE do Arduino e do framework Arduino.

4.2.3 Rust + Espressif IDF

Rust é uma linguagem de programação mais segura do que C/C++, tão eficiente quanto. A Espressif tem investido no suporte da linguagem em seu framework Espressif IDF, como evidenciado em: `https://github.com/esp-rs`. A configuração do framework com a linguagem pode ser um pouco complicado para placas ESP32 mais antigas, porém configuração pode ser mais simples para modelos mais recentes da placa que são feitos com suporte à linguagem de programação em mente (C3) `https://github.com/esp-rs/esp-rust-board`. Rust é a linguagem de programação mais amada segundo a pesquisa de 2021 do site StackOverflow `https://insights.stackoverflow.com/survey/2021`. também foi a mais amada em 2020 segundo a pesquisa do ano anterior `https://insights.stackoverflow.com/survey/2020`. Também recentemente Rust foi aprovada como segunda linguagem do kernel Linux `https://lore.kernel.or`

g/lkml/20211206140313.5653-1-ojeda@kernel.org/. Isso não possui precendentes, pois C++ e várias outras linguagens de baixo nível desde o surgimento do Linux já apareceram, mas nunca foram aceitas. C e assembly eram as únicas linguagens de programação permitidas naquele projeto.

Apesar dos pontos positivos, a recência das tecnologias e sua baixa popularidade significam que não seria tão fácil achar respostas para dúvidas. Aliado à falta de familiaridade do orientador e desse projeto com as ferramentas, poderia resultar em um alto esforço para revisar e estender o código produzido. Portanto, essas tecnologias não foram utilizada.

4.3 POR QUE NÃO UTILIZAR SOLUÇÕES MAIS PRONTAS PARA REALIZAR INGESTÃO DOS DADOS (GOOGLE IOT CORE, AWS IOT)

Há várias plataformas integradas de IoT disponíveis no mercado, que juntam ingestão, armazenamento, alerta, análise de dados de dispositivos. Algumas das mais famosas são oferecidas por provedores de serviços de computação na nuvem como GCP e AWS. Porém, pelo caráter de software como serviço desses sistemas, há alguns pontos negativos. Não há a possibilidade de rodar componentes dessas plataformas localmente para validar rapidamente o funcionamento de cada parte desse experimento. Como esse caso de uso pede 1000 pontos por segundo para uma placa, o que é consideravelmente alto para o número de dispositivos atuantes, e as plataformas costumam cobrar pelo número de requisições, há a chance de a conta da nuvem acabar ficando alta. Soluções de IoT costumam lidar com dados mais lentos (ex: temperatura do ar, nível de água numa caixa d'água) e de fácil agregação, ou eventos discretos (ex: Campainha da casa foi tocada). Então para esse caso de uso haveria pouco proveito das soluções oferecidas.

5 IMPLEMENTANDO O ENVIO DE DADOS

O envio de dados tomou prioridade à integração via SPI pelo fato de ser uma incerteza maior. SPI é utilizado em software embarcado há muito tempo, e as escolhas a serem tomadas e dúvidas de desempenho eram menores.

5.1 ESP8266 PARA INFLUXDB VIA UDP

Essa arquitetura foi a primeira testada. Possuía ótimas velocidade, contudo o protocolo UDP tem seus problemas, como não ser seguro (os pacotes são enviados sem criptografia. Qualquer roteador no meio do caminho entre cliente e servidor podem ler os dados). Por não ter conceito de conexão, há a perda de várias garantias presentes no TCP. UDP não garante ordenamento de pacotes, não há resposta para confirmar se o dado foi recebido. Além disso, alguns modems bloqueiam esse tipo de pacote, já que um grande tráfego UDP pode ser confundido com ataques de Denial of Service, como evidenciado em `https://resources.sei.cmu.edu/asset_files/WhitePaper/1996_019_001_496172.pdf#page=5`. Também não respeita congestionamento da rede, usa toda a banda disponível, independente de outras aplicações na mesma rede. Nessa etapa também é feita uma conexão a uma rede local via WiFi, utilizando a biblioteca do Arduino. É necessário apenas providenciar o nome da rede(SSID) e a senha. O InfluxDB foi configurado utilizando a ferramenta Docker, executando no mesmo computador em que o código era desenvolvido. Já que tanto a ESP8266 quanto o computador de desenvolvimento estavam na mesma rede WiFi, o endereço de IP local do computador de desenvolvimento foi utilizado na ESP8266 como destino dos dados.

5.2 UTILIZANDO A PLATAFORMA UBIDOTS

A plataforma de IoT da Ubidots foi testada num momento de estagnação do andamento do projeto. Essa plataforma possui integração através de vários protocolos. HTTP, MQTT. Foi utilizado MQTT, através de uma biblioteca do Arduino + ESP8266 disponibilizada pela Ubidots. A plataforma permite envio de comandos através da interface, permite a possibilidade de plotar dados de diversos dispositivos, montar quadros para visualização, alertas.

5.2.1 Utilizando a ESP32 para realizar o envio

Tendo incluído a biblioteca de timer de hardware, ‘ESP8266TimerInterrupt‘, e a biblioteca Ubidots no mesmo projeto na ESP8266, erros ao conectar à WiFi foram encontrados. Pesquisas na internet por casos de erros similares não retornaram resultados. Utilizar a biblioteca PubSubClient diretamente para enviar os dados via MQTT

também não resolveu o erro. O mesmo erro aconteceu utilizando outra placa ESP8266, então a possibilidade de ser um problema de hardware foi descartada.

Dado que um kit de desenvolvimento de mesmo fabricante, ESP32, já havia sido disponibilizada para o projeto, ela foi utilizada para substituir a ESP8266 como microcontrolador para realizar o envio. Outra opção teria sido utilizar um framework diferente, porém alterar o kit de desenvolvimento utilizado pareceu uma opção mais simples. Como a plataforma Arduino foi utilizada, poucas partes do código tiveram que ser alteradas, notavelmente o include e a função de conexão à WiFi. A rotina de interrupção utilizada foi a de timer de hardware `https://github.com/khoih-prog/ESP32TimerInterrupt#why-using-isr-based-hardware-timer-interrupt-is-better`. Por poder interromper outros trechos do código. Como a aplicação desse projeto exige precisão a nível de milissegundos, isso se torna bastante importante e vantajoso.

5.2.2 Desafios encontrados ao utilizar a plataforma Ubidots

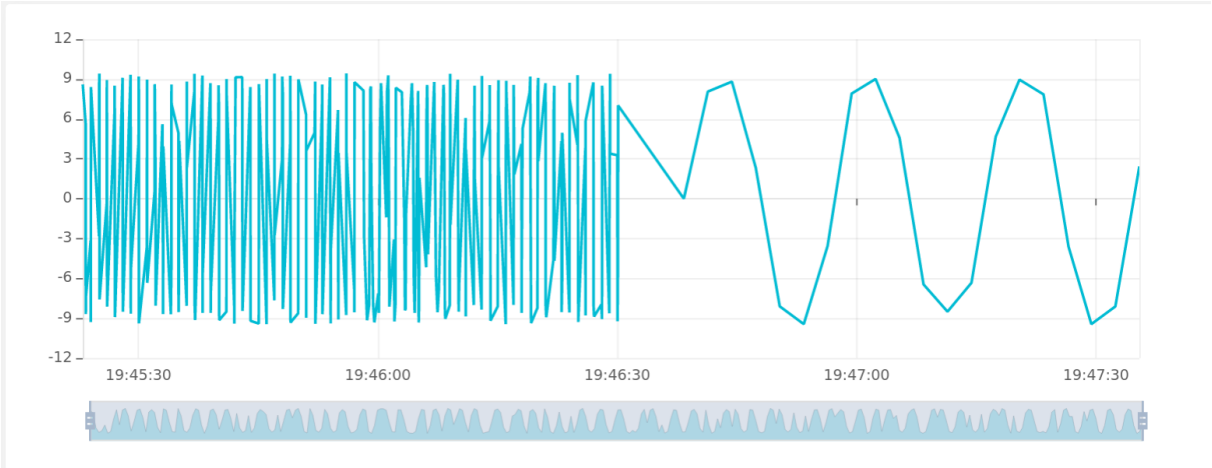
A plataforma Ubidots parecia promissora, porém havia algumas limitações para o que era desejado atingir. Uma delas é uma capacidade de transferência máxima de dados baixa. O formato utilizado para envio de mensagem (JSON) tinha um cabeçalho grande e a biblioteca só permitia uma mensagem que cabia 5 pontos no máximo. Tão relevante quanto isso é o fato de só ser possível uma precisão de 1 ponto por segundo para a leitura dos dados. Caso seja feito o envio de 5 pontos com o mesmo segundo, mas milissegundos separados, apenas o último ponto (ordenado pelo tempo) será armazenado. Isso não é problema para aplicações que lidam com variáveis lentas como temperaturas, mas para esse projeto não é o suficiente. Uma ilustração dessas limitações se dá na Figura 8. À esquerda observa-se dados enviados a uma frequência acima de 1 ponto por segundo. Não há continuidade dos dados. À direita se vê dados enviados a uma frequência de 1 dado por segundo, uma senoide de período de 10 segundos.

Também não é possível exportar dados armazenados no Ubidots para análise posterior. Nesse momento foi estabelecido como requisito para a solução uma performance de ingestão de 1000 pontos por segundo. No caso da aferição de uma tensão alternada a 60Hz seria o suficiente para se ter informações extraíveis até a 8a harmônica (480Hz) segundo o teorema da amostragem de Nyquist–Shannon.

5.3 TROCANDO O DESTINO DOS DADOS PARA INFLUXDB AO INVÉS DE UBIDOTS

Então foi decidido voltar a utilizar o InfluxDB. Que dessa vez foi notado que em sua interface gráfica de gerenciamento de banco de dados também tinha funções de plotagem e de extração de fatias de dados em formato CSV. Para validar a ve-

Figura 8 – Envio para plataforma Ubidots. À esquerda observa-se dados distorcidos devido ao envio mais frequente do que permitido pela resolução máxima. À direita se vê dados mais lentos que não foram distorcidos



Fonte: do Autor

locidade do InfluxDB, foi escrito la linguagem de programação python um programa que enviava 1000 pontos a cada segundo para o influxDB localmente. O InfluxDB foi configurado utilizando a imagem Docker publicamente disponível, e o resultado foi um tempo de resposta de 100 a 150 ms. Utilizando a ferramenta "tc"(Traffic Control) `https://bencane.com/2012/07/16/tc-adding-simulated-network-latency-to-your-linux-server/`, foi adicionado um atraso de rede de 100 ms à interface de rede virtual que se conecta ao contêiner onde o processo do banco de dados executa para simular uma latência de rede de um caso onde o influxDB estivesse em um data center ou nuvem, e o tempo de resposta total ficou em certa de 320ms. O fato de um aumento de latência de 100 ms aumentar o tempo total de resposta em mais de 100 ms já é esperado, já que a relação de latência com tempo de resposta em comunicação HTTP não é uma equação linear.

5.4 ADICIONANDO INFORMAÇÃO TEMPORAL AOS DADOS

Se não é informada a dimensão de tempo nos dados que são enviados ao InfluxDB, ele adicionará a informação de tempo baseado em quando os dados foram recebidos. Como vários dados que foram aferidos em tempos diferentes serão enviados em uma mesma requisição, é necessário adicionar a informação de tempo em cada um desses. Já que a ESP32 não possui real time clock, nem um relógio atômico, servidores de tempo de rede (NTP) podem ser utilizados. O tempo real é obtido de um conjunto de servidores NTP localizado no brasil (`br.pool.ntp.org`) logo após a inicialização da WiFi.

Uma relação é feita entre o tempo retornado pelos servidores NTP e número de microssegundos desde que a placa foi inicializada (retornado pela função micros()). Em partes seguintes do código, para calcular o tempo atual, calcula-se a diferença entre o retorno de micros() e valor de micros() quando o tempo dos servidores NTP foi recebido, obtendo-se o número de microssegundos que passaram desde a consulta ao tempo dos servidores NTP. E essa quantidade de microssegundos é somada ao tempo que foi obtido na consulta do tempo aos servidores NTP.

O timestamp referência não é atualizado ao longo da vida da ESP32 para evitar troca de ordenamento de pontos entre envios diferentes. No caso de uma atualização de o tempo real ter sido diminuído de 100 milissegundos, o envio seguinte sobrescreveria a os 100 últimos pontos do envio anterior.

6 IMPLEMENTANDO COMUNICAÇÃO SPI

Para tornar mais ágil a implementação da comunicação SPI, foi decidido utilizar uma placa de desenvolvimento para simular o controlador do conversor. Já que havia recebido uma ESP8266 e uma ESP32 para realizar o projeto, a ESP8266 foi utilizada.

6.1 TROCANDO O AMBIENTE DE DESENVOLVIMENTO PARA PLATFORMIO + VS CODE

Assim que comecei a tentar programar para duas placas diferentes, alternando entre elas, mais uma limitação da IDE do Arduino se evidenciou. A alteração do modelo de uma placa em uma janela da IDE é aplicada em todas as outras janelas. É necessário utilizar máquinas virtuais para conseguir isolar os ambientes, ou ter diferentes instalações. Como já estava descontente com o editor de código não ter muitas utilidades como auto-completar, pular para definição e outras funcionalidades tidas como garantidas para desenvolvimento de software, decidi procurar outra opção.

Uma opção que já havia passado em minhas buscas, mas não havia me comprometido era o PlatformIO. É uma extensão para o editor de texto VS Code que integra com diferentes frameworks, sendo Arduino um deles. Um projeto PlatformIO já vem organizado com uma estrutura de pastas separado em src, lib e test. Na pasta src deve estar o arquivo main, na pasta test devem estar os testes unitários a serem executados.

Para programar duas placas diferentes, é possível abrir uma janela do vscode em cada pasta de cada projeto, ou abrir as duas pastas na mesma janela e especificar para qual projeto se deve rodar cada comando (compilar, upload de código, testes unitários, monitorar serial). Outra utilidade é ter na configuração do projeto quais bibliotecas são necessárias para compilar, além do framework utilizado. No Arduino a importação de bibliotecas não se faz por arquivos de configuração, mas por configurações dentro da IDE. Ter a configuração como arquivo ajuda para o caso de compartilhar código, de desenvolver em conjunto com outras pessoas, e também no caso de troca de computador da mesma pessoa.

Outras funcionalidades úteis são poder navegar para a definição de uma função, classe, variável ou macro ao apertar Ctrl + clique com o cursor do mouse em cima do símbolo.

Essa troca se provou bastante economizadora de futuro stress, pois com testes automatizados houveram menos casos em que fosse necessário debugar o funcionamento do código, além de erros de compilação serem mais legíveis.

6.2 MONTAGEM

Como ambas as placas de desenvolvimento possuem alimentação via USB, é possível conectar seus pinos de tensão de entrada (Vin) e terra (GND) para fazer com que a alimentação de tensão de entrada de uma placa seja replicada para a outra. Como não há regulador de tensão entre a entrada da USB e o pino Vin, não há riscos de se sobrecarregar algum componente da placa que seja alimentada primariamente.

Para conectar as placas via SPI, é necessário conectar MOSI com MOSI, MISO com MISO, SCK com SCK e SS com SS. Um ponto a se notar é que a ESP32 possui duas interfaces SPI disponíveis para uso geral. HSPI e VSPI, ilustrado na Figura 9. Ambas tem as mesmas possibilidades e capacidades, mas pinagens diferentes. A interface SPI utilizada na ESP32 foi a VSPI, por ser a padrão. Como ponto de partida utilizando 1 Mbit/s para a frequência do clock.

Sobre a placa destinada ao envio de dados ao influxDB(ESP32) ser configurada como mestre ou slave não faz tanta diferença. Todos os comportamentos nesse projeto tem uma relação dual. Para fazer a comunicação via SPI, tendo Esp32 como slave, esta configuraria uma função de reação a novos dados vindo da master e receberia os dados da ESP8266. Se a Esp32 for configurada como mestre, esta pode ser configurada com uma interrupção de hardware na qual requisita para a slave os dados a cada 1ms.

O diagrama de blocos da montagem utilizada está na Figura 10. À esquerda observa-se a ESP32, à direita está a ESP8266, configurada como slave, simulando o FPGA controlador do conversor.

6.3 PROTOCOLO BASE POSSÍVEL PARA SPI

A ESP8266 possui um protocolo SPI definido possível para funcionar como slave que se integra a interrupções de hardware e já tem biblioteca disponível. Há a escolha entre utilizar esse protocolo existente ou o desenvolvimento de um protocolo diferente, utilizando diferentes tamanhos de mensagens. Como implementar um protocolo próprio seria muito custoso em desenvolvimento, e o protocolo existente é suficiente para a aplicação desse projeto, a escolha foi de utilizá-lo.

Na referência técnica do chip, há a informação da presença de comandos SPI de ler o status, escrever dados e ler dados `https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf`. Para ler dados, envia-se 2 (primeiro byte para o comando (0x01), segundo byte para o endereço (0x00))bytes e se lê 32 bytes. Para escrever, envia-se 34 bytes. Primeiro byte para o comando (0x02), segundo byte para o endereço (0x00), os 32 bytes restantes são para a informação que será escrita no dispositivo slave.

Um ponto a se notar é que esse protocolo funciona a nível de vários bytes.

Esse protocolo pode ser utilizado sem muitos problemas, já que a ESP32 tem 240MHz de cpu e só é necessário uma requisição SPI a cada 1ms, mas caso seja desejado economizar transferência em SPI e só haja necessidade de 8 bytes de resposta para a leitura, por exemplo, o restante seria desperdiçado. Ha atual implementação a resposta ao comando de leitura tem o tamanho de 17 bytes, então há um desperdício de 15 bytes.

6.4 FORMATO DAS MENSAGENS SPI

Então por fim foi implementado o seguinte formato de mensagem para comunicação: O comando de escrita foi utilizado para setar o tipo de informação que a master deseja ler da slave. O comando de leitura foi utilizado para receber os dados do formato escolhido. Os dados a serem lidos são 2 números do formato ponto flutuante de dupla precisão (binary64/double precision float), conforme IEEE-754 `https://ieeexplore.ieee.org/document/8766229` representados cada um por seus 8 bytes, juntos a um byte de checksum. O formato de dados na leitura é ilustrado em Figura 11, no qual a master faz um pedido de leitura, e recebe os valores 1,23 e 2,01, com o checksum tendo valor hexadecimal 45, 69 em valor inteiro.

6.5 ROTINA DE LEITURA SPI

A rotina interrupção de leitura é disparada a um intervalo configurável e preenche uma posição num buffer de tamanho configurável. Após preencher essa posição, incrementa a variável da posição, para que a próxima posição seja ocupada na próxima vez que a rotina for acionada. Para sinalizar à rotina de envio de dados para o InfluxDB que o buffer está cheio, outra variável é utilizada, mas dessa vez booleana.

6.6 DETALHES SOBRE O USO DO CHECKSUM

O checksum utilizado é bastante simples, vindo de `https://forum.arduino.cc/t/simple-checksum-that-a-noob-can-use/300443/3`. Esse algoritmo simples permite uma confiabilidade melhor caso o dado tenha sido corrompido entre aferição e o resgate. Sobrescrita do registrador da slave durante a leitura e erros de bits durante a transmissão devido a ruído são os candidatos principais desse tipo de erro.

O algoritmo faz uma soma dos valores dos 16 bytes a serem enviados pela 8266 em módulo 256 (valor máximo de um byte) e o envia como 17° byte. A ESP32 então calcula o checksum dos 16 primeiros bytes recebidos e os compara com o 17°. Por ser uma soma, no caso de todos os bits enviados serem zerados, o resultado também será zero.

Então esse algoritmo em si não aponta o erro do caso de a slave estar incomunicável. Porém, devido a aplicação fazer envios de números de ponto flutuante, uma aferição de dados exatamente iguais a zero é muito improvável. Então foi considerado que esse caso (todos os bits recebidos serem iguais a zero) constitui um erro de comunicação. Há a possibilidade de deixar o protocolo um pouco mais explícito, porém. Fazendo algo como setar algum byte com valor constante sempre, ou adicionando um valor constante não zero ao algoritmo de checksum.

Nota-se que esse algoritmo não é garantia de segurança. É trivial, para um agente malicioso, querendo interferir com as medições, gerar diferentes bytes cuja soma é igual. Porém para a aplicação desse projeto, em que há uma confiança na fonte dos dados SPI por ter que estar fisicamente conectada, e que a fonte própria gera o checksum na mesma transação em que envia os dados, e o fato de que não é feito mais nada com o checksum, deixam esse algoritmo tão útil quanto algo mais complexo como CRC, ou uma função de hash criptográfico SHA.

Outra opção além de implementar o checksum seria fazer o uso de alguma codificação de sinal que fosse auto-corrigível, como por exemplo o código de Hamming (HAMMING, 1950). Porém, como o meio de transmissão SPI é pequeno o suficiente, supõe-se que não haveria erros frequentes o suficiente para justificar o aumento de banda associado a esse uso.

6.7 EVITANDO A CORRUPÇÃO DA INFORMAÇÃO

Caso o slave escreva no registrador de leitura enquanto o master está lendo, um dado errado pode ser lido, como ilustrado na Figura 12. Isso acontece pois a ESP8266, slave, estava configurada para periodicamente atualizar o buffer utilizado para disponibilizar dados para leitura, e essa alteração de buffer não é sincronizada com outros acessos.

A maneira mais simples de remediar isso seria atualizando o valor logo após ele ser lido, então se supõe que haverá um tempo grande até o valor ser lido novamente, como na Figura 13. Essa foi a estratégia utilizada no restante desse projeto.

Essa estratégia não é muito boa pois no caso de uma leitura ser atrasada, representado na Figura 14, será lido um valor bastante defasado temporalmente, e como o tempo da medição é atribuído relativo a quando a master recebe o dado, haverá uma distorção da forma de onda.

Uma solução mais drástica seria inverter as posições. Então a ESP32 ficaria como slave, recebendo comandos de escrita, e a master (controlador do conversor, simulado pela ESP8266) enviaria periodicamente. Essa estratégia evita esse bug e acaba ficando uma estratégia "melhor esforço"para a mestre. Apesar de não haver na biblioteca oficial da ESP32 uma forma de usá-la como slave em SPI, há uma biblioteca open-source `https://github.com/hideakitai/ESP32SPISlave`. A API em si

é bastante diferente da utilizada para a ESP8266. Parece haver problemas em utilizar essa biblioteca para envios maiores do que 8 bits, porém. Evidenciado em `https://github.com/hideakitai/ESP32SPISlave/issues/7`. E no caso de não ser possível utilizar a biblioteca, poderia fazer sentido alterar as posições e fazer da ESP8266 novamente a placa para envio de dados para o InfluxDB, já que não haveria mais a incompatibilidade entre bibliotecas.

Outra maneira seria sempre enviar um comando de atualizar o valor antes de um comando de leitura. Assim a slave nunca atualiza o previamente no registrador. Isso se torna um protocolo mais sofisticado e com mais garantias, mas requer mais instruções. Como esse é um caso com dados numa dimensão de milissegundos, esse aumento na duração da troca de informações pode ser custosa.

6.8 RESULTADOS PARCIAIS

A ESP8266 foi configurada para retornar dados simulando duas senoides de 60Hz, com 3a e 6a harmônicas, defasadas de 90 graus, e de amplitudes idênticas.

Então, plotando as formas de ondas como armazenadas no banco de dados InfluxDB, ilustra-se a Figura 15. Percebe-se que as formas de ondas estão distorcidas. Quando há um aumento do tempo analisado, na Figura 16 também observa-se que há um atraso grande (cerca de 9 segundos) entre envios diferentes. Isso se deve ao fato de apenas um buffer ser utilizado para realizar o envio, e enquanto este está sendo enviado, o recebimento de dados da slave fica pausado.

6.9 OTIMIZANDO O ENVIO DE DADOS

O algoritmo implementado para formatar o texto era ineficaz por usar em parte a classe ‘String‘ da biblioteca do Arduino e em parte funções básicas de C como ‘sprintf‘. Para otimizar isso, utilizei concatenações com a funcionalidade ‘stpcpy‘ da biblioteca ‘string.h‘. E também pre-aloquei um buffer utilizando ‘malloc‘ com uma capacidade acima da esperada para utilização. Como a classe ‘String‘ do arduino realoca um buffer quando ele chega a seu limite, sucessivas concatenações podem ser muito custosas. Como o envio é realizado com 1000 linhas diferentes de dados por vez, dependendo do algoritmo utilizado para redimensionar o array, o custo disso não é desprezível.

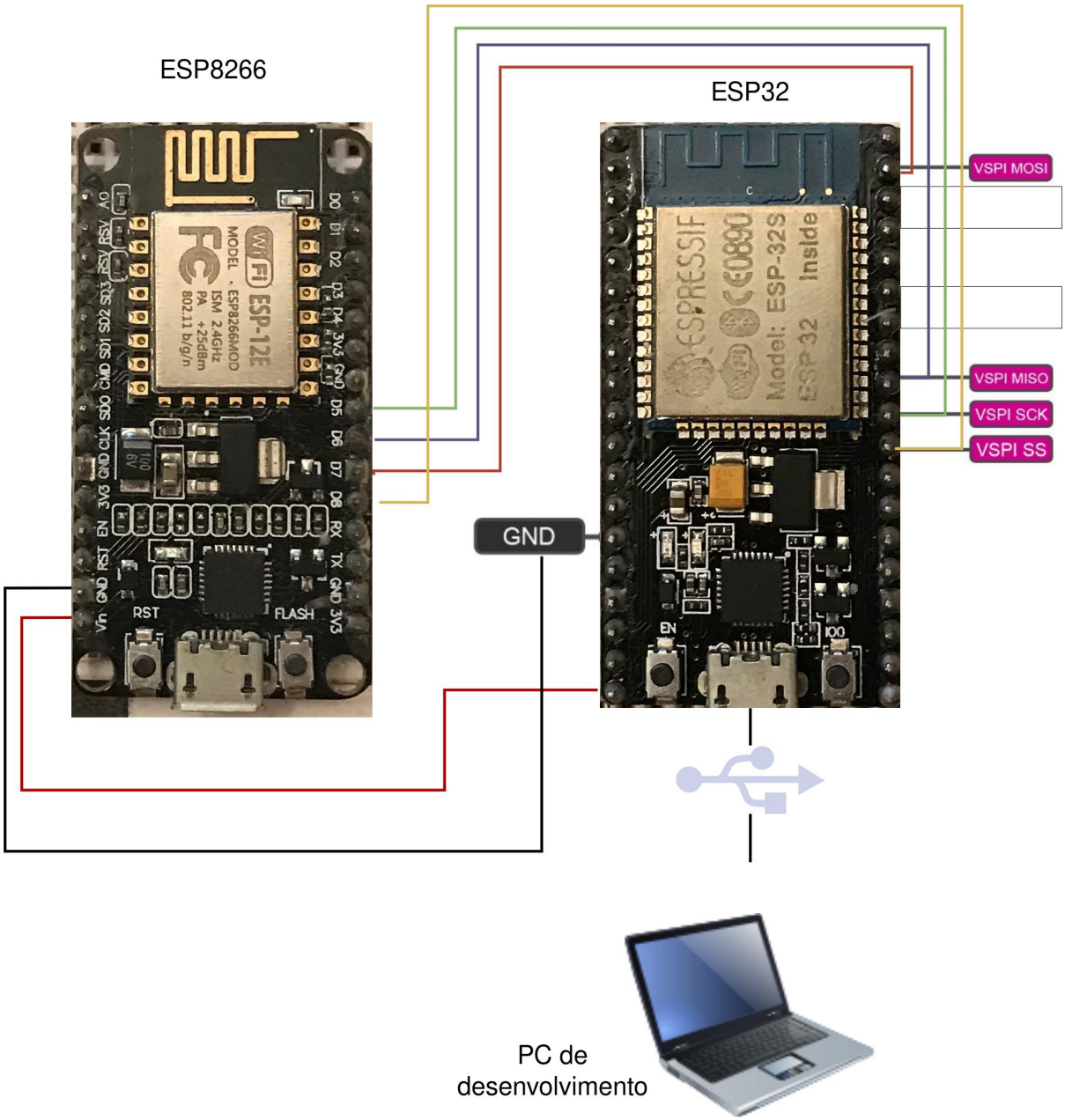
Além disso, estava enviando para a porta serial toda a mensagem formatada, como forma de me ajudar a monitorar o sistema. Dado que a carga de dados é na ordem de dezenas de kilobytes e que a comunicação serial estava configurada para utilizar uma baud rate de 115200, havia um atraso considerável. 115200/8 = 14400 bytes/segundo. Então só para imprimir os dados no terminal, havia a possibilidade de gastar mais de 3 segundos. O resultado dessas otimizações se dá na Figura 17.

Nota-se claramente que ainda há janelas de tempo em que não há dados, porém estes são de menos de 1 segundo. Para diminuir, ou até acabar com essas janelas, há a possibilidade de alterar a arquitetura dessa comunicação. Haveriam 2 buffers, e assim que um deles ficasse cheio, esse seria travado de escrita, encaminhado para envio ao influxDB. E enquanto o envio acontece, o outro buffer estaria sendo cheio pelos dados vindos da slave. Seria importante continuar usando apenas uma rotina de interrupção, porém, para haver uma sincronia dos dados.

6.10 FLUXOGRAMA DO CÓDIGO IMPLEMENTADO

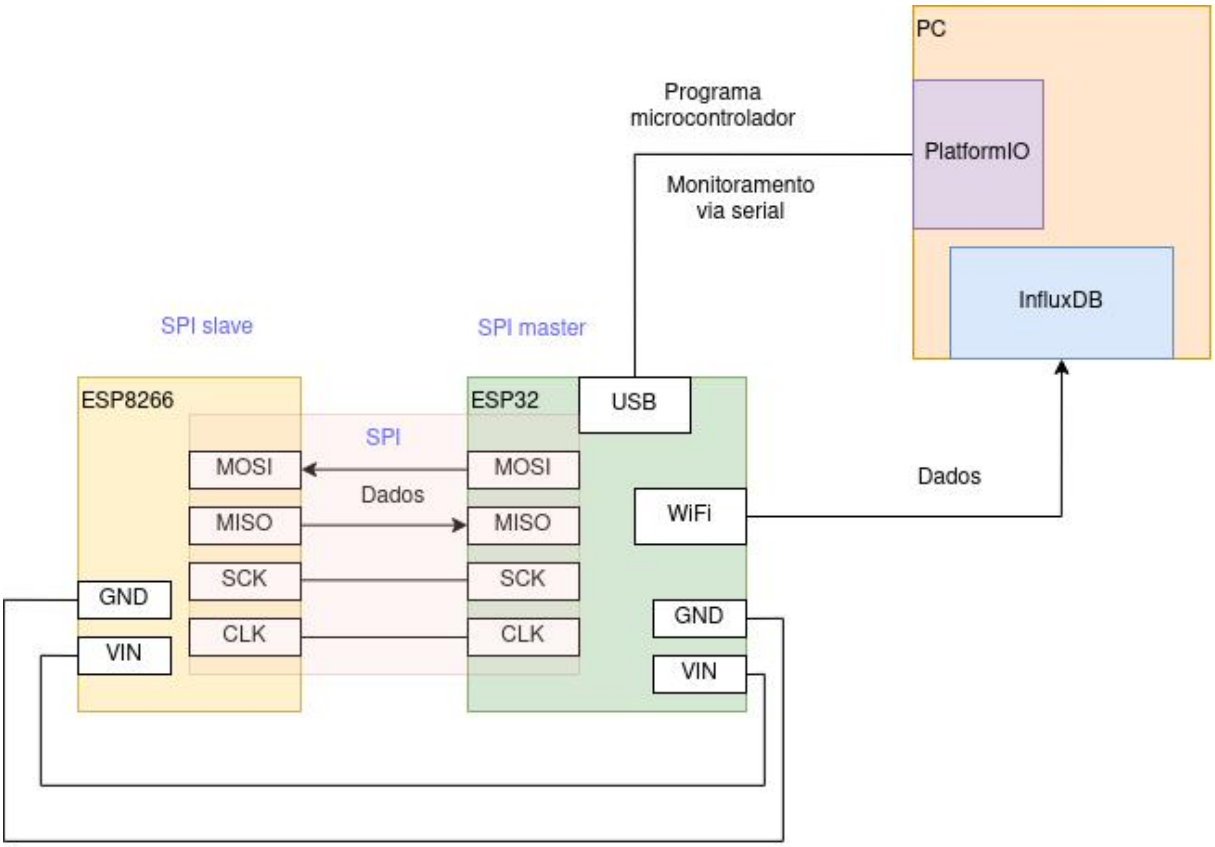
Um diagrama de blocos sobre como funciona o código da ESP32 participando como master na comnicação SPI nesse ponto do projeto está na Figura 18.

Figura 9 – Diagrama de conexão entre os pinos da ESP8266 e ESP32. Os pinos MOSI e MISO, SCK e SS de cada placa são conectados para a comunicação SPI. Os pinos Vin e GND são conectados para alimentar a ESP8266 através da ESP32



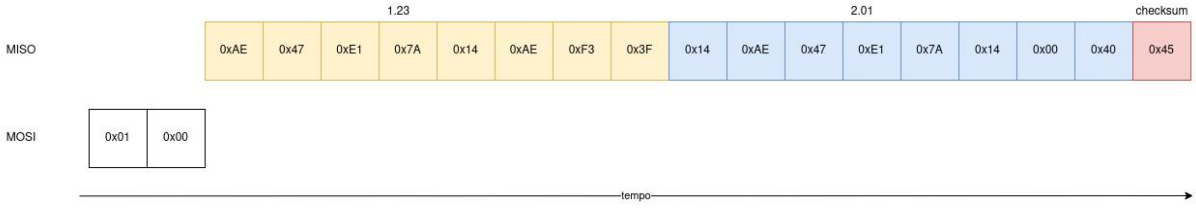
Fonte: do Autor

Figura 10 – Diagrama de blocos de montagem de desenvolvimento. ESP8266 e ESP32 se comunicam via SPI, ESP8266 alimentada transitivamente pela ESP32. ESP32 envia dados ao InfluxDB hospedado no computador numa rede local



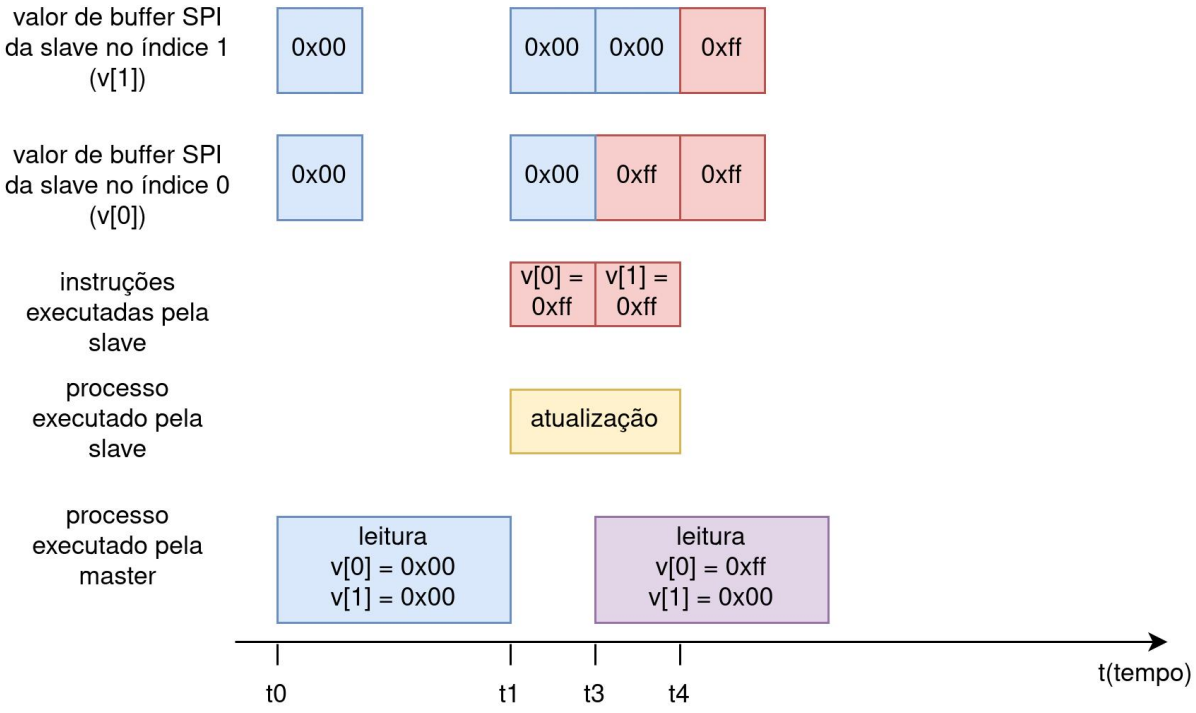
Fonte: do Autor

Figura 11 – Exemplo de mensagem SPI de leitura. Master envia 2 bytes que sinalizam leitura através do MOSI, slave envia 17 bytes de informação. 8 bytes para cada número e 1 byte para checksum



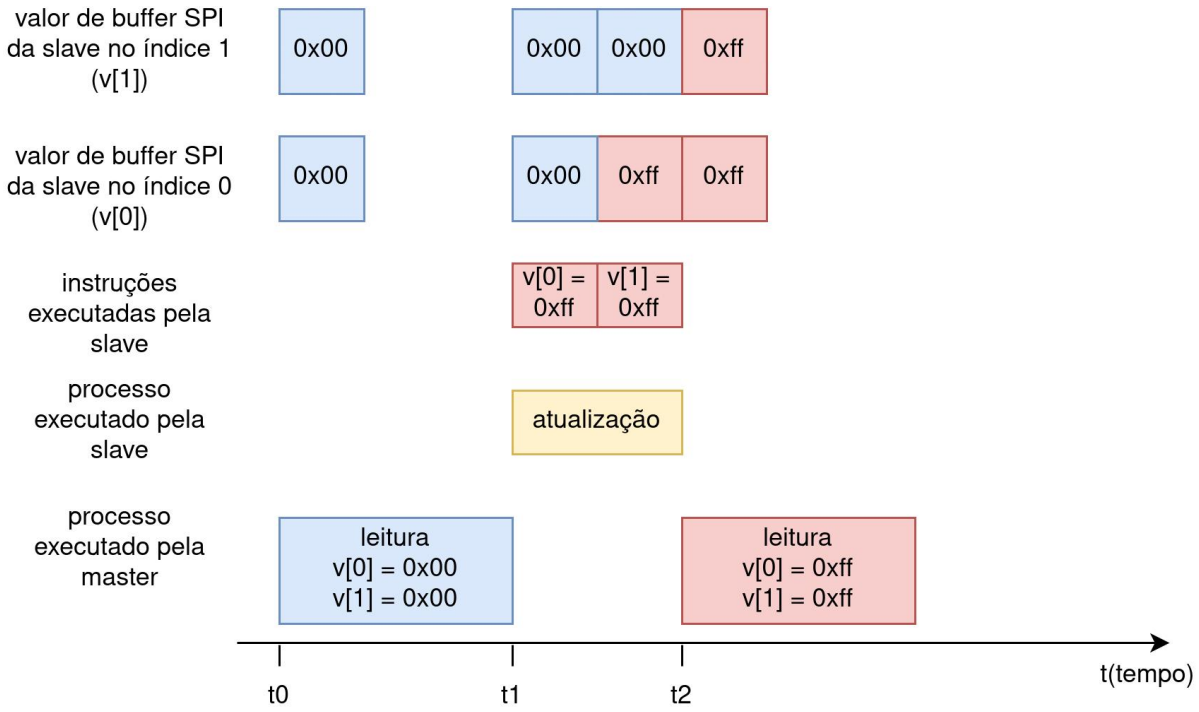
Fonte: do Autor

Figura 12 – Diagrama de leitura corrompida. em t1 a atualização do buffer SPI começa na slave, em t3 a master começa a leitura, porém nem todos os itens do buffer da slave estão atualizados. Então a leitura é de um estado inválido. Representado a nível de byte, porém erro também pode acontecer a nivel de bit



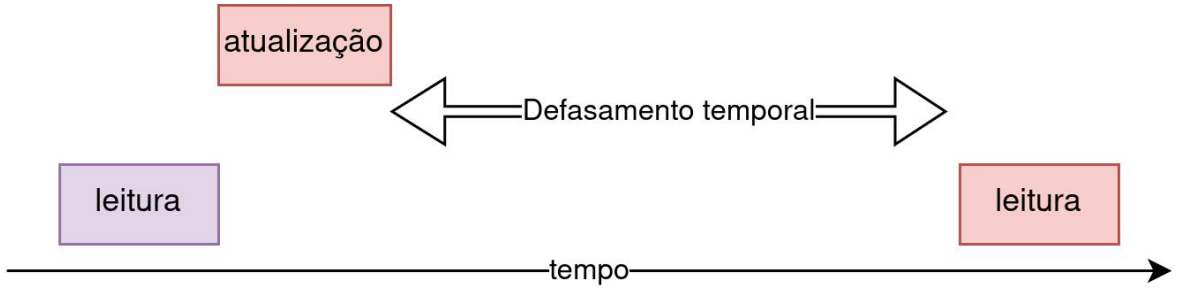
Fonte: do Autor

Figura 13 – Atualização do buffer após leitura. tendo que o intervalo entre leituras (t2 - t1) é maior do que o intervalo necessário para a slave atualizar seu buffer, acontecerá que a leitura sempre será de um valor válido



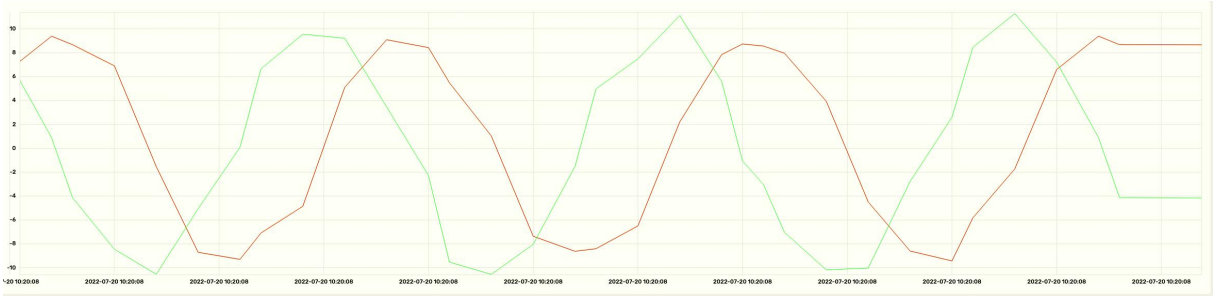
Fonte: do Autor

Figura 14 – Atualização do buffer após leitura. Em casos de leituras muito esparsas, serão lidos valores que foram atualizados no tempo da última leitura, o que pode ser um valor defasado



Fonte: do Autor

Figura 15 – Visualização no InfluxDB de uma senoide e uma cossenoide de 60Hz, com 3ª e 6ª harmônicas

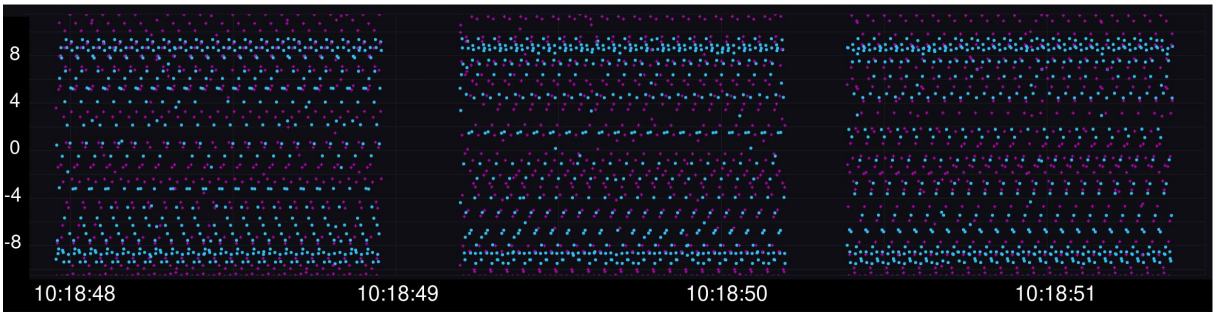


Fonte: do Autor

Figura 16 – Intervalo entre diferentes envios é considerável, maior que 1 segundo

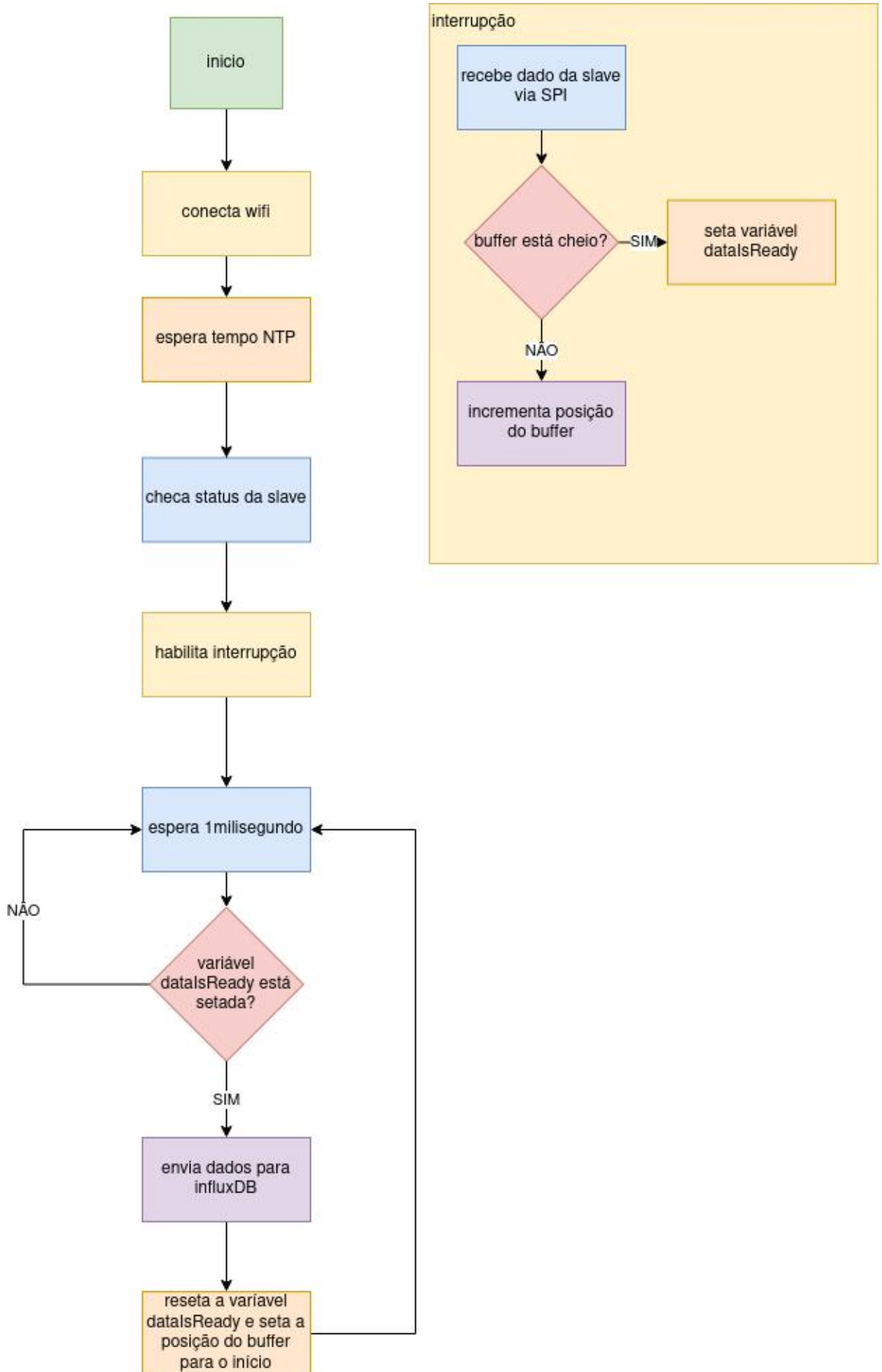
Fonte: do Autor

Figura 17 – Intervalo entre envios foi diminuído após otimizações, na ordem de 200 a 600 milissegundos



Fonte: do Autor

Figura 18 – Diagrama do código da ESP32 funcionando como master na comunicação SPI, coletando dados da ESP8266 e enviando para InfluxDB



7 IMPLEMENTANDO COMUNICAÇÃO VIA MQTT

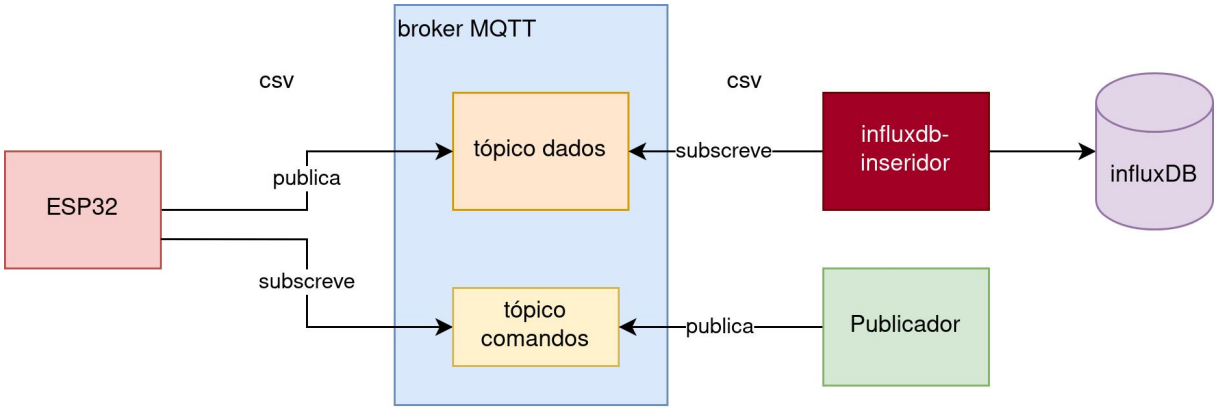
Para se fazer possível uma comunicação bidirecional com a ESP32, um protocolo bastante usado em IoT é MQTT. Isso pode ser útil para acionar, reiniciar, mudar parâmetros, e exercer outros tipo de controle remotamente.

Também é tido como bom costume em software restringir acesso a bancos de dados. Tendo o MQTT entre o dispositivo e o banco de dados, é obtida uma camada de isolamento da implementação. Caso seja desejado alterar o banco de dados utilizado para armazenar os parâmetros, não seria necessário alterar o código da ESP32.

Já que não há mais a necessidade de adesão ao formato da mensagem do InfluxDB, abre-se a opção de usar formatos mais compactos como CSV, ou até formatos binários estruturados como Apache Avro ou Google Protobuf. Isso pode se provar útil para diminuir o uso de rede da ESP.

Porém essa configuração exige dois serviços a mais. Um seria o "broker"MQTT, responsável por passar mensagens entre publicadores e subscritos. Outro serviço seria o inseridor de dados no InfluxDB, que receberia mensagens do broker MQTT, interpretaria as mensagens, e inseriria dados no InfluxDB. Um exemplo disso está na Figura 19. Nela também há a ilustração de um bloco chamado "aplicação", que pode ser um site, um aplicativo, qualquer aplicação que tenha acesso ao broker MQTT.

Figura 19 – Design para comunicação via MQTT. ESP32 publica no tópico ‘dados‘ e recebe comandos pelo tópico ‘comandos‘. Uma possível aplicação pode publicar no tópico ‘comandos‘. Um serviço influxdb-inseridor recebe mensagens do tópico ‘dados‘ e os insere no banco de dados InfluxDB



Fonte: do Autor

Para simplicidade, o formato de mensagem escolhido foi o CSV, logo para enviar os dados de variáveis "variavel1"igual a 1.23 e "variavel2"igual a 2.01, aferidos no timestamp 2334 da ESP para o influxDB, a ESP enviaria uma mensagem "1.23,2.01,2334". Caso mais pontos sejam enviados por vez, esses devem ser enviados separados por linhas.

Foram implementados os comandos de Resetar a ESP32, desligar a rotina de envio de dados e ligar a rotina de envio de dados.

7.1 CLIENTE MQTT ARDUINO

Para ser obtida uma conexão ao broker, a biblioteca "PubSubClient"do arduino foi utilizada, que possui funcionalidades tanto de publicar quanto de se subscrever a um tópico.

7.2 BROKER

O broker utilizado foi o eclipse mosquitto `https://mosquitto.org/`, por ser popular e ter disponível imagem docker para rodar localmente. A porta 1883 foi utilizada, padrão para o protocolo MQTT.

7.3 INFLUXDB-INSERIDOR

Utilizando a linguagem de programação Rust, pela minha familiaridade e afinidade, foi criado um serviço simples que se subscreve com o broker MQTT a tópicos em que a ESP irá publicar mensagens. Para a comunicação MQTT a biblioteca "paho-mqtt"foi utilizada, porém para a inserção no influxDB, a biblioteca existente está defasada e não é compatível com a última versão do banco de dados, então utilizei uma biblioteca para cliente HTTP genérico e implementei o formato de escrita, já que não foram utilizadas muitas funcionalidades do banco de dados nesse serviço.

O diagrama de blocos da montagem experimental do projeto nessa etapa é ilustrado na Figura 20.

7.4 DEMONSTRAÇÃO DOS COMANDOS

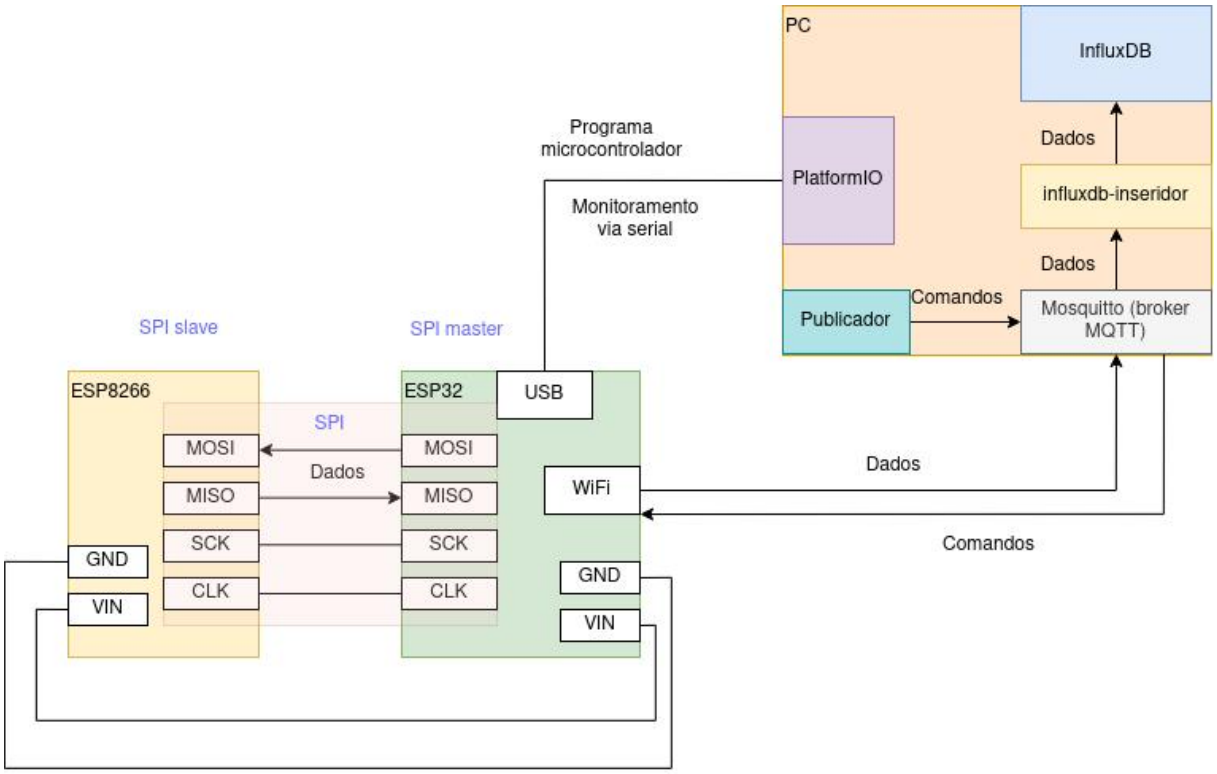
Lendo os logs da ESP32, do influxdb-inseridor, e tendo uma aplicação de linha de comando para enviar mensagens para o broker MQTT, marcado como Publicador, é possível ver o comportamento do sistema quando submetido a comandos.

Na Figura 21 é possível ver os logs da ESP32 printando registros a cada envio, e no influxdb-inseridor são observados registros a cada vez que o serviço lê dados do broker MQTT, enviados pela ESP32.

Ao ser enviado um comando para desligar as interrupções SPI e envio de dados, na Figura 22, através do Publicador, pode-se perceber como a ESP32 não mais envia dados, apenas imprime que está rodando. Também não são mais recebidos novos dados no influxdb-inseridor.

As interrupções e envio de dados são retomadas após o comando devido, como ilustrado em Figura 23. E, por fim, também há a possibilidade de resetar a ESP32

Figura 20 – Diagrama de blocos de montagem de desenvolvimento. ESP8266 e ESP32 se comunicam via SPI, ESP8266 alimentada transitivamente pela ESP32. ESP32 envia dados ao broker MQTT Mosquitto, que através do influxdb-inseridor chegam ao InfluxDB. Publicador publica mensagens no broker MQTT que são lidas pela ESP32. broker MQTT, InfluxDB e influxdb-inseridor hospedados no computador de desenvolvimento em rede local

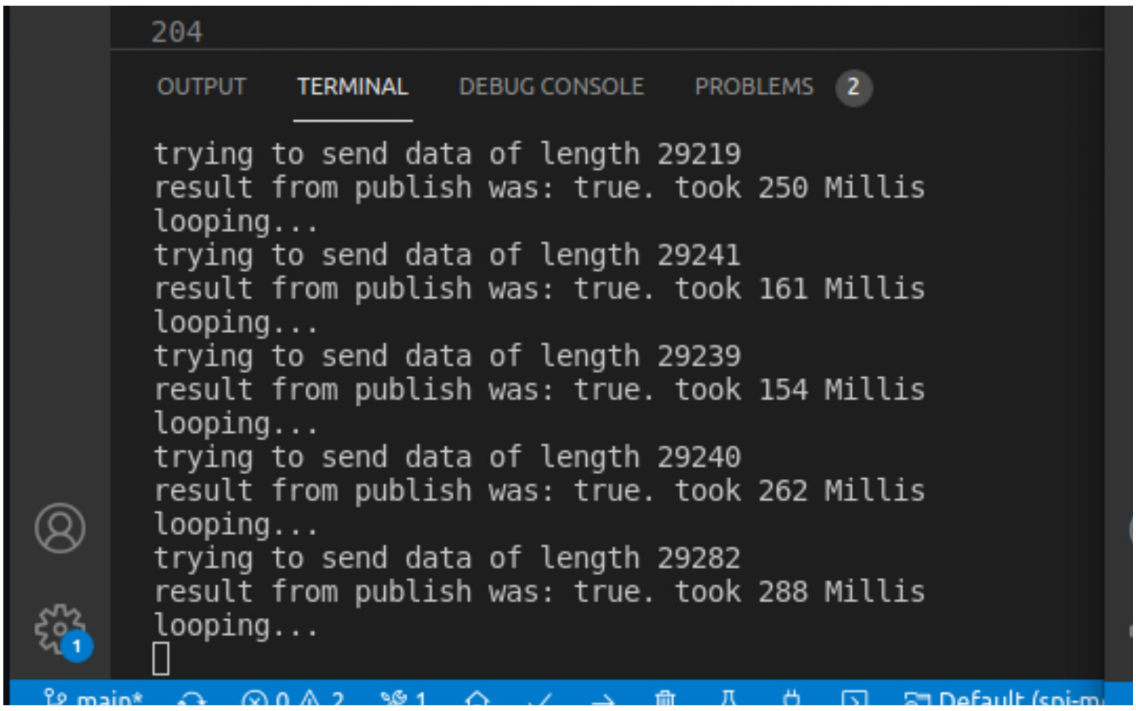


Fonte: do Autor

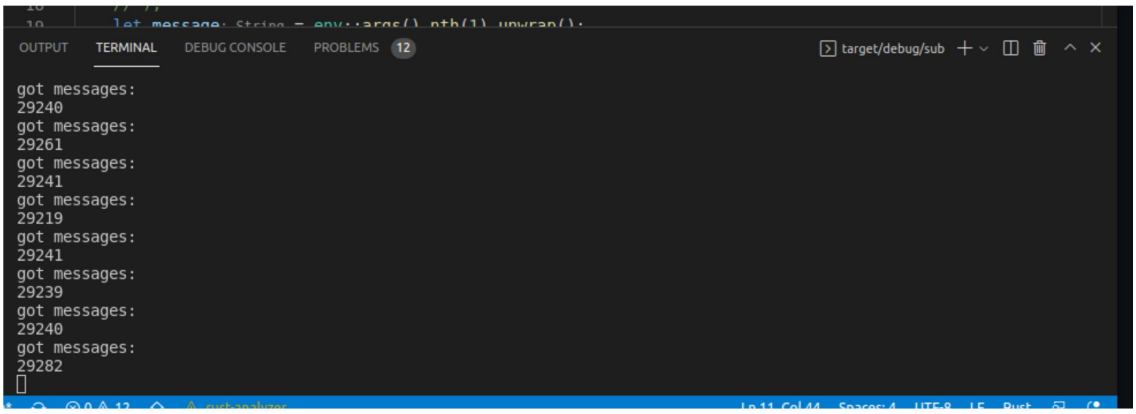
através de outro comando na Figura 24. Ao ter esse comando recebido pela ESP32, é possível ver que o recebimento de dados é momentaneamente pausado no influxdb-inseridor devido ao tempo de inicialização da ESP32.

Figura 21 – Envio de dados pela ESP32 ativado, influxdb-inseridor recebendo dados e inserindo no InfluxDB com sucesso

ESP32



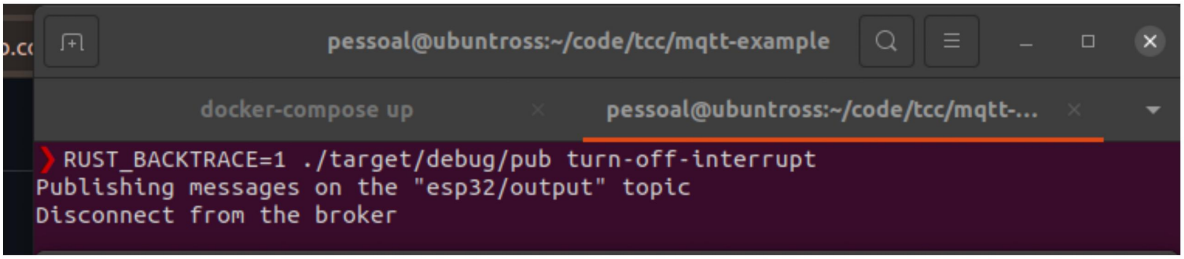
influxdb-inseridor



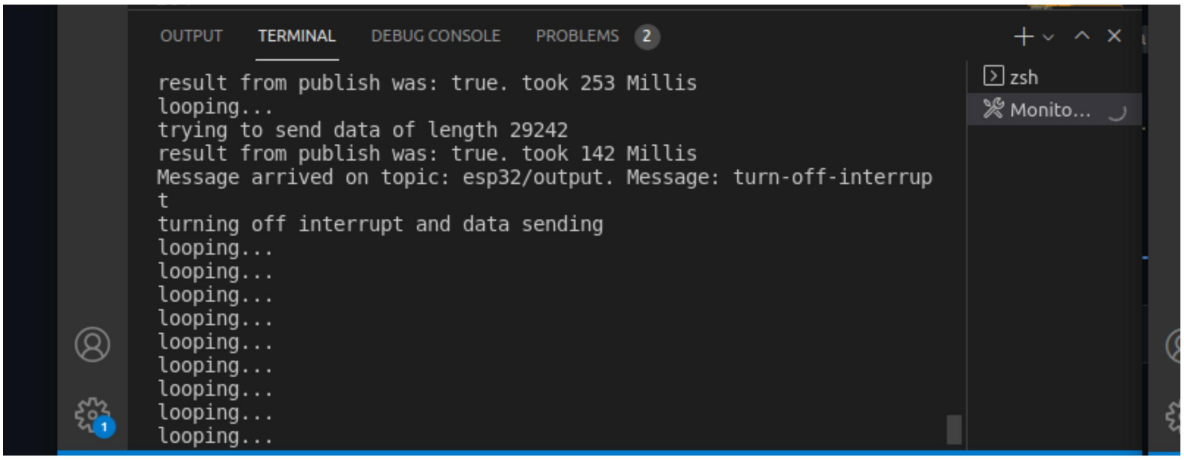
Fonte: do Autor

Figura 22 – Ao ser enviado um comando de desligar a interrupção utilizando o Publicador, é possível ver os envios da ESP32 não mais acontecerem

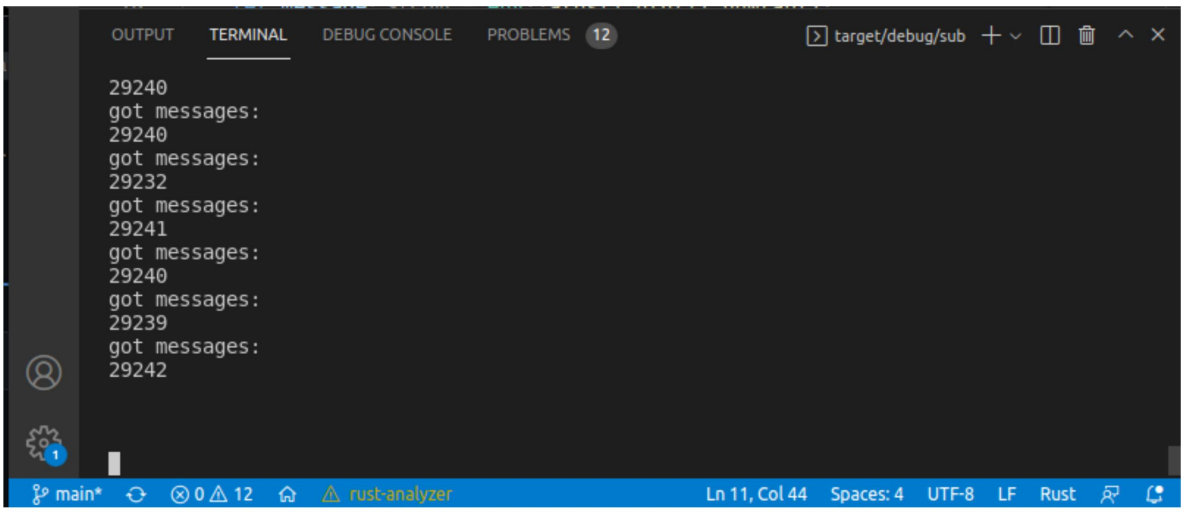
Publicador



ESP32



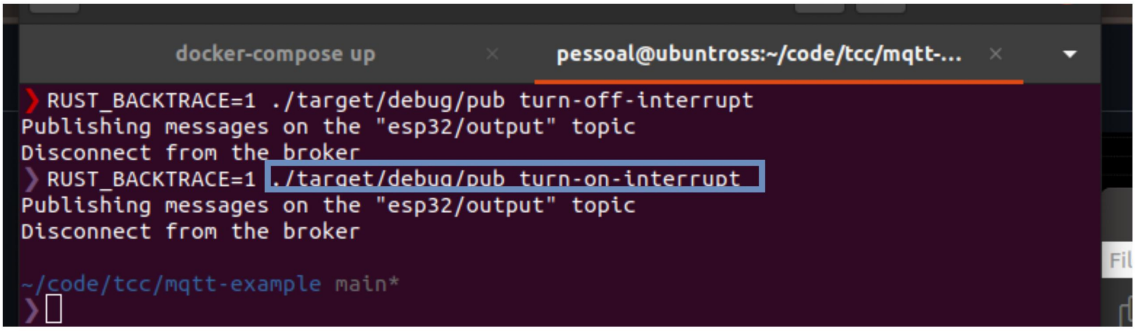
influxdb-inseridor



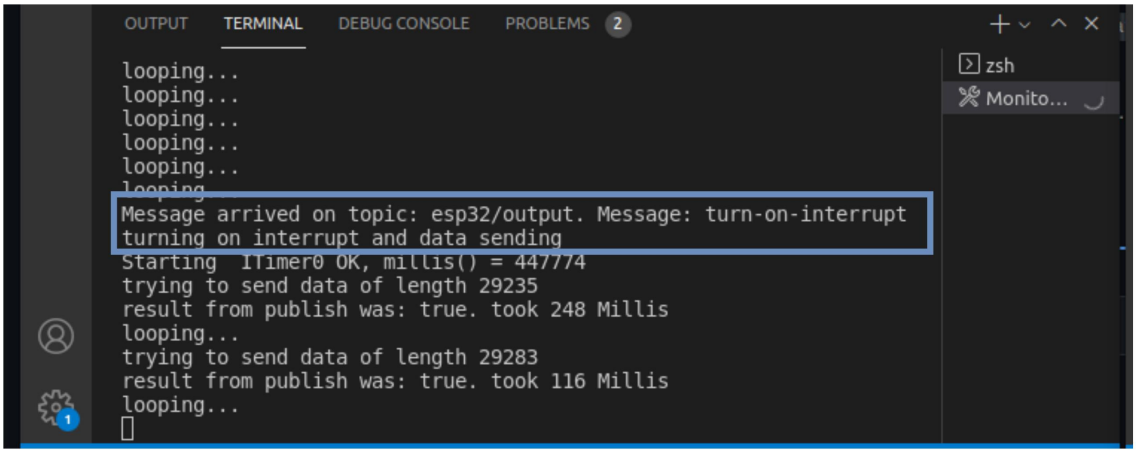
Fonte: do Autor

Figura 23 – Ao ser enviado um comando de religar as interrupções pelo Publicador, é possível ver o envio de dados pela ESP32 voltar, assim como o processamento de mensagens no influxdb-inseridor

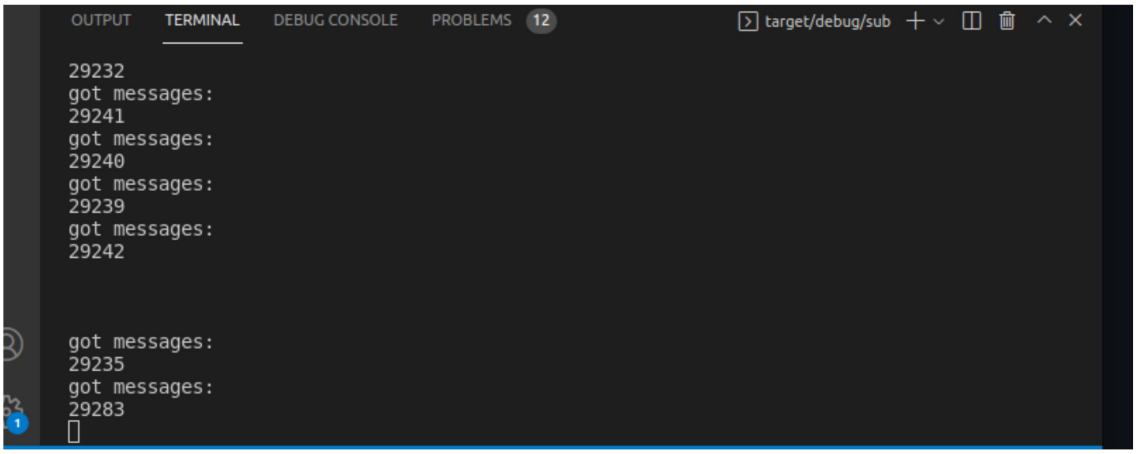
Publicador



ESP32



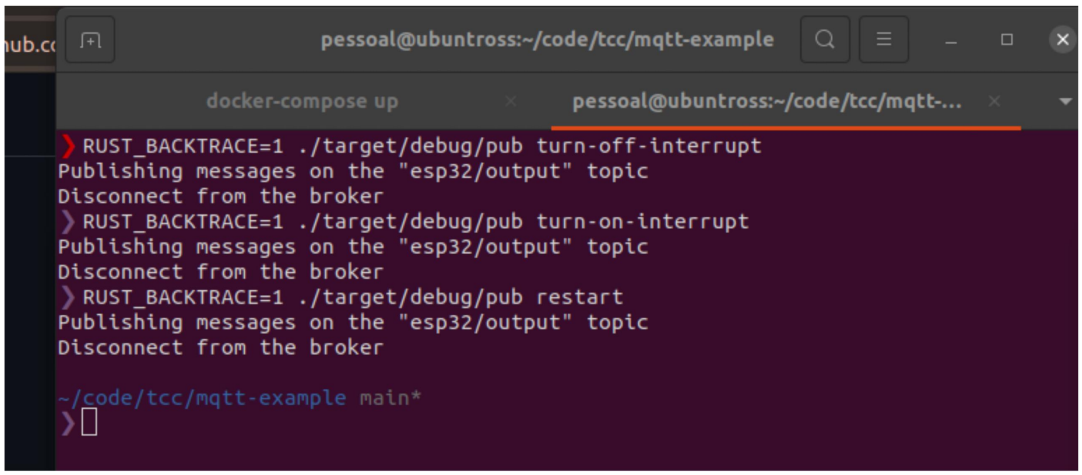
influxdb-inseridor



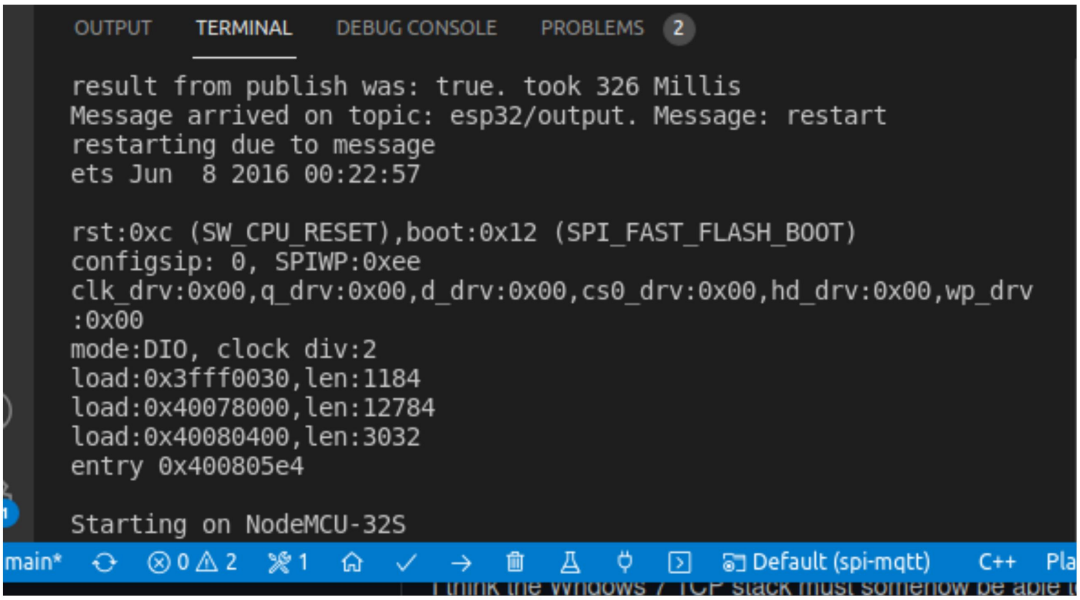
Fonte: do Autor

Figura 24 – Ao ser enviado um comando de resetar através do Publicador, é possível ver a ESP32 resetar e seus envios pausarem enquanto sua rotina de inicialização executa

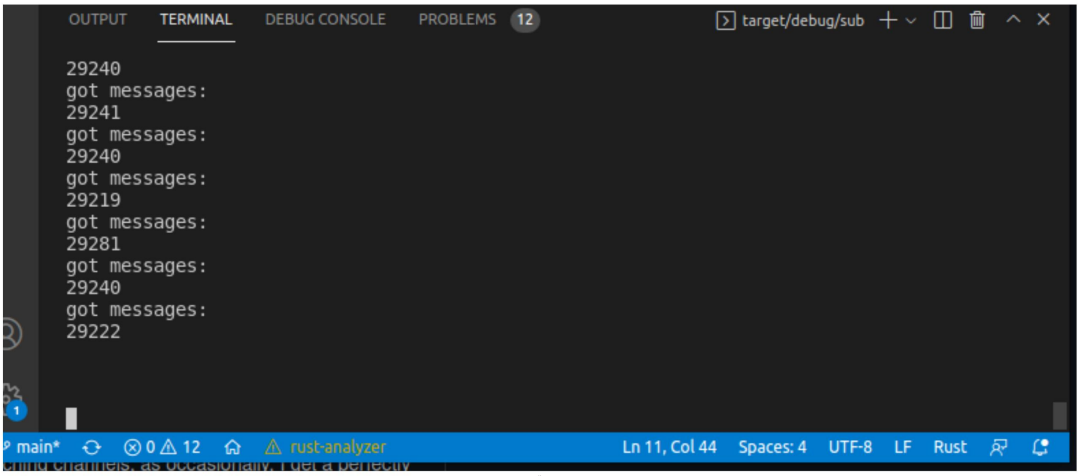
Publicador



ESP32



influxdb-inseridor



7.5 PERFORMANCE

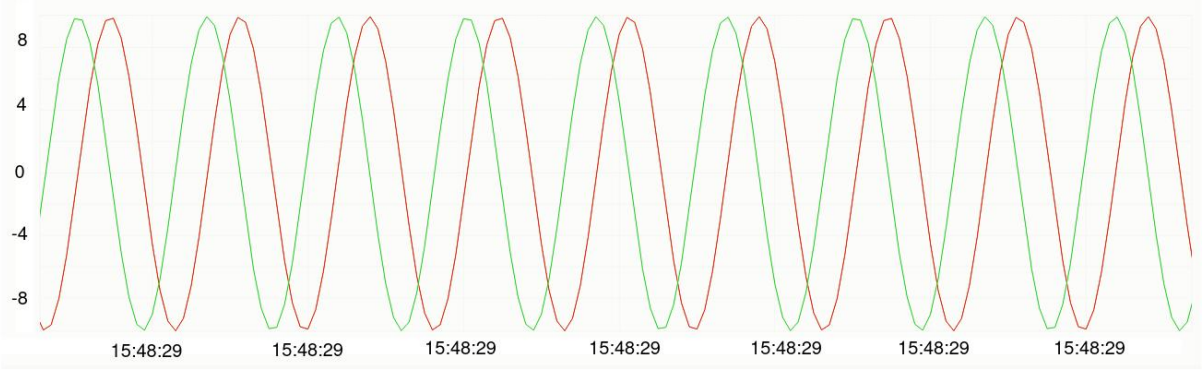
A performance de envio de dados para o broker MQTT foi um tanto decepcionante. Para a mesma carga de 1000 pontos, totalizando cerca de 16400 bytes, executada em sequência 101 vezes, obteve-se a seguinte distribuição de tempos de resposta: média 365 ms, mediana 263 ms, percentil95 817ms. Essa baixa performance talvez seja atribuída ao cliente MQTT do arduino, ao broker MQTT utilizado, a configuração desse broker, ou a algo intrínseco do protocolo não conseguir lidar bem com mensagens grandes.

Ao implementar a amostragem a um intervalo de 500 microssegundos, ou seja, até 2000 pontos por segundo, foi obtida maior lentidão no envio e problemas para utilizar a biblioteca PubSubClient do Arduino, já que o tamanho máximo do envio pela biblioteca estava bastante próximo. Então, como solução paliativa foi implementado o envio de 1000 pontos por envio a cada 500 milissegundos. Porém, ainda assim o intervalo entre envios permaneceu alto.

7.6 ENVIO DE DIVERSAS FORMAS DE ONDA

Abaixo senoides de diferentes frequências, componentes harmônicas e taxas de amostragem são observadas. Há um detalhamento grande numa senoide sem harmônicas de 60Hz, ilustrado na Figura 25.

Figura 25 – Seno e Cosseno a 60Hz, 1000 pontos por segundo

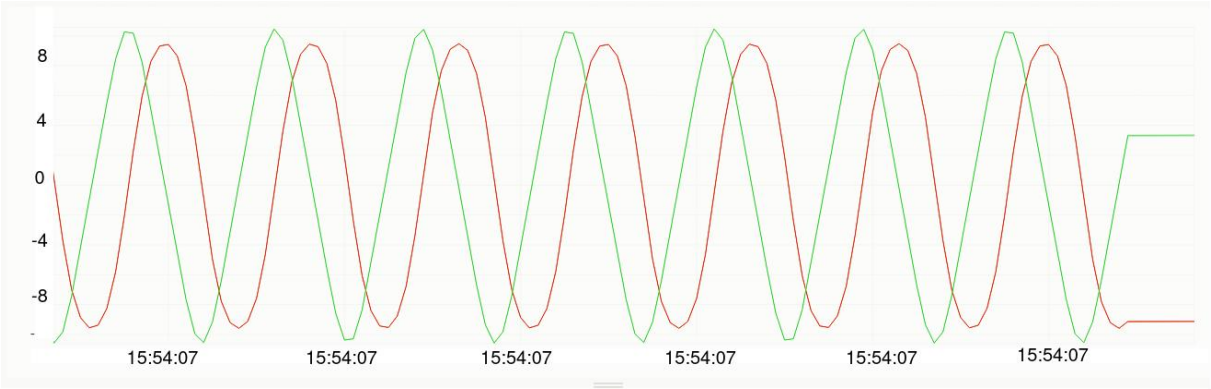


Fonte: do Autor

Adiciona-se 0.5 de amplitude na terceira harmônica de 60Hz na Figura 26, e ainda há pouca distorção.

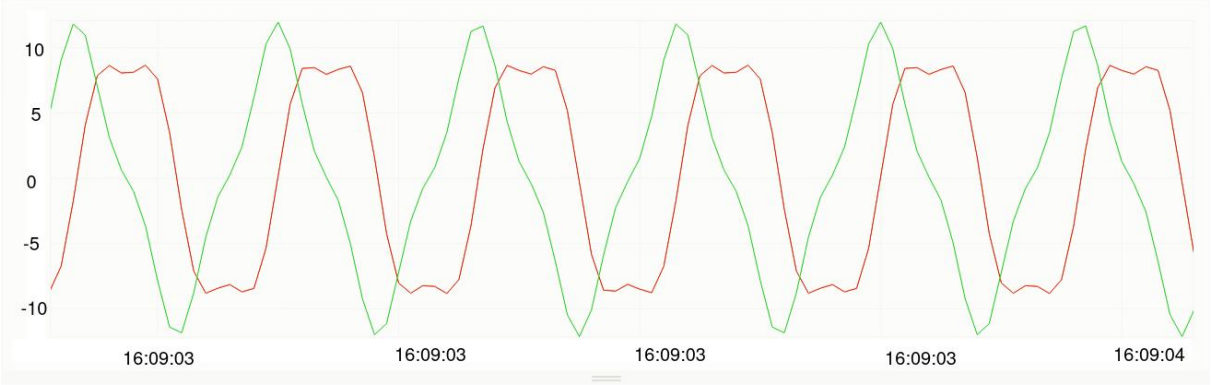
Na Figura 27 há uma 3ª harmônica mais proeminente, com amplitude 2, mas ainda com boa visualização. Na Figura 28 temos 6ª harmônica com amplitude 2 e é visível uma certa distorção.

Figura 26 – Seno e Cosseno a 60Hz, fundamental de amplitude 10, com 3ª harmônica de amplitude 0,5



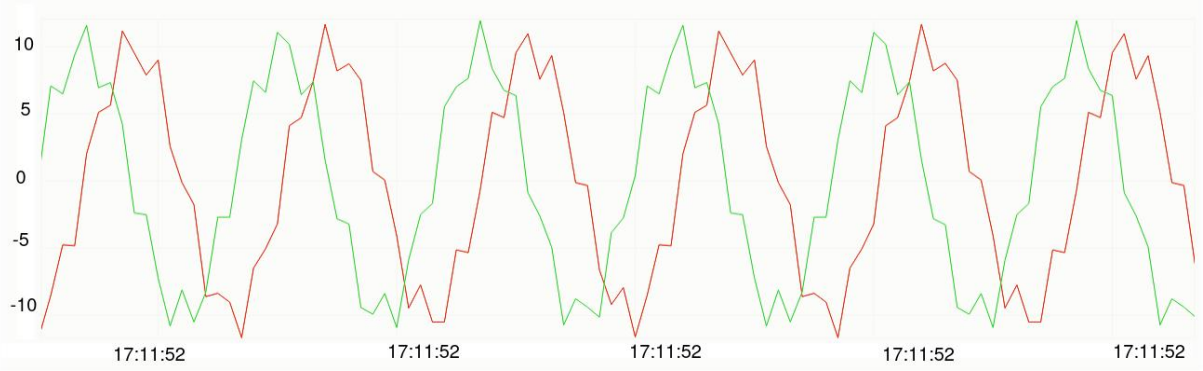
Fonte: do Autor

Figura 27 – Seno e Cosseno a 60Hz, fundamental de amplitude 10, com 3ª harmônica de amplitude 2



Fonte: do Autor

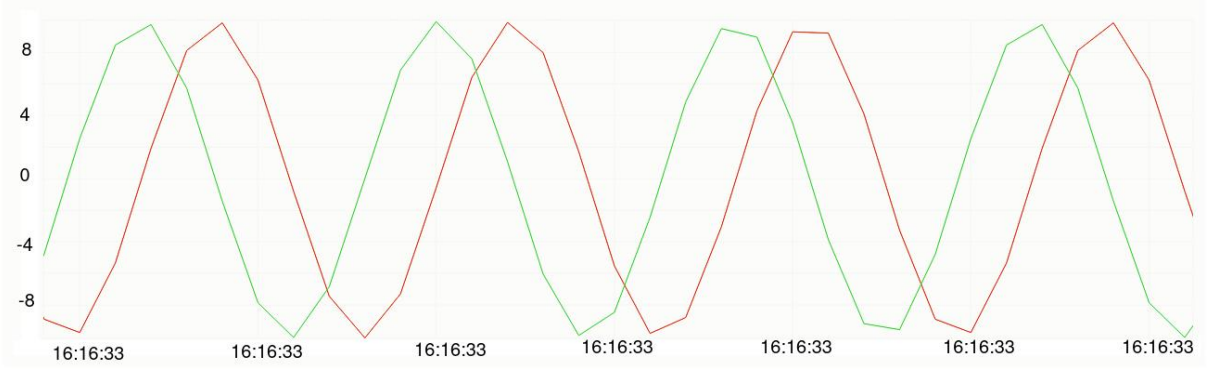
Figura 28 – Seno e Cosseno a 60Hz, fundamental de amplitude 10, com 6ª harmônica de amplitude 2



Fonte: do Autor

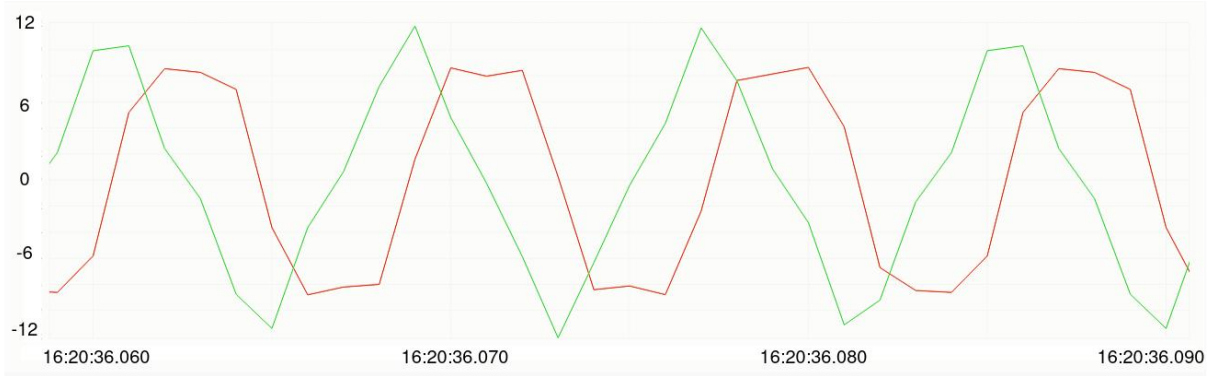
Observa-se que quanto maior a frequência da fundamental, mais distorcida fica a forma de onda. A Figura 29 mostra uma senoide de 120Hz, e com a adição de apenas uma 3ª harmônica, ilustrada na Figura 30, já é notável a distorção e irregularidade da forma de onda.

Figura 29 – Seno e Cosseno a 120Hz



Fonte: do Autor

Figura 30 – Seno e Cosseno a 120Hz de amplitude 10, com 3ª harmônica de amplitude 2



Fonte: do Autor

Amostrando a 2000 pontos por segundo, há um certo aumento de detalhamento, como comparando a senoide de 60Hz com 3ª harmônica de amplitude 2 amostrada a 1000 pontos por segundo na Figura 27 com a amostrada a 2000 pontos por segundo na Figura 31.

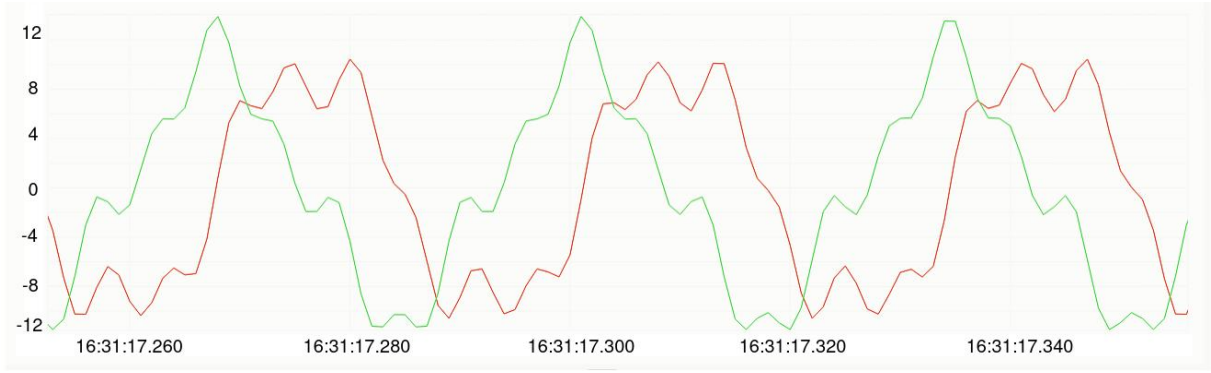
Espera-se que dobrando a taxa de amostragem e a frequência da onda amostrada, o grau de detalhamento seria o mesmo. Porém, vemos na Figura 32 uma senoide de 30Hz amostrada a 1000 pontos por segundo com mesma proporção entre harmônicos que a senoide de 60Hz da Figura 33, que foi amostrada a 2000 pontos. E é clara o

Figura 31 – Seno e Coseno a 60Hz, fundamental de amplitude 10, com 3ª harmônica de amplitude 2, 2000 pontos por segundo

Fonte: do Autor

detalhamento superior das formas de onda de 30Hz. Uma perda de dados durante a comunicação SPI é uma hipótese para a degradação.

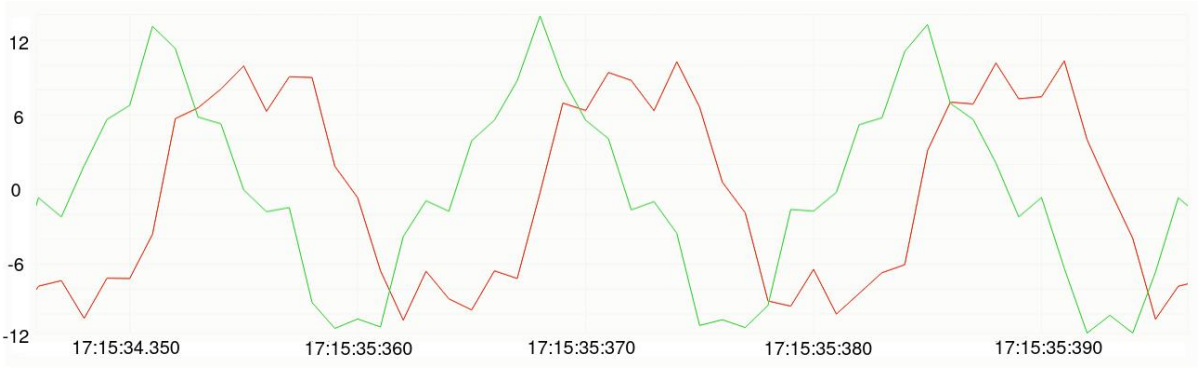
Figura 32 – Seno e Cosseno a 30Hz de amplitude 10, com 3ª harmônica de amplitude 2 e 6ª harmônica de amplitude 2



Fonte: do Autor

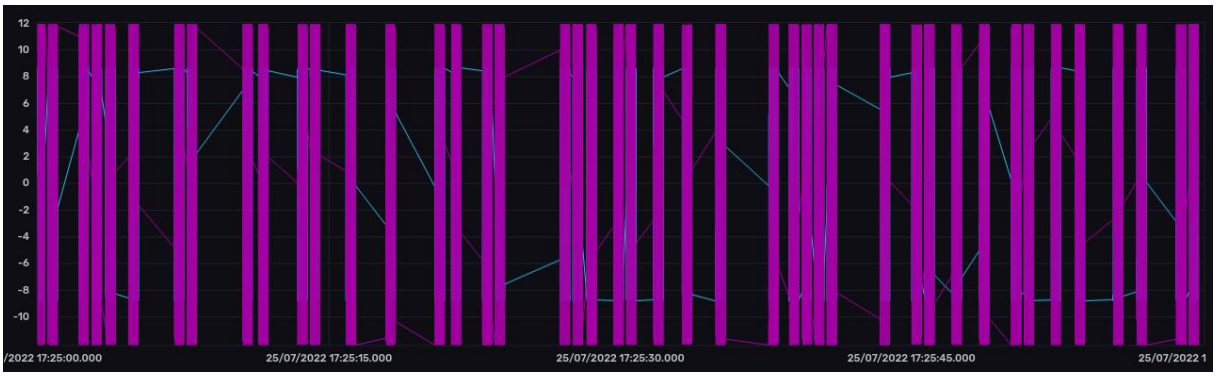
Nota-se na Figura 34 que nos envios de taxa de amostragem de 2000 pontos por segundo se tem um intervalo entre envios consideravelmente maior.

Figura 33 – Seno e Cosseno a 60Hz, fundamental de amplitude 10, com 3ª harmônica de amplitude 2, 6ª harmônica de amplitude 2, 2000 pontos por segundo



Fonte: do Autor

Figura 34 – Intervalos entre envios a uma amostragem de 2000 pontos por segundo. Observa-se que esses intervalos são bastante variáveis e tem ordem de mais de um segundo



Fonte: do Autor

8 ESTIMATIVA DE CUSTO PARA TER INFRAESTRUTURA NA NUVEM

Para se ter o InfluxDB, influxdb-inseridor e o broker MQTT, é possível colocá-los todos num mesmo servidor virtualizado de tamanho pequeno, com configurações simples. Logo, tendo a AWS como provedora de nuvem, seria utilizado um servidor do tipo t3a.micro, que possui 1 GB de memória e 0,4 a 2 unidades de processamento, custaria USD 0,0094 por hora segundo a documentação da AWS `https://aws.amazon.com/ec2/pricing/on-demand/`, ou menos de 7 dólares por mês. Seria necessário ter uma quantidade razoável de armazenamento, então utilizando o EBS (Elastic Block Store) `https://aws.amazon.com/ebs/pricing/` é cobrado USD 0,08 por gigabyte de disco por mês. 50 GB já seriam o suficiente, dado que o InfluxDB seja configurado para só manter dados de alguma recência (2 semanas de idade ao máximo, por exemplo), e os dados mais antigos serem enviados para um local mais barato, como AWS S3, onde é cobrado 5 centavos de dólar por Terabyte de dados por mês. Então o subtotal ficaria em menos de 11 dólares por mês.

Para configurar essas máquinas, a criação de um arquivo docker-compose pode ser realizada para se ter um ponto só de entrada para os 3 processos. Esse tipo de arquivo permite que a ferramenta Docker virtualize mais de uma aplicação através de um comando.

Um ponto importante a ser notado é que todos os provedores de computação na nuvem não cobram tráfego de rede de dados que entra na nuvem, mas cobram pelo dado extraído. No caso da AWS, é cobrado 9 centavos de dólar por Gigabyte baixado. Isso pode parecer pouco para essa aplicação, mas caso se tenha anos de histórico que se deseja baixar e analisar, a conta pode sair cara. Então idealmente para fazer uma descarga grande de dados, poderia fazer sentido realizar uma compressão desses na nuvem antes de serem resgatados, para economizar o tráfego de saída da nuvem.

9 CONCLUSÃO

9.1 SOBRE DESENVOLVIMENTO

9.1.1 Ferramentas

PlatformIO se aproveita do ambiente do editor de texto VS Code para modernizar o desenvolvimento pelo lazer (ou até profissional), deixando mais próximo ao que desenvolvedores web conhecem. Exemplo: poder customizar, ser de fácil instalação, ter todas as configurações em arquivos na mesma pasta do projeto, ter comandos de terminal para compilar, testar e fazer upload.

Levar mais tempo para escolher as ferramentas seria bom, em retrospecto. A IDE do Arduino foi escolhida a princípio pois era imaginado uma facilidade de desenvolvimento. Porém várias tarefas automatizadas em Integrated Development Environment (IDE)s modernas precisavam ser realizadas de forma manual na IDE do Arduino, como renomeação de variáveis, verificação de linha de código em que erro de compilação aconteceu, formatação do texto, entre outras. E essas tarefas manuais se provaram um fator negativo.

Além disso, devido à simplicidade da ferramenta, ela pode encorajar o desenvolvimento de más práticas de desenvolvimento de software. Como, por exemplo, querer testar código sem utilizar testes unitários, depender da printar na porta Serial para debugar o código, deixar o código inteiro num arquivo só, ter que realizar passos manuais para alterar configurações.

9.1.2 Código Fonte Utilizado

O código fonte utilizado nesse projeto está em `https://github.com/renecouto/tcc-codigo`

9.1.3 Testes

Teria sido útil também configurar testes de integração para cada parte. Exemplo, testar que pedir dados para a slave funcionava. Testar que interrupções são chamadas. Testar ler do banco de dados após a escrita. Porém fazer isso necessitaria mais refatoração. E quando o código é refatorado, as coisas costumam parar de funcionar. Isso revela mais uma vez como é frutífero modelar o código como um primeiro passo. Como ter poucas funções que acessam variáveis globais é útil. Sim, há casos em que a forma mais eficiente de se transmitir informação é com variáveis globais voláteis, mas são poucos. Nominalmente, o caso da interrupção de hardware, explicado em `https://github.com/khoih-prog/ESP32TimerInterrupt#important-notes-about-isr`.

9.1.4 Segurança

Atualmente o código versionado no GitHub possui as credenciais tanto para a conexão WiFi quanto para o acesso ao banco de dados. É um bom costume em desenvolvimento de software nunca armazenar credenciais junto ao código. Uma opção simples para isso isolar todas as credenciais em um arquivo 'credentials.h' e remover esse arquivo do versionamento.

As conexões com MQTT e influxDB também não estão utilizando criptografia, nem autenticação. Isso seria de suma importância caso haja interesse em configurar esses serviços na nuvem, ou de serem expostos de qualquer forma pública, fora da rede local. Também seria importante criar usuários diferentes para o banco de dados, um com apenas acesso à leitura, para visualizar gráficos, e outro com acesso de escrita, para os sistemas que inserem dados.

9.2 SOBRE A SOLUÇÃO EM SI

Foi possível enviar parâmetros a uma taxa consideravelmente alta. Há algumas distorções na forma de onda devido ao grau de precisão exigido para uma grandeza tão rápida quanto uma tensão alternada. O protocolo SPI utilizado possui certa flexibilidade, e a comunicação via MQTT abre várias possibilidades. A plotagem utilizando a interface do InfluxDB é satisfatória, porém pode ficar bastante lenta quando intervalos da ordem de horas são selecionados, dado a carga de dados. A conexão à rede local utilizando WiFi é robusta e tem boas velocidades, não foi identificada como limitante nesse projeto.

O objetivo de enviar dados para a nuvem não foi concluído, nem a programação do controlador do conversor de potência em si. Em contrapartida, a comunicação via MQTT, que não era objetivo, foi implementada.

Esse projeto foi marcado por idas e vindas, muitos aprendizados relacionados à área de sistemas embarcados, bancos de dados e IoT.

9.3 TRABALHOS FUTUROS

9.3.1 Implementar Função de calcular FFT no InfluxDB

Os dados foram salvos no banco de dados InfluxDB. É possível então fazer painéis para monitoramento dessas métricas. Uma função que poderia ser incorporada ao banco de dados InfluxDB que melhoraria as analises desses dados seria a computação da FFT (fast fourier transform, ou transformação de fourier rápida).

A adição dessa funcionalidade foi requisitada, mas ainda não atendida `https://github.com/influxdata/influxdb/issues/15122`. Como comentado na requisição, é bastante custoso extrair dados do banco de dados, computar a transformada e

reinseri-los, além do fato de isso necessitar de outra tabela. Se a funcionalidade de FFT for implementada no banco de dados em si, seria possível também exportar esses resultados para outras ferramentas de visualização como Grafana.

9.3.1.1 Visualização dos dados de forma análoga a um osciloscópio

Num osciloscópio é comum a visualização de formas de onda num formato estático, que facilita a análise e verificação de mudanças na forma de onda ao longo do tempo. No InfluxDB é possível visualizar apenas fatias de tempo selecionadas. Se for desejado uma atualização os dados, é necessário selecionar outra fatia de tempo, ou configurar para atualizar a cada N segundos. Porém, é necessário fazer o cálculo da duração da fatia de tempo, e manualmente escolher o tamanho da janela. No caso de a frequência mudar, os dados sairiam de sincronia aos poucos. Para esse caso de uso, seria interessante uma detecção automática do período da onda observada, para que ciclos consecutivos dela sejam mostrados um acima do outro, assim como no osciloscópio, sem ter os dados deslizando.

REFERÊNCIAS

"MQTT Version 3.1.1 Plus Errata 01". [*S.l.*], dez. 2015. Disponível em: http://docs.oasis-%20open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html.

ABARESHI, Behzad; WILLIAMS, Doug; MARSHALL, Robert. Building a telescope engineering data system with Redis, InfluxDB and Grafana. *In*: GUZMAN, Juan C.; IBSEN, Jorge (Ed.). **Software and Cyberinfrastructure for Astronomy V**. [*S.l.*]: SPIE, 2018. v. 10707. International Society for Optics e Photonics, p. 452–458.

APACHE. **Apache Flink**. [*S.l.*: *s.n.*], 2014. https://flink.apache.org. Acessado em: 27 de julho de 2022.

APACHE. **Apache hudi**. [*S.l.*: *s.n.*], 2021. https://hudi.apache.org. Acessado em: 27 de julho de 2022.

APACHE. **Apache Spark**. [*S.l.*: *s.n.*], 2018. https://spark.apache.org. Acessado em: 27 de julho de 2022.

ARBUGERI, José Augusto. **RETIFICADOR PFC OPERANDO EM ALTA FREQUÊNCIA EMPREGANDO SEMICONDUTORES GAN: ASPECTOS DE PROJETO, LAYOUT, MODULAÇÃO E CONTROLE**. Florianópolis: [*s.n.*], jul. 2019.

BARHATE, Akshay; MUNDADA, Kirti; KULKARNI, Bhargavi; DEORE, Manasi. Smart Internet of Things Based Automatic Power Factor Control. *In*: PROCEEDINGS of International Conference on Communication and Information Processing (ICCIP). [*S.l.*: *s.n.*], 2019.

BARRY, Peter; CROWLEY, Patrick. Chapter 4 - Embedded Platform Architecture. *In*: BARRY, Peter; CROWLEY, Patrick (Ed.). **Modern Embedded Computing**. Boston: Morgan Kaufmann, 2012. p. 41–97. ISBN 978-0-12-391490-3.

BARYBIN, Oleksii; ZAITSEVA, Elina; BRAZHNYI, Volodymyr. Testing the Security ESP32 Internet of Things Devices. *In*: 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC SnT). [*S.l.*: *s.n.*], 2019. p. 143–146.

BHAGAVATHY, P; LATHA, R; THAMIZHMARAN, E. Development of IoT Enabled Smart APFC Panel for Industrial Loads. *In*: 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT). [*S.l.*: *s.n.*], 2019. p. 1–5.

FERNÁNDEZ-CARAMÉS, Tiago M. An intelligent power outlet system for the smart home of the Internet of Things. **International Journal of Distributed Sensor**

Networks, SAGE Publications Sage UK: London, England, v. 11, n. 11, p. 214805, 2015.

GOOGLE. **Drive API**. [*S.l.*: *s.n.*]. `https://developers.google.com/drive/api`. Acessado em: 03 de julho de 2022.

HAMMING, Richard W. Error detecting and error correcting codes. **The Bell system technical journal**, Nokia Bell Labs, v. 29, n. 2, p. 147–160, 1950.

KODALI, Ravi Kishore; SORATKAL, SreeRamya. MQTT based home automation system using ESP8266. *In*: 2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC). [*S.l.*: *s.n.*], 2016. p. 1–5.

MARTINVIITA, Mikael. Time series database in Industrial IoT and its testing tool. **Master's Thesis, Degree Programme in Computer Science and Engineering**, p. 1–62, 2018.

MUTEBA, K.F.; DJOUANI, K; OLWAL, T. 5G NB-IoT: Design, Considerations, Solutions and Challenges. **Procedia Computer Science**, v. 198, p. 86–93, 2022. 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare. ISSN 1877-0509.

NAYLOR, David; FINAMORE, Alessandro; LEONTIADIS, Ilias; GRUNENBERGER, Yan; MELLIA, Marco; MUNAFÒ, Maurizio; PAPAGIANNAKI, Konstantina; STEENKISTE, Peter. The Cost of the "S"in HTTPS. *In*: PROCEEDINGS of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. Sydney, Australia: Association for Computing Machinery, 2014. (CoNEXT '14), p. 133–140.

OUGHTON, Edward J.; FRIAS, Zoraida; VAN DER GAAST, Sietse; VAN DER BERG, Rudolf. Assessing the capacity, coverage and cost of 5G infrastructure strategies: Analysis of the Netherlands. **Telematics and Informatics**, v. 37, p. 50–69, 2019. ISSN 0736-5853.

RAD, Babak Bashari; BHATTI, Harrison John; AHMADI, Mohammad. An introduction to docker and analysis of its performance. **International Journal of Computer Science and Network Security (IJCSNS)**, International Journal of Computer Science e Network Security, v. 17, n. 3, p. 228, 2017.

SINGH, Meena; RAJAN, M.A.; SHIVRAJ, V.L.; BALAMURALIDHAR, P. Secure MQTT for Internet of Things (IoT). *In*: 2015 Fifth International Conference on Communication Systems and Network Technologies. [*S.l.*: *s.n.*], 2015. p. 746–751.

VACCARI, Ivan; AIELLO, Maurizio; CAMBIASO, Enrico. SlowITe, a Novel Denial of Service Attack Affecting MQTT. **Sensors**, v. 20, n. 10, 2020. ISSN 1424-8220.

VALIALKIN, Aliaksandr. **When size matters — benchmarking Victoria: Metrics vs Timescale and InfluxDB**. [*S.l.*: *s.n.*], 2018. `https://valyala.medium.com/when-size-matters-benchmarking-victoriametrics-vs-timescale-and-influxdb-6035811952d4`. Acessado em: 25 de julho de 2022.

VARGHESE, Susan G; KURIAN, Ciji Pearl; GEORGE, VI; JOHN, Anupriya; NAYAK, Varsha; UPADHYAY, Anil. Comparative study of zigBee topologies for IoT-based lighting automation. **IET Wireless Sensor Systems**, Wiley Online Library, v. 9, n. 4, p. 201–207, 2019.

VOHRA, Deepak. Apache Avro. *In*: PRACTICAL Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools. Berkeley, CA: Apress, 2016. p. 303–323. ISBN 978-1-4842-2199-0.

GLOSSÁRIO

WiFi Rede de internet sem fio

5G Rede de internet sem fio a satélite

código aberto Programas de software de código aberto tem seu código-fonte disponível publicamente para visualização e edição

imagens Docker Arquivos que definem o sistema operacional ou aplicação que a ferramenta Docker irá inicializar

framework Conjunto de bibliotecas e/ou programas de software altamente integradas

real time clock Componente físico de microcontroladores que permite armazenar de maneira eficiente a informação da data e horário atual, sendo atualizada em tempo real, com possibilidade de sincronização com servidores

timestamp Estampa de tempo. Informação de data e horário, usualmente utilizando o fuso horário GMT

MQTT Protocolo de comunicação em rede