Universidade Federal do Rio Grande do Norte

Centro de Tecnologia - CT

Curso de Engenharia Mecatrônica

SmartContact: Dispositivo IoT para Monitoramento em Tempo Real de Contator Eletromecânico

Albertho Síziney Costa

Natal

2023

Albertho Síziney Costa

SmartContact: Dispositivo IoT para Monitoramento em Tempo Real de Contator Eletromecânico

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do título de Engenheiro Mecatrônico, orientado pelo Prof. Heitor Medeiros Florencio

Universidade Federal do Rio Grande do Norte (UFRN)

Centro de Tecnologia (CT)

Curso de Engenharia Mecatrônica

Orientador: Heitor Medeiros Florencio

Natal

2023

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Costa, Albertho Síziney.
SmartContact: dispositivo IoT para monitoramento em tempo real de contator eletromecânico / Albertho Siziney Costa. - 2023.
63 f.: il.

Trabalho de Conclusão de Curso - TCC (graduação) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Curso de Engenharia Mecatrônica. Natal, RN, 2023.
Orientação: Prof. Dr. Heitor Medeiros Florencio.

1. Internet das Coisas (IoT) - TCC. 2. Indústria 4.0 - TCC. 3. Contator Eletromecânico - TCC. 4. ESP32-C3 - TCC. 5. Comunicação MQTT - TCC. I. Florencio, Heitor Medeiros. II. Título.

RN/UF/BCZM CDU 004.738.5:655

Elaborado por Fernanda de Medeiros Ferreira Aquino - CRB-15/301

Albertho Síziney Costa

SmartContact: Dispositivo IoT para Monitoramento em Tempo Real de Contator Eletromecânico

Trabalho aprovado. Natal, 15 de dezembro de 2023:

Heitor Medeiros Florencio
Orientador

Samaherni Morais Dias
Avaliador interno

Luiz Eduardo Cunha Leite
Avaliador interno

Jefferson Doolan Fernandes
Avaliador externo

Natal
2023

Agradecimentos

À minha mãe Elizabeth, por me ensinar que todos os sonhos podem ser conquistados, mesmo diante dos mais adversos desafios.

À minha avó, dona Yvonete, por me ensinar que o amor e a empatia deve prevalecer sempre em nossos corações.

À minha namorada Clarice, pela paciência, compreensão e encorajamento nos momentos de estudo intermináveis.

Aos meus irmãos de outra mãe Belinha, Fred e Ted, pelo companheirismo incondicional.

A todos os meus familiares.

Ao meu professor orientador Heitor Medeiros Florêncio, pela constante disponibilidade apesar de toda correria, e pela oportunidade proporcionada de explorar e compreender mais profundamente o setor industrial.

Aos amigos que o curso me proporcionou, em especial: Igor; Jordiel; Keryson; Mateus; Danilo; Emanuel; Rafael; Enrico; Ícaro; Jonathan; Fernanda; Bruna; Carlos; Vinicius; Guilherme; Luiz; Samuel; Felipe; Lucas; Kleiton, Jefferson e Vitor.

Resumo

Em um contexto atual de Indústria 4.0 e Internet das Coisas - IoT (Internet of Things), torna-se essencial a modernização de alguns sistemas de controle e automação em indústrias com o uso de novas tecnologias. A aplicação de soluções de IoT nos sistemas industriais possibilita a conexão de diferentes máquinas e componentes de uma planta industrial através da Internet para fornecer informações e, consequentemente, apoiar na tomada de decisões. O contator eletromecânico é amplamente utilizado em quadros elétricos industriais devido a sua capacidade de acionar cargas de alta potência, como motores elétricos. Entretanto, os contatores disponíveis no mercado não são projetados para se comunicar com as soluções de IoT, o que dificulta o monitoramento das plantas industriais por meio dos contatores. Este trabalho apresenta o projeto de um dispositivo IoT capaz de realizar o monitoramento em tempo real de um contator eletromecânico, por meio de um sensor magnético, e de disponibilizar os dados para uma plataforma IoT. O projeto do dispositivo foi baseado no SoC (System on Chip) ESP32-C3 da ESPRESSIF. Os dados são enviados para a plataforma Adafruit IO através do protocolo de comunicação MQTT (Message Queuing Telemetry Transport). Os resultados apresentam os dados recebidos na plataforma do dispositivo IoT acoplado a um contator em uma bancada de teste.

Palavras-chave: IoT; Indústria 4.0; Contator Eletromecânico; ESP32-C3; MQTT.

Abstract

In the current context of Industry 4.0 and the Internet of Things (IoT), the modernization of control and automation systems in industries becomes essential through the insertion of new technologies. The application of IoT solutions in industrial systems enables the connection of different machines and components within a plant via the Internet to provide information and, consequently, support decision-making. Electromechanical contactors are widely used in industrial electrical panels due to their ability to control high-power loads, such as electric motors. However, the contactors available in the market are not designed to communicate with IoT solutions, making it challenging to monitor industrial plants through contactors. This work presents the design of an IoT device capable of real-time monitoring of an electromechanical contactor using a magnetic sensor and making the data available to an IoT platform. The device's design was based on the ESP32-C3 System on Chip (SoC) from ESPRESSIF. The data is transmitted to the Adafruit IO platform using the MQTT (Message Queuing Telemetry Transport) communication protocol. The results showcase the data received on the IoT device platform attached to a contactor in a test bench.

Keywords: IoT; Industry 4.0; Electromechanical Contactor; ESP32-C3; MQTT.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFRN	Universidade Federal do Rio Grande do Norte
VSCode	Visual Studio Code
MQTT	*Message Queuing Telemetry Transport*
IoT	*Internet of Things*
IIoT	*Industrial Internet of Things*
Wi-Fi	*Wireless Fidelity*
JSON	*JavaScript Object Notation*
HTTP	*Hypertext Transfer Protocol*
HTML	*Markup Language*
LED	*Light Emitting Diode*
NVS	*Non-Volatile Storage*
RF	*Radio Frequency*
GPIO	*General Purpose Input/Output*
RTOS	*Real-Time Operating System*
TWDT	*Task WatchDog Timer*
WDT	*WatchDog Timer*
WEB	*World Wide Web*
AP	*Access Point*
STA	*Station*
IP	*Internet Protocol*
NTP	*Network Time Protocol*
QoS	*Quality of Service*
RTC	*Real Time Clock*

DSM	*Deep Sleep Mode*
PCB	*Printed Circuit Board*
SoC	*System on Chip*
LE	*Low Energy*
PWM	*Pulse Width Modulation*
ADC	*Analogue-to-Digital Converter*
UART	*Universal Asynchronous Receiver/Transmitter*
SPI	*Serial Peripheral Interface*
SSID	*Service Set Identifier*
I2C	*Inter-Integrated Circuit*
SRAM	*Static Random-Access Memory*
ROM	*Read-Only Memory*
ESP-IDF	*Espressif IoT Development Framework*
GND	*Ground*
USB	*Universal Serial Bus*
SAR	*Successive Approximation Register*
CPU	*Central Processing Unit*
PMU	*Power Management Unit*
CLP	*Controlador Lógico Programável*
AWS	*Amazon Web Services*

Sumário

1 Introdução

Nos últimos anos, o avanço tecnológico tem transformado significativamente a maneira como interagimos com o mundo ao nosso redor, impulsionando o uso de dispositivos inteligentes e conectados. No meio industrial, o conjunto de tecnologias, tais como Inteligência Artificial (IA), Internet das Coisas (IoT), Manufatura Aditiva, entre outras, desencadeou uma revolução tecnológica denominada de quarta revolução industrial, ou Indústria 4.0 (SCHWAB, 2017).

Os sistemas embarcados, que já estavam em constante evolução tecnológica, somados às novas tecnologias da Tecnologia da Informação (TI), tornam-se elementos essenciais na integração do mundo físico das indústrias com o mundo virtual através da Internet.

O termo Internet das Coisas Industrial (IIoT) é utilizado para aplicações de soluções de IoT no meio industrial. Os projetos de IoT na indústria envolvem a interconexão de máquinas, controladores e componentes industriais por meio da Internet a fim de disponibilizar dados às aplicações e serviços, possibilitando a análise de dados em tempo real.

A instalação de dispositivos IoT em plantas industriais impacta diretamente na tomada de decisões e, consequentemente, na eficiência produtiva. Nos dias atuais, já é possível encontrar sensores capazes de coletar os dados e transmiti-los para aplicações em nuvem (LIMA; GOMES, 2021).

1.1 Motivação

Os sistemas de controle e automação nas plantas industriais contêm sensores, atuadores, controladores, interfaces homem-máquina e dispositivos de acionamento de máquinas.

Dentro desse contexto, já existem controladores industriais com tecnologias de comunicação para soluções de IoT, como os controladores XP340 da empresa Altus Sistemas de Automação, A8-G da empresa Alfacomp Automação Industrial e Delta AS200 da empresa Kalatec Automação. Além disso, existem dispositivos industriais que realizam a interconexão de controladores com soluções de IoT, denominados de gateway IoT, como o IOT2040 da empresa Siemens (BABAYIGIT; ABUBAKER, 2023). Entretanto, a modernização de máquinas e plantas industriais pode gerar um custo alto quando considera-se o tempo de máquina parada e o custo de instalação dos novos equipamentos.

Existem alguns quadros de comando de máquinas que não têm controladores industriais e usam contatores eletromecânicos para o controle e acionamento das máquinas.

Por outro lado, existem sistemas de automação que, mesmo contendo contatores para o acionamento das máquinas, o controle é realizado por controladores industriais. Assim, o contator eletromecânico é um componente fundamental e amplamente utilizado em projetos de automação desde os mais simples até os mais complexos.

O contator eletromecânico é um elemento que pode determinar se um motor conectado a esse, será acionado ou não. Por isso, tornar os contatores objetos IoT dentro de uma planta industrial permite fornecer informações em tempo real das máquinas aos supervisores e gerentes da fábrica.

Um contator pode ser conectado à Internet através de um dispositivo IoT capaz de acoplar-se frontalmente ao contator, semelhante a um bloco auxiliar, para coletar o estado do mesmo. Como qualquer outro dispositivo IoT, esse dispositivo deve ser capaz de conectar-se à internet através de protocolos de comunicação para IoT, como o MQTT, e publicar os dados em alguma plataforma ou aplicação.

Assim, a principal motivação deste trabalho foi buscar uma solução para transformar os contatores eletromecânicos em dispositivos IoT, permitindo que as máquinas se conectem e gerem dados em tempo real da fábrica. Essa abordagem visa não apenas modernizar as plantas industriais, mas também contribuir para o campo emergente de sistemas embarcados e IoT, demonstrando como essas tecnologias podem ser integradas de maneira eficaz para aplicações industriais.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um dispositivo IoT, denominado de *SmartContact*, capaz de acopla-se a um contator para capturar o seu estado (ligado ou desligado) e disponibilizar esses dados para alguma aplicação através da Internet.

O dispositivo *SmartContact* deve enviar o estado do contator e o horário da captura desse estado. Após essa coleta, o dispositivo deve conectar-se à Internet e enviar, através do protocolo de comunicação MQTT, os dados para um *broker* MQTT de alguma plataforma ou aplicação. Além disso, é necessário ter uma funcionalidade de configuração inicial que permita o usuário informar os dados da rede sem fio do ambiente e os dados do *broker* MQTT.

Este trabalho abrange diversas áreas, desde o projeto e concepção da placa eletrônica do dispositivo até a fabricação mecânica do case acoplável ao contator. A aplicação de um SoC é essencial para a integração do dispositivo ao ecossistema IoT, possibilitando o desenvolvimento do *firmware* capaz de realizar as funções de processamento, armazenamento

e conexão com a Internet.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Definir o modelo de encaixe do contator a ser monitorado pelo dispositivo IoT;
- Definir o sensor responsável por capturar o estado do contator;
- Definir o microcontrolador ou SoC a ser utilizado no projeto a partir dos requisitos;
- Projetar uma placa de circuito PCB do dispositivo;
- Projetar um *case* capaz de armazenar os componentes e acoplar-se ao contator;
- Desenvolver o *firmware* do SoC para implementar as funcionalidades do dispositivo;
- Definir o protocolo de comunicação IoT a ser utilizado;
- Realizar testes de publicação dos dados do contator (estado e horário da coleta) em alguma plataforma IoT.

O projeto do dispositivo *SmartContact* foi validado com testes realizados com um contator do fabricante *Schneider Electric* e os dados dos testes foram enviados para a plataforma Adafruit IO.

1.3 Estrutura do Trabalho

O Capítulo 2 descreve os conceitos teóricos acerca dos contatores eletromecânicos, sistemas embarcados, Internet das Coisas, protocolo de comunicação MQTT e as ferramentas utilizadas para o desenvolvimento deste trabalho. No Capítulo 3 são apresentados os materiais e métodos utilizados no desenvolvimento do projeto, incluindo os requisitos funcionais e não funcionais acerca do projeto do sistema embarcado desenvolvido, a arquitetura IoT do sistema, as etapas de projeto eletrônico e mecânico, os diagramas dos códigos do *firmware*, e as informações sobre plataforma IoT escolhida para validação. O Capítulo 4 aborda os resultados obtidos a partir a utilização do protótipo finalizado. E, por fim, o Capítulo 5 traz as considerações finais do trabalho, como a comparação entre os resultados e o esperado para o projeto, bem como as possíveis melhorias.

2 Fundamentação teórica

Neste capítulo são apresentados os conceitos sobre contatores eletromecânicos e suas aplicações em sistemas industriais. Além disso, são abordados conceitos sobre Sistemas Embarcados e Internet das Coisas (IoT), como dispositivos de hardware, protocolos de comunicação e plataforma IoT.

2.1 Contator eletromecânico e seu papel em sistemas industriais

Na indústria há uma grande quantidade de dispositivos elétricos e eletromecânicos que realizam alguma ação de atuação em processos industriais. Esses dispositivos possuem funções específicas nos circuitos, as quais podem estar relacionadas com a proteção de componentes e/ou pessoas; a sinalização de processos; o acionamento de maquinários e demais funções. De modo geral, os circuitos implementados em quadros ou painéis elétricos podem ser divididos em: circuito de força e circuito de comando.

Os circuitos de força contêm os componentes que permitem a passagem de corrente elétrica para acionar a máquina quando o comando for realizado, além de proteger a máquina. Os componentes do circuito de força são mais robustos devido aos altos valores de corrente e tensão requeridos para acionar a carga. Por outro lado, os circuitos de comando contêm os componentes que implementam a lógica de funcionamento e a sinalização dos quadros, necessitando de valores de corrente elétrica menores.

A instalação desses dispositivos é feita comumente em quadros, cabines e painéis elétricos, sendo possível encontrar formatos e tamanhos que variam de acordo com a quantidade e dimensões dos componentes a serem instalados. A alimentação geral desses dispositivos pode ser feita em série com uma chave seccionadora, a qual possui a importante função de garantir a segurança em situações que exijam o desligamento dos componentes, bem como pode ser usada para seccionar o circuito geral de alimentação antes de realizar uma possível manutenção do quadro, cabine ou do painel (HOJAIJ, 2023).

Os componentes elétricos de proteção são responsáveis por garantir a segurança do maquinário em si, assim como a segurança humana. Estes componentes podem ser: fusíveis; disjuntores; disjuntores motores; relés de sobrecarga, entre outros. Cada um possui uma função específica e a sua localização no circuito pode variar. Os fusíveis permitem uma resposta rápida na proteção contra curtos-circuitos, porém exigem a troca ao operar, uma vez que o curto-circuito implica no rompimento do seu filamento interno. Os disjuntores realizam a proteção contra curtos-circuitos através do seccionamento do circuito em que o mesmo está inserido, sendo necessário somente acioná-lo novamente após a manutenção

do sistema (HOJAIJ, 2023).

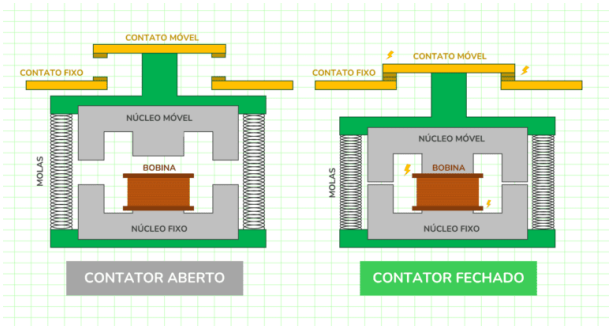
Com relação à sinalização na indústria, a sua aplicação pode variar de acordo com o ambiente em que o maquinário se encontra. Por exemplo, caso a indústria seja composta por máquinas que geram muito ruído, a sinalização do tipo sonora não seria tão eficiente, assim como a situação em que os maquinários encontram-se muito distantes do operador. Nesse último caso, seria mais interessante a utilização de dispositivos de sinalização visual como lâmpada ou LED. Além disso, os dispositivos de sinalização juntamente com os de comando viabilizam a elaboração de uma interface entre a máquina e o operador.

Os dispositivos de comando permitem a execução de uma lógica para acionamento das máquinas do sistema, seja de forma manual ou automática. No caso da atuação manual, o operador pode acionar através de chaves manuais ou botoeiras. Os quadros de comando que permitem a atuação automática, normalmente, utilizam elementos controladores para realizar a automação, seja controlador industrial ou controlador lógico programável CLP.

A automação dos acionamentos de motores elétricos é algo imprescindível na indústria, pois permite a obtenção de respostas rápidas e precisas ao ligar ou desligar esses esquipamentos. Dessa forma, um dispositivo fundamental na indústria, responsável por atuar no circuito de força dada uma ação proveniente do circuito de comando é o contator eletromecânico. O primeiro contator foi inventado pela antiga indústria *Telemecanique* em 1924, atualmente denominada *Schneider Electric* (ELECTRIC, 2021), e nos dias atuais ainda estão presentes na maioria dos quadros industriais que implementam acionamentos de motores e cargas em geral.

O contator eletromecânico é composto por: bobina, núcleo, contatos e mola. A bobina, ao ser alimentada, é responsável por gerar um campo eletromagnético que irá atrair um componente ferromagnético móvel do contator. Esse componente ferromagnético móvel é uma parte do núcleo, sendo a outra parte também ferromagnética, mas fixa. Ambas as partes são separadas por uma mola, a qual garante um estado de repouso do núcleo quando não houver atração eletromagnética da bobina. A parte móvel do núcleo é ligada a uma parte móvel dos contatos, estes constituídos de placas metálicas, e a movimentação dessas partes móveis irá comutar os contatos de força e contatos auxiliares presentes no contator (SANTOS, 2022).

Figura 1 – Partes do Contator Eletromecânico.



Fonte: Site da *Schneider Electric*. Acesso em 02-12-2023. Disponível em: <https://blog.se.com/br/eletrica/2022/06/14/o-que-e-um-contator-e-qual-sua-funcao/>.

O objetivo do circuito de comando é acionar as bobinas dos contatores para que os contatos dos contatores comutem. A comutação dos contatos do contator permite a passagem de corrente para a máquina no circuito de força. O contator é um elemento que faz parte dos dois circuitos: circuito de comando e circuito de força. A Figura 2 apresenta um contator do fabricante *Schneider Electric*.

Figura 2 – Contator do fabricante *Schneider Electric*.

Fonte: Site da *Schneider Electric*. Acesso em 02-12-2023. Disponível em: <https://www.se.com/br/pt/>.

Um termo amplamente utilizado para a quarta revolução industrial, caracterizada no cenário atual, é a Indústria 4.0. Essa revolução trouxe, entre outros aspectos, conceitos e tecnologias que permitem conectar o mundo físico ao digital (SCHWAB, 2017). Considerando essas tecnologias, torna-se possível integrar máquinas dos sistemas industriais através da Internet, permitindo um acesso rápido e em tempo real dos dados dos processos. No entanto, o custo de atualização ou modernização das máquinas para estabelecer uma

comunicação com a Internet nem sempre é algo acessível, e muitas vezes o espaço disponível nos quadros ou panéis elétricos não é suficiente para a montagem de novos dispositivos, como um CLP.

Considerando sistemas industriais que contêm máquinas que são acionadas por contatores, uma possível solução para a supervisão de processos industriais é o monitoramento dos contatores. O contator pode ser conectado à Internet através de um dispositivo IoT, permitindo o monitoramento do processo em tempo real por parte dos supervisores e gerentes da fábrica. Esse dispositivo IoT pode ser projetado para ser acoplado ao contator, semelhante a um bloco auxiliar, com o objetivo de detectar as variações de acionamentos do mesmo.

2.2 Sistemas Embarcados

Em diversos itens eletrônicos é possível notar que suas funcionalidades são bem específicas para a sua aplicação, como é o caso de alguns eletrodomésticos: geladeira; micro-ondas; televisores, ar-condicionados, entre outros. Esses itens possuem funcionalidade única, o que impede sua utilização em aplicações diferentes das quais foram projetados. Para esses tipos de sistemas, dá-se o nome Sistemas Embarcados (BARROS; CAVALCANTE, 2010).

Para que sejam projetados, os sistemas embarcados necessitam de uma lista de requisitos funcionais e não funcionais, os quais servirão para definir as suas aplicações e o seu funcionamento. Os requisitos funcionais estão mais voltados às funções que são aparentes aos usuários desse sistema; por outro lado, os requisitos não funcionais remetem aos aspectos intrínsecos ao funcionamento interno do sistema. Alguns dos parâmetros utilizados para comentar os requisitos não funcionais são: confiabilidade, manutenabilidade, disponibilidade, segurança e confidencialidade (ZURITA, 2011).

Os sistemas embarcados são caracterizados por apresentarem funções específicas, baixo consumo de energia, tamanho reduzido e operação em tempo real. A limitação do consumo de energia, o tamanho reduzido e as funções dedicadas são fatores que motivam a escolha de microcontroladores para o desenvolvimento de sistemas embarcados. Um fator importante é identificar a limitação de recursos computacionais, como memória e processamento, dos dispositivos de hardware para o sistema embarcado.

O microcontrolador é um *chip* caracterizado por possuir periféricos de entradas e saídas, memórias e processador integrados. Esses periféricos normalmente são componentes optados para suprimir uma necessidade em *hardware* e/ou *software* que o sistema embarcado irá ter.

Devido aos recursos de processamento limitados ao comparar com dispositivos

de funções genéricas (computadores), os sistemas embarcados normalmente possuem características que podem ser cruciais para o seu funcionamento. Uma delas é a possível necessidade de operar em tempo real. O Sistema Operacional em Tempo Real (RTOS) diferencia de um sistema operacional comum devido a sua maior exigência em finalizar uma dada função ou tarefa (*task*), em um tempo predeterminado (ZURITA, 2011).

O tempo de processamento pode ser um requisito importante para o sistema embarcado desenvolvido. Dessa forma, existem recursos que atuam intervindo em problemas de mau funcionamento, como circuitos de detecção de travamento no *software*. Um recurso muito utilizado é o WDT (*WatchDog Timer*). Essa funcionalidade pode ser configurada para interromper uma dada tarefa ou função ao perceber que houve um problema na execução. Por exemplo, é possível definir um tempo máximo de execução em uma tarefa e relacionar com a atuação de um *WatchDog*, fazendo-o resetar o *hardware* caso essa tarefa exceda o tempo máximo definido.

Em sistemas embarcados, é possível ainda haver a presença de componentes denominados SoC. Esses componentes diferem dos microcontroladores por, muitas vezes, possuírem uma capacidade maior de processamento, integrando também mais periféricos em um único *chip*. Alguns exemplos desses dispositivos são os SoC fabricados pela ESPRESSIF, como as versões ESP32-S2, ESP32-S3, ESP32-C3 entre outros.

Para facilitar a fase de prototipagem em projetos de sistemas embarcados, existem algumas placas de desenvolvimento *DevKit* (*development kit*) baseadas em SoC. Essas placas podem possuir modelos e versões diferentes, podendo ainda serem fabricadas por empresas distintas. Entre vários modelos, alguns exemplos de DevKit baseados em versões de SoC da ESPRESSIF são:

- ESP32-C3-DevKitC-02: fabricado pela própria ESPRESSIF;
- ESP32-PICO-KIT: fabricado pela própria ESPRESSIF;
- DOIT ESP32 DevKit V1: fabricado pela DOIT;
- Wemos Lolin ESP32 OLED: fabricado pela Wemos e etc;
- Seeed Studio XIAO ESP32C3: fabricado pela Seeed Studio;
- Seeed Studio XIAO ESP32S3: fabricado pela Seeed Studio;

Todas essas placas de desenvolvimento têm em comum a possibilidade de estabelecerem conexões com a Internet, configurando assim, placas de desenvolvimento para sistemas embarcados baseadas em internet das coisas IoT.

Figura 3 – DevKit Modelo ESP32-C3-DevKitC-02.

Fonte: *ESPRESSIF*. Acesso em 25-11-2023. Disponível em: <https://www.espressif.com/en/products/devkits>

2.2.1 FreeRTOS

Um exemplo de sistema operacional em tempo real é o *kernel* de código aberto denominado FreeRTOS. Esse sistema permite a criação de tarefas orientadas a eventos, as quais podem ou não serem implementadas com interrupções, podendo funcionar de forma periódica ou não (MIRANDA BRUNO DOURADO; CARMINATI., 2021). Ao estar utilizando o *kernel* do FreeRTOS, é possível definir o modo preemptivo ou colaborativo para as tarefas.

No modo preemptivo, as prioridades definidas para cada tarefa são consideradas pelo *kernel* no escalonamento entre elas. Dessa forma, é possível definir níveis de prioridade entre as tarefas, deixando o escalonador que está presente no *kernel*, gerenciar essas trocas. Nesse modo, caso uma tarefa exceda o limite de tempo definido, sua execução será interrompida pelo escalonador, dando oportunidade para outra tarefa com a prioridade mais alta.

O modo colaborativo é implementado para que o *kernel* não interrompa a execução das tarefas durante o processo. Dessa forma, cabe ao desenvolvedor definir tempos máximos de execução, interrupções ou eventos que permitam ao escalonador selecionar outra tarefa para ser executada.

Uma forma de implementar as tarefas no freeRTOS é através de *Event Groups* ou Grupo de Eventos. Nesse contexto, as tarefas são sincronizadas através de pontos de sincronização, que são os locais no código onde as tarefas aguardam sem consumir a CPU.

Para que a tarefa entre em execução, são definidos os *Event Bits* ou Bits de Eventos, que são verificados através de máscaras de um ou mais bit. Dessa forma, a tarefa aguarda essa máscara de bit mudar para um valor desejado - valor esse setado por outras tarefas ou funções - para então iniciar a sua execução (FREERTOS.ORG, 2023).

2.3 Internet das Coisas

A Internet das Coisas IoT é um termo aplicado a um conjunto de objetos capazes de estabelecerem comunicações entre si através da rede Internet. A conexão dos objetos através da Internet possibilita ações de monitoramento e controle de forma remota. A implantação de soluções de IoT na indústria pode trazer grandes resultados, como aumentar a eficiência dos processos de manutenção das máquinas e, consequentemente, aumentar a produtividade a partir de tomada de decisões com base nos dados coletados das máquinas. Os dispositivos IoT aplicados em cenários industriais, denominam-se dispositivos IIoT.

A Internet das Coisas vem se destacando cada vez mais nos últimos anos, e com a constante necessidade de avanços tecnológicos, a sua aplicação tem impactado diversos setores, com ênfase no setor industrial. O termo surgiu com o pesquisador Kevin Ashton em meados de 1999, no Instituto de Tecnologia de Massachusetts. Kevin tinha a pretensão de tornar os dispositivos eletrônicos comunicáveis entre si, de modo que os dados trocados entre eles seriam agora gerenciados por um computador, e isso iria garantir uma rápida tomada de decisões, com uma maior facilidade de visualização dos dados referentes aos dispositivos (BENEDITO et al., 2018).

De forma mais sintética, um sistema IoT conecta uma coisa à *Internet* através de protocolos de comunicação. Para isso, podem ser utilizados sistemas embarcados com microcontroladores ou SoC, que possuam periféricos suficientes para realizar as funções desejadas. No entanto, um sistema embarcado não necessariamente é um sistema IoT, pois não há a obrigatoriedade de conexões com a Internet em sistemas embarcados, podendo esses, realizar outros tipos de protocolos de comunicação, como *Bluetooth*, *ZigBee* entre outros.

Para os sistemas IoT, existem diferentes protocolos de comunicação para conectar o dispositivo à Internet, como o protocolo de camada física e enlace Wi-Fi (IEEE 802.11) e os protocolos de aplicação MQTT e HTTP.

2.3.1 Protocolo de Comunicação MQTT

Em um contexto de dispositivos IoT, um protocolo de camada de aplicação muito utilizado é o MQTT. Esse protocolo foi lançado no ano de 1999 pelo Dr. Andy Stanford-Clark da IBM e por Arlen Nipper da atual Eurotech. Esse protocolo foi projetado para garantir uma boa segurança no envio dos dados, através de encriptações de mensagens, e

uma boa confiabilidade através de 3 níveis de qualidade de serviço QoS. É também um método de comunicação bidirecional, onde os dispositivos se comunicam com a nuvem, e a nuvem se comunica com os dispositivos (MQTT.ORG, 2022).

Entre os conceitos vistos em uma comunicação MQTT estão: *broker*; publicação e inscrição; tópicos MQTT; qualidades de serviço QoS, mensagens retidas (*retained messages*) e mensagens *Will* (SONI; MAKWANA, 2017).

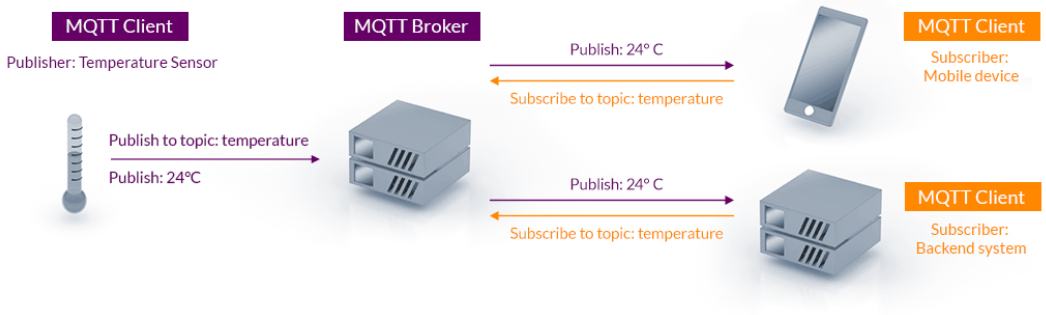
O protocolo MQTT utiliza o modelo publicar-assinar (*publish–subscribe*). O *broker* é o responsável por gerenciar o controle das mensagens que são enviadas pelos clientes (publicações), bem como gerir as inscrições (assinantes) realizadas em tópicos criados.

Como dito anteriormente, o protocolo em questão é bidirecional, logo, para que as mensagens sejam trocadas, o cliente (*client*) deve se inscrever em um dado tópico existente. Essa inscrição garante que o cliente receba todas as mensagens que forem publicadas nesse tópico, podendo também publicar no tópico inscrito. As definições de tópicos garantem uma organização no direcionamento das mensagens enviadas ao *broker*, de modo que somente os clientes inscritos nos tópicos, receberão as mensagens publicadas nesses.

Para que o cliente receba as mensagens já publicadas em um tópico, ao realizar sua inscrição no mesmo, o protocolo disponibiliza as mensagens retidas (*retained messages*). Essas mensagens são guardadas no *broker* para que um cliente, ao se inscrever em um tópico, receba esses dados retidos mesmo realizando a inscrição após os seus envios. Por outro lado, as mensagens *Will*, garantem que um dispositivo conectado ao *broker* possa notificar um ou mais tópicos, caso esse cliente seja desconectado (SONI; MAKWANA, 2017).

A publicação das mensagens no *broker* pode ser definida em até três níveis de qualidade do serviço de envio de mensagens (QoS). O primeiro nível, representado por QoS0, não garante o envio da mensagem, de modo que a mesma é enviada somente uma vez. O segundo nível QoS1 garante que a mensagem seja enviada pelo menos uma vez, podendo então ser enviada mais vezes. Por último, o nível QoS2, a mensagem será enviada exatamente uma vez (SONI; MAKWANA, 2017).

Figura 4 – Comunicação MQTT.



Fonte: *MQTT.org*. Acesso em 20-11-2023. Disponível em: <https://mqtt.org/>

A Figura 4 apresenta um exemplo de comunicação MQTT, em que um sensor de temperatura é um cliente de um determinado *broker* e publica em um tópico chamado *temperature* o valor de temperatura mensurado. O *broker*, por sua vez, fornece esse valor de temperatura para dois clientes que estão inscritos no mesmo tópico ("*temperature*") que o sensor publicou. Esses clientes são exemplificados como um dispositivo móvel (celular) e um sistema de *backend*.

Nos projetos IoT que utilizam o protocolo MQTT, é necessário incluir alguma aplicação ou plataforma que possa receber os dados dos dispositivos através da assinatura no *broker*. Existem várias ferramentas que implementam *brokers*. No entanto, alguns serviços podem ainda possuir acesso parcial ou totalmente limitado, não sendo considerados serviços de código aberto. Um exemplo desse tipo de serviço é o *broker* da plataforma Adafruit IO.

Alguns *broker* que se enquadram em serviços disponíveis e públicos são listados a seguir.

1. Mosquitto: "test.mosquitto.org"(Porta TCP: 1883).
2. HiveMQ: "broker.hivemq.com"(Porta TCP: 1883).
3. Eclipse: "mqtt.eclipse.org"(Porta TCP: 1883).
4. Fluux: "mqtt.fluux.io"(Porta TCP: 1883).

Uma forma de visualizar, tratar e armazenar os dados trocados entre clientes e *broker* é através da utilização de plataformas IoT.

2.3.2 Plataforma IoT

As plataformas IoT são serviços empregados em sistemas de *Internet* das Coisas, capazes de gerenciar uma grande quantidade de dispositivos conectados. Esse gerenciamento

pode ser útil na análise das informações recebidas, podendo também elaborar um histórico dos dados trocados. As plataformas podem ainda ser consideradas serviços abertos para sua total utilização, parcialmente abertos ou limitados. Segundo um estudo realizado por Vogel et al. (VOGEL BAHTIJAR, 2020), entre as plataformas IoT abertas mais aceitas, destacam-se as plataformas *NodeMCU* e *ThingSpeak*.

Uma plataforma IoT de acesso parcialmente limitado, mas que possui fácil utilização, e para certas aplicações pode se mostrar muito útil, é a Adafruit IO. Na sua forma gratuita, essa plataforma permite a criação de até 2 dispositivos para simulação na plataforma; permite também a elaboração de até 10 tópicos no *broker* MQTT, denominados *feeds*. Para a visualização interativa dos dados trocados entre *broker* e clientes, é permitida a criação de até 5 *dashboards*. O armazenamento dos dados nessa forma gratuita pode ser feito considerando um histórico de até 30 dias.

Veja na figura 5 um exemplo de *dashboard* implementado na plataforma AdafruitIO para o monitoramento de temperatura e humidade.

Figura 5 – AdafruitIO Exemplo de *Dashboard*.

Fonte: *AdafruitIO*. Acesso em 01-12-2023. Disponível em:
<https://learn.adafruit.com/no-code-humidity-and-temp-tracker>

Os serviços da Adafruit IO com os sistemas de comunicação MQTT não são compartilhados entre outros usuários, de modo que ao criar uma conta na plataforma, o seu acesso é exclusivo. O acesso ao *broker* da plataforma é através do seu nome (*username*) e sua chave de acesso (*active_key*).

Neste projeto foi desenvolvido um dispositivo IoT para atuar como um sistema embarcado em um contator eletromecânico da marca Schneider. Esse dispositivo terá um acoplamento do tipo frontal, semelhante a um bloco auxiliar de extensão de portas para contatores, e terá a função de monitorar o estado (ligado ou desligado) do contator aclopado. A comunicação para a troca de informações entre o dispositivo e a *internet* será feita através do protocolo de aplicação MQTT, e a plataforma utilizada para a visualização

e armazenamento dos dados trocados será a AdafruitIO.

3 Metodologia

O presente capítulo tem o objetivo de descrever o desenvolvimento do dispositivo SmartContact, especificando os requisitos funcionais e não funcionais do sistema, bem como a arquitetura geral do projeto e suas funcionalidades.

3.1 Requisitos do Projeto

Os requisitos gerais do projeto podem ser divididos em: funcionais e não funcionais. Os requisitos funcionais estabelecem as características de funcionamento que o dispositivo deve possuir para atender às necessidades desejadas. Por outro lado, os requisitos não funcionais tratam as características técnicas voltadas ao desempenho e atributos de qualidade do sistema.

O dispositivo desenvolvido neste projeto deve ser capaz de detectar o estado do contator, o qual representa o estado de alguma máquina em um sistema industrial, e disponibilizar o dado para o broker MQTT de alguma aplicação. No entanto, algumas funcionalidades de fácil configuração do dispositivo são necessárias para tornar o dispositivo uma solução *plug-and-play*.

3.1.1 Requisitos Funcionais

Os requisitos funcionais são definidos a seguir conforme o funcionamento do dispositivo SmartContact.

A primeira ação do dispositivo ao ser energizado é verificar se existem informações de acesso a alguma rede Wi-Fi, como SSID e chave de segurança, salvos na memória não volátil NVS e se há algum registro de atualização de data e hora no momento em que o dispositivo iniciou. Além disso, o dispositivo deve salvar o estado do contator, bem como o horário em que realizou a leitura do estado.

Caso não sejam encontrados dados de rede Wi-Fi, o SmartContact deve iniciar um modo de configuração que ficará disponível para o usuário inserir as informações de rede do ambiente. O dispositivo será o AP (*Access Point*) de uma rede Wi-Fi e o usuário deve se conectar diretamente ao dispositivo através de algum aparelho eletrônico. Em seguida, o usuário deve acessar uma página WEB via endereço IP do dispositivo para inserir as informações de rede.

A comunicação entre o dispositivo IoT e o aparelho do usuário é através do protocolo de comunicação HTTP. A página WEB deve disponibilizar campos para o usuário inserir

os seguintes dados: nome (SSID) e senha da rede Wi-Fi, o endereço do *broker* MQTT, tópico para publicação dos dados capturados, e o tempo máximo de duração do modo DSM (*Deep Sleep Mode*). Esse tempo define o período que o dispositivo ficará em modo de economia de energia DSM, ou "Modo de Sono Profundo", após envio dos dados.

Após a inserção dos dados na página WEB, o dispositivo deve se conectar à rede Wi-Fi do local e atualizar as suas informações de data e hora através do protocolo NTP (*Network Time Protocol*).

O envio dos dados capturados pelo dispositivo deve ser feito pelo protocolo de comunicação MQTT através de um arquivo no formato JSON, e após o envio, o dispositivo deve entrar em modo de economia de energia DSM. Para que seja possível redefinir os dados informados pelo usuário, o dispositivo deve ainda ter a função de *reset factory*.

Por último, o dispositivo deve reiniciar de acordo com as seguintes condições: tempo máximo de DSM definido pelo usuário, reinício forçado pelo usuário ou mudança no estado do contator. A tabela 1 resume tais requisitos.

Tabela 1 – Requisitos Funcionais: conectividade e comunicação.

Numeração	**Nome**	**Descrição**
[**RF01**]	Verificar dados salvos	Verificação de dados salvos na memória não volátil NVS.
[**RF02**]	Capturar o estado do contator	Captura e armazenamento do estado do contator na NVS, bem como o horário da aquisição do dado.
[**RF03**]	Iniciar o modo AP	Modo *Access Point* para comunicação com usuário.
[**RF04**]	Definir as configurações de rede	Execução de servidor WEB para página de preenchimento de informações sobre a rede Wi-Fi e dados do broker MQTT do ambiente.
[**RF05**]	Iniciar o modo STA	Conexão com rede Wi-Fi.
[**RF06**]	Ajustar horário via NTP	Ajuste de data e hora via protocolo NTP.
[**RF07**]	Iniciar comunicação MQTT	Comunicação com *broker* MQTT no tópico especificado pelo usuário, com os dados formatados em JSON.
[**RF08**]	Iniciar modo de economia de energia	Modo de Sono Profundo (DSM) após o envio dos dados para o *broker*.
[**RF09**]	Desligar modo de economia de energia	Desligar o modo de DSM através de um tempo máximo definido pelo usuário, ou através de uma reinicialização forçada.
[**RF10**]	*Reset Factory*	Exclusão dos dados salvos na NVS.

Fonte: Elaborado pelo autor.

Com relação aos requisitos relacionados à parte conexão do SmartContact com o contator, o dispositivo deve ser projetado para encaixar em um contator *Schneider* seguindo o padrão de encaixe de bloco auxiliar com agrupamento frontal.

O dispositivo deve ainda ser alimentado por uma bateria, possuindo uma chave *ON/OFF* para ligar/desligar o mesmo. Após energizado e encaixado ao contator, o dispositivo deve iniciar a verificação do estado do contator, podendo esse está ligado ou desligado.

O SmartContact deve conter sinalizações para indicar o seu estado ao usuário. A interface visual entre o dispositivo e o usuário deve ser através de dois LED's. Um LED deve informar que o dispositivo IoT está ligado, e o outro deve informar quando o mesmo estiver capturando um dado, estabelecendo conexão ou em algum processo de envio de dados. O usuário poderá interagir diretamente com o dispositivo através de um botão do tipo *Push Button*. Essa interação servirá para retornar do estado de DSM para o modo ativo de forma forçada, sendo possível também realizar o *reset factory* caso o botão seja pressionado por mais de 10 segundos. A tabela 2 resume tais requisitos.

Tabela 2 – Requisitos Funcionais: Interfaces.

Numeração	Nome	Descrição
[**RF11**]	Encaixe frontal no contator	O projeto mecânico do dispositivo deve permitir um encaixe frontal em contator *Schneider* e seja dimensionalmente compatível.
[**RF12**]	Alimentação via bateria	Deve ser alimentado através de uma bateria compatível.
[**RF13**]	Possuir chave *ON/OFF*	Deve permitir ligar/desligar o dispositivo, seccionando a alimentação com a bateria.
[**RF14**]	Possuir *Push Button*	Deve possuir um botão do tipo *Push Button* capaz de permitir ao usuário forçar o retorno do modo DSM para o modo ativo pressionando uma vez. Deve também permitir resetar os dados salvos na memória caso seja pressionado por mais de 10 segundos.
[**RF15**]	Possuir dois LED	Deve possuir dois LED's de interface entre usuário e dispositivo.

Fonte: Elaborado pelo autor.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais do projeto são especificados a seguir.

Com relação à eficiência energética, o SoC utilizado no projeto deve possuir um modo de economia de energia que realize um consumo muito baixo, com critério de aceitação de aproximadamente 5μA, permitindo o prolongamento do tempo de vida útil da bateria.

O dispositivo pode se conectar apenas na rede informada pelo usuário durante a etapa de *Setup*. Além disso, a publicação do dado deve ser apenas no broker MQTT e no tópico informados durante o *Setup*. Com relação à comunicação MQTT, não será necessária a confirmação de recebimento da mensagem pelo *broker*, correspondendo a um QoS igual a 0.

O dispositivo deve atender também aos requisitos de segurança e confiabilidade da conectividade com a rede Wi-Fi, com o servidor do protocolo NTP e com o *broker* MQTT. Para isso, devem ser utilizadas funções de *WatchDog* para garantir um tempo máximo de 60 segundos durante cada tentativa de conexão. Se esse tempo for alcançado, o dispositivo deve entrar em modo de economia de energia e retornar apenas quando uma condição de reinício ocorrer novamente. Essa condição impede o consumo indesejado de bateria em

situações inesperadas, como a desconexão com o servidor utilizado na comunicação NTP, desconexão da rede Wi-Fi ou até problemas atrelados ao *broker* em si.

Com relação à confiabilidade da aquisição do dado do estado do contator, o dispositivo deve capturar o estado do contator juntamente com a data e hora de incialização do disposito. Dessa forma, a primeira tarefa realizada ao identificar e salvar o estado mais atual do contator deve ser salvar o horário lido no RTC (*Real Time Clock*) caso o mesmo esteja configurado. Para essa verificação de configuração do RTC, deve existir uma variável salva em sua própria memória, que receberá o valor 1 quando uma configuração for feita pelo protocolo NTP. Na situação em que a memória do RTC for apagada, o dispositivo deve iniciar uma contagem em segundos, até o momento de reconfiguração NTP, para assim, estimar o horário em que o mesmo capturou o estado do contator. A tabela 3 resume tais requisitos.

Tabela 3 – Requisitos Não Funcionais.

Numeração	Classificação	Descrição
[**RNF01**]	Eficiência Energética	Deve possuir um modo de economia de energia que realize um consumo ínfimo, com critério de aceitação de aproximadamente 5 μA.
[**RNF02**]	Segurança/Conexão Wi-Fi	Deve se conectar somente à rede Wi-Fi informada pelo usuário.
[**RNF03**]	Segurança/Comunicação MQTT	Deve se comunicar somente ao *broker* MQTT informado pelo usuário, utilizando um QoS igual a 0.
[**RNF04**]	Confiabilidade/Wi-Fi	Deve tentar se conectar durante um período de no máximo 60 segundos. O *WatchDog* deverá reiniciar o dispositivo após esse tempo. Na inicialização do dispositivo, caso o motivo seja a interrupção pelo *WatchDog* na tarefa em questão, o dispositivo deverá iniciar o modo de sono profundo DSM.
[**RNF05**]	Confiabilidade/NTP	Deve tentar se conectar durante um período de no máximo 60 segundos. O *WatchDog* deverá reiniciar o dispositivo após esse tempo. Na inicialização do dispositivo, caso o motivo seja a interrupção pelo *WatchDog* na tarefa em questão, o dispositivo deverá iniciar o modo de sono profundo DSM.
[**RNF06**]	Confiabilidade/Tópico MQTT	Deve tentar se conectar durante um período de no máximo 60 segundos. O *WatchDog* deverá reiniciar o dispositivo após esse tempo. Na inicialização do dispositivo, caso o motivo seja a interrupção pelo *WatchDog* na tarefa em questão, o dispositivo deverá iniciar o modo de sono profundo DSM.
[**RNF07**]	Confiabilidade/Aquisição de Dados	Deve capturar o horário logo após a verificação do estado do contator. Caso o horário pelo RTC esteja desconfigurado, deverá estimar esse tempo através de uma contagem que inicia após a aquisição do estado e finaliza após a configuração via NTP.
[**RNF08**]	Facilidade de Instalação	Deve ser fácil de ser instalado, se considerando *Plug-and-Play*. Além disso, não deve interferir de forma alguma nas instalações elétricas do quadro ou painel o qual será instalado.

Fonte: Elaborado pelo autor.

3.2 Arquitetura IoT do Projeto

A arquitetura do projeto IoT apresenta as tecnologias de hardware e software utilizadas no projeto e suas conexões.

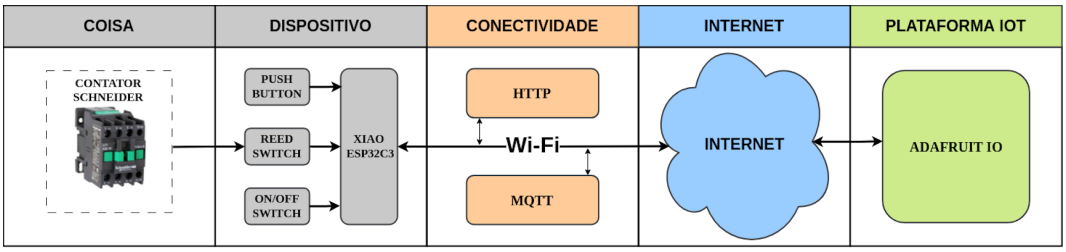
Para atender aos requisitos propostos para o SmartContact, foi utilizado o *devkit Seeed Studio Xiao ESP32C3*, o qual é baseado no SoC ESP32-C3. A escolha do ESP32-C3 ocorreu devido às suas características de hardware, como as interfaces de comunicação,

e do tamanho reduzido. O projeto exige um dispositivo com tamanho reduzido para ser compatível a um bloco auxiliar de encaixe frontal para contatores.

O sensor *Reed Switch* foi escolhido para detectar do estado do contator. Um imã deve ser instalado na parte externa móvel do contator para possibilitar a detecção do seu estado. Quando o sensor *Reed Switch* detecta o imã significa que o contator está desligado. Ao energizar o contator, a parte móvel é acionada e o sensor *Reed Switch* não detecta o imã próximo ao dispositivo. Além disso, o dispositivo IoT é composto por um *Push Button* e também uma chave para seccionar a alimentação do *devkit*.

O dispositivo IoT se conecta à Internet através do protocolo Wi-Fi e utiliza dois protocolos de aplicação: MQTT e HTTP. O protocolo HTTP é utilizado durante o momento de *setup* inicial. O SmartContact disponibiliza uma interface WEB para que o usuário insira os dados de configuração da rede Wi-Fi, os quais o dispositivo deve manter a conexão durante o envio dos dados para a aplicação. O protocolo MQTT é utilizado para enviar o dado no formato JSON contendo a informação do *status* do contator junto com o horário da aquisição do dado. Os dados são enviados para um broker MQTT na plataforma Adafruit IO.

Figura 6 – Diagrama da Arquitetura IoT.



Fonte: Elaborado pelo autor.

O diagrama da arquitetura IoT da figura 6 apresenta as relações entre os elementos do projeto, desde a coisa (*thing*) até a plataforma IoT. Observa-se que o contator é conectado à Internet através dos sensores e SoC ESP32-C3 por meio dos protocolos de comunicação Wi-Fi e MQTT. A plataforma AdafruitIO é utilizada para receber e visualizar os dados. No entanto, o dado poderia ser consumido por outra aplicação.

3.2.1 Sensoriamento

O componente *Reed Switch* é um sensor magnético que funciona como uma chave normalmente aberta. O mesmo é composto por um gás inerte que envolve duas placas metálicas no seu interior, e ao ser aproximado por campo magnético fecha o contato entre as placas. Sua tensão máxima de operação pode chegar a 200 VDC com corrente máxima

de 500 mA. Normalmente, o comprimento de um terminal ao outro é de aproximadamente 40 mm.

Figura 7 – Chave Magnética *Reed Switch*.

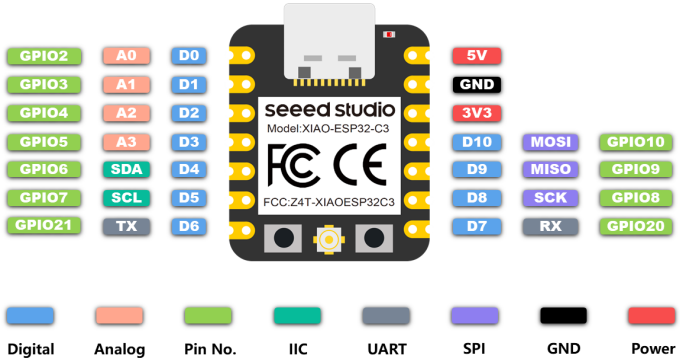
Fonte: *Smartkits*. Acesso em 22-11-2023. Disponível em:
<https://www.smartkits.com.br/reed-switch-chave-magnetica/>

Para a utilização desse sensor no projeto, se faz necessário posicionar um perqueno imã na região móvel externa do contator, de modo que o dispositivo consiga identificar as variações do acionamento simplesmente pela variação da proximidade entre o *Reed Switch* e o imã.

3.2.2 Xiao ESP32C3

Com suporte ao protocolo de comunicação IEEE 802.11 b/g/n (Wi-Fi) e *Bluetooth* 5.0 LE, a placa de desenvolvimento IoT *Seeed Xiao ESP32C3* pode ser utilizada com amplas possibilidades de aplicações em projetos. O ESP32C3 possui 11 portas digitais que podem ser configuradas como entradas ou saídas, e contém funcionalidades como: geração de sinal PWM (*Pulse Width Modulation*), circuito de ADC (*Analogue-to-Digital Converter*) e interfaces UART (*Universal Asynchronous Receiver/Transmitter*), I2C (*Inter-Integrated Circuit*) e SPI (*Serial Peripheral Interface*). O *Seeed Xiao ESP32C3* também fornece acesso à porta para alimentação da placa Vin, referência GND e tensão regulada de 3,3V. O regulador de tensão presente no *Devkit* é o XC6210B332MR, o qual permite uma entrada mínima de 1,5V e máxima de 6V. A Figura 8 apresenta os terminais do Xiao ESP32C3.

Figura 8 – Diagrama de pinos do Xiao ESP32C3.



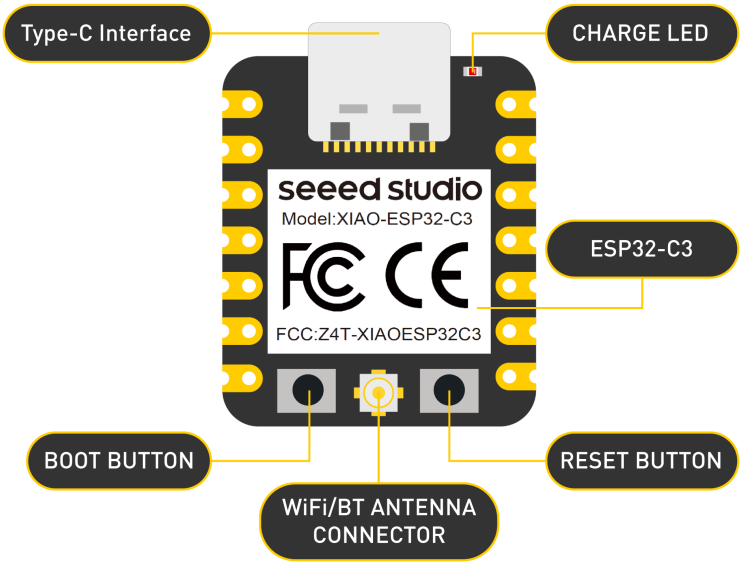
Fonte: *XIAO ESP32C3*. Acesso em 22-11-2023. Disponível em: <https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/>

Os pinos GPIO D0 e D1 foram utilizados para monitorar as interrupções dos sensores *Reed Switch* e *Push Button* respectivamente. Essa escolha se deu em função da necessidade de retornar do DSM por interrupção externa, logo necessita de GPIO atreladas ao RTC.

O projeto do dispositivo IoT contém dois LEDs. O primeiro LED está conectado ao pino D9 e o segundo LED ao pino D8. A placa possui também um LED integrado e uma memória *Flash* NVS (*Non-Volatile Storage*) de 4 MB.

Com relação à programação, o ESP32C3 pode ser programado através do *framework* da Espressif ESP-IDF, através da plataforma do Arduino (Arduino IDE) ou até via *MicroPython* (STUDIO, 2022).

Figura 9 – Seeed Studio Xiao ESP32C3.



Fonte: *Seeed Studio*. Acesso em 22-11-2023. Disponível em:
<https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/>

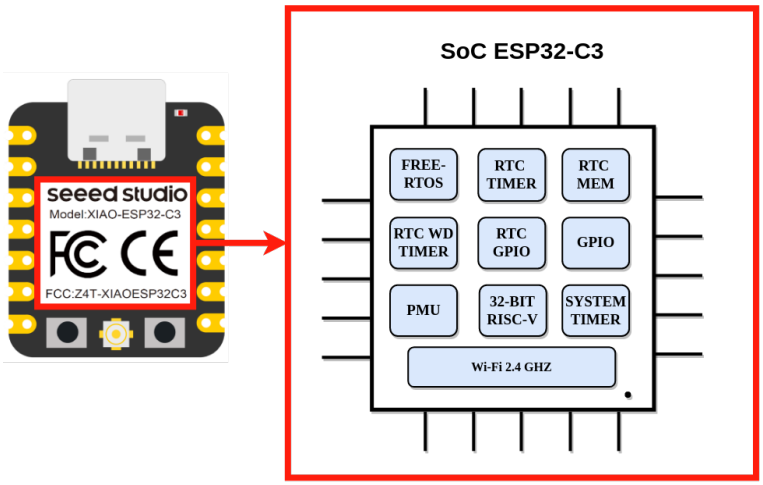
Como pode ser visto na figura 9, a placa permite conexão USB tipo C, que poderá servir como alimentação e como método para carregar o programa. Possui também conector para antena Wi-Fi ou *Bluetooth*, e dois botões "*Boot Button*"e "*Reset Button*". O botão "*Boot Button*"pode ser utilizado para iniciar o modo *boot*, que permite carregar um dado programa para a placa. E o botão "*Reset Button*"permite resetar o programa que está carregado e em processo de leitura pelo microcontrolador.

O *Devkit* é composto pelo SoC ESP32-C3, o qual possui um processador de 32 bits RISC-V de 160 MHz. A escolha dessa arquitetura baseada no SoC em questão também se deu pelo seu suporte a algumas funções e periféricos. Esses estão descritos a seguir:

- *Free*RTOS;
- Wi-Fi 2.4 GHz (Transmissor e receptor);
- RTC *Memory*;
- RTC *Timer*;
- RTC GPIO;
- GPIO;
- RTC *WatchDog Timer*.

- *System Timer*;
- PMU;
- Processador de 32 bits Risc-V;

Figura 10 – Xiao ESP32C3 SoC



Fonte: Elaborado pelo autor.

3.2.2.1 SoC ESP32-C3

O SoC ESP32C3 foi projetado para ser um chip compacto com apenas 32 pinos, processador RISC-V de único núcleo, interface Wi-Fi integrada e baixo consumo de energia. A especificação da sua nomenclatura diz respeito a possibilidade de haver ou não uma memória *Flash* encapsulada no próprio SoC, e o seu tamanho caso exista. Por exemplo, o ESP32-C3 não possui memória *Flash* encapsulada, mas possui suporte à conexão SPI externa desse tipo de memória. Já a versão ESP32-C3FN4 possui memória *Flash* encapsulada com 4MB de armazenamento (ESPRESSIF, 2023). Além da memória não volátil, algumas das configurações de memórias integradas podem ser vistas a seguir.

- 384KB de memória ROM: inicialização e funções principais;
- 400KB de memória SRAM: dados de instruções (16KB reservados para memória *cache*);
- 8KB de memória SRAM no RTC: pode ser acessada pela CPU principal e pode armazenar dados em DSM.

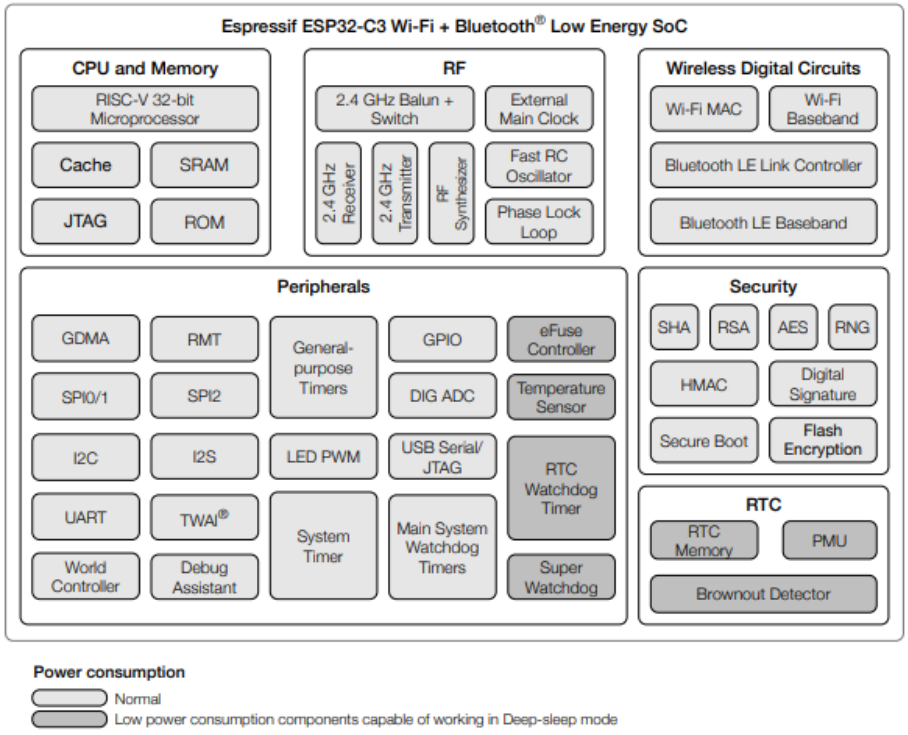
O SoC ESP32C3 suporta 4 modos de gerenciamento de energia tratados pelo PMU: *Active*, *Modem-sleep*, *Light-sleep* e *Deep-sleep*. Neste projeto são utilizados apenas dois

modos: *Active* e *Deep-sleep*. O modo ativo refere-se ao modo comum de operação, onde a CPU, os circuitos de conexão Wi-Fi e todos os periféricos estão alimentados. Esse modo será utilizado para processar os dados e realizar todos os procedimentos antes de iniciar o modo de economia de energia. O modo *Deep-sleep* se trata do modo de economia de energia, o qual mantém somente o RTC alimentado e realiza um consumo energético próximo de 5μA (ESPRESSIF, 2023).

Com relação aos periféricos analógicos integrados, o SoC possui dois conversores ADC do tipo SAR de 12 bits e um sensor de temperatura. O primeiro ADC (ADC 1) possui 5 canais calibrados de fábrica; já o segundo (ADC 2) possui somente suporte a 1 canal e o mesmo não é calibrado. O sensor de temperatura presente possui um *range* de -40 °C a 125 °C, e sua finalidade é monitorar a temperatura interna do *chip*, a qual normalmente é relativamente maior que a temperatura externa. A temperatura lida pelo sensor pode ser acessada digitalmente através da saída do ADC (ESPRESSIF, 2023). Para o dispositivo IoT desenvolvido, nenhum desses periféricos analógicos foram utilizados.

Demais periféricos e itens integrados ao SoC ESP32-C3 podem ser vistos na figura 11.

Figura 11 – ESP32C3 SoC



Fonte: *ESPRESSIF*. Acesso em 22-11-2023. Disponível em: <https://www.espressif.com/en/products/socs/esp32-c3>

3.2.3 Protocolo de Comunicação HTTP

O protocolo de aplicação HTTP é utilizado para atender aos requisitos RF03 (Iniciar o modo AP) e RF04 (Definir as configurações de rede) através de uma página WEB acessível ao usuário.

Como pode ser observado no diagrama da arquitetura IoT na figura 6, há dois protocolos de comunicação de camada de aplicação sendo utilizados no projeto.

O primeiro se trata do protocolo HTTP, o qual permite ao usuário definir as configurações de conectividade do dispositivo. O cabeçalho `esp_http_server.h` foi utilizado para implementar o protocolo HTTP no ESP32-C3, fornecendo declarações e interfaces necessárias para o servidor HTTP. O protocolo HTTP é utilizado quando o dispositivo IoT entra no modo AP e gera a rede Wi-Fi denominada "SmartContact". Caso o cliente (usuário) estabeleça uma comunicação com a rede, o dispositivo IoT disponibiliza uma página WEB através do endereço IP 192.168.4.1, que pode ser acessada de qualquer navegador WEB. O dispositivo IoT responde às solicitações HTTP do cliente, enviadas via o método *GET*, retornando uma página HTML personalizada para preenchimento dos dados de configuração de rede.

Após o preenchimento, o dispositivo IoT recebe uma requisição *POST* para armazenar os dados inseridos pelo usuário na página WEB. Ao clicar no botão "salvar", os dados são enviados ao servidor, permitindo a interação bidirecional e a personalização das configurações do dispositivo de acordo com as preferências do usuário. Essa integração efetiva do protocolo HTTP no ESP32-C3 constitui um elemento fundamental para a operacionalidade e a adaptabilidade do sistema às necessidades do usuário final.

3.2.4 Protocolo de Comunicação MQTT

O protocolo de aplicação MQTT é utilizado para permitir a publicação dos dados do contator em um broker com o intuito de fornecer para a aplicação final.

O segundo protocolo de comunicação implementado se trata do MQTT. A ESPRESSIF fornece o cabeçalho `mqtt_client.h`, o qual viabiliza a implementação do protocolo MQTT no ESP32-C3. Esse cabeçalho implementa um cliente MQTT no dispositivo IoT, permitindo a inscrição ou publicação em tópicos de algum *broker*. Atualmente, o cabeçalho possui suporte ao MQTT5 e recursos como autenticação, definições de *Last Will* para situações de desconexão inesperadas e três níveis de Qualidade de Serviço QoS. O projeto foi implementado para gerar um dado no formado JSON contendo o estado do contator e o horário da coleta desse estado. O ESP32-C3 se conecta ao *broker* MQTT, cujo endereço foi informado anteriormente pelo usuário via protocolo HTTP, e em seguida se inscreve no tópico informado pelo usuário através da função `esp_mqtt_client_subscribe`. Por último, é possível publicar o dado em JSON no tópico, com o parâmetro de QoS optado.

Para essa aplicação, o QoS escolhido foi 0.

Para ambos protocolos de comunicação implementados, seus formatos de execução são baseados em eventos, o que possibilita uma execução assíncrona com o programa principal, gerando interrupções (*callback*) que ocorrem de acordo com os manipuladores de eventos criados. Por exemplo, ao receber uma resposta HTTP ou ao se inscrever no tópico MQTT, um manipulador de eventos correspondente é invocado para processar e atuar de acordo com essa resposta. Isso permite que o *firmware* continue a operar eficientemente, respondendo a eventos à medida que ocorrem.

3.3 Projeto Eletrônico

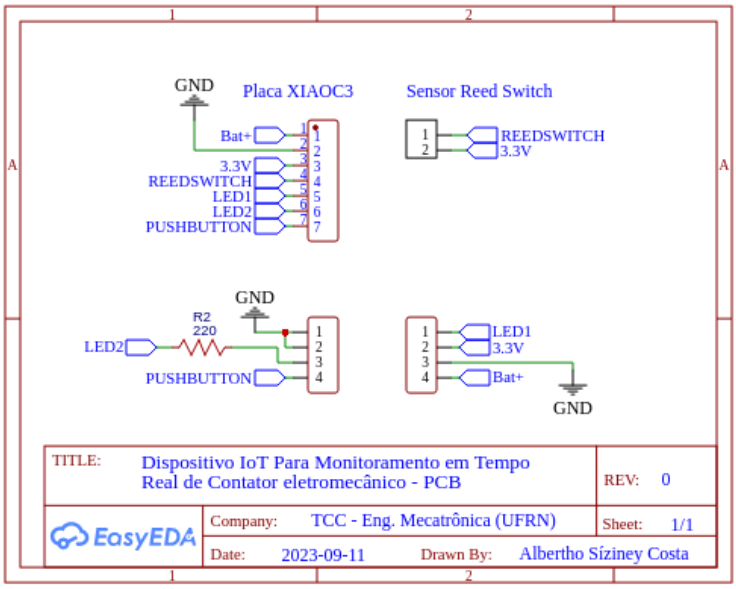
O projeto eletrônico contempla as etapas de esquemático e design da placa de circuito e a programação do microcontrolador.

As etapas de esquemático e design do projeto eletrônico foram desenvolvidas na plataforma *EasyEDA*. O EasyEDA é uma plataforma online que oferece ferramentas de design de circuitos e placas de circuito impresso (PCB). Ele é projetado para simplificar o processo de criação e design de circuitos eletrônicos, fornecendo uma interface intuitiva e recursos colaborativos entre os usuários.

3.3.1 Circuito do Esquemático

O esquemático eletrônico desenvolvido possui quatro componentes do tipo *Female Pin Header*. O primeiro componente tem capacidade para 7 conectores que são conectados ao *Devkit* a partir de um encaixe lateral. O segundo componente *Female Pin Header* possui 2 conectores utilizado para conectar os terminais do sensor *Reed Switch*. Por fim, os outros 2 componentes possuem, cada um, quatro conectores, os quais são utilizados para permitir a conexão entre os LED's, *Push Button*, chave *ON/OFF* e bateria.

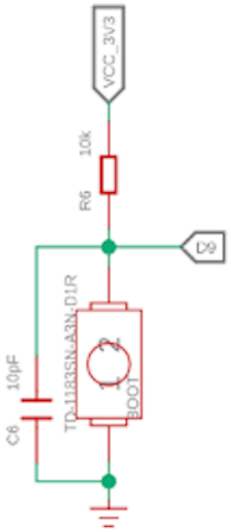
Figura 12 – Circuito Eletrônico no *EasyEDA*



Fonte: Elaborado pelo autor.

Note que somente o LED2 possui um resistor de 220Ω conectado em série para limitar a sua corrente consumida. Essa escolha se deu em virtude do esquema eletrônico do *Devkit Seeed Studio Xiao ESP32C3* já fornecer um resistor de *Pull-Up* conectado à porta D9, que foi escolhida para o LED1.

Figura 13 – *Pull-Up* XIAO ESP32C3

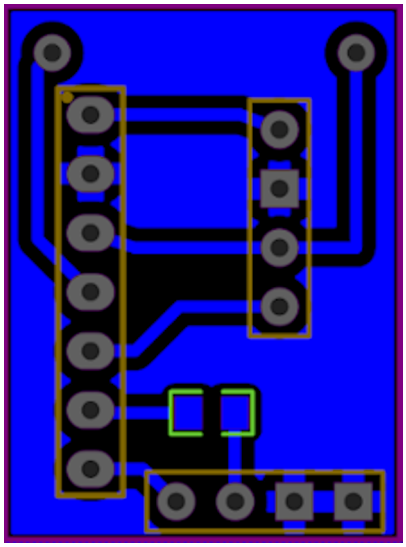


Fonte: *Datasheet*. Acesso em 15-11-2023. Disponível em: <https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/>

3.3.2 *Layout* da PCB

Após o projeto do esquemático do circuito eletrônico, foi projetado o layout da PCB no EasyEDA. Os componentes do circuito foram posicionados com base na integração com o projeto da case do dispositivo final. A Figura 14 apresenta o layout da placa.

Figura 14 – PCB - *EasyEDA*



Fonte: Elaborado pelo autor.

Os conectores do sensor *Reed Switch* foram posicionados na extremidade superior da placa, pois é necessário posicionar o sensor o mais próximo possível do imã presente no contator. Os demais componentes foram alocados em posições estratégicas para conexão com os periféricos. Os furos da placa foram dimensionados para um diâmetro de 0.8 mm e as larguras das trilhas foram projetadas para 0.5 mm. Por último, foi gerada uma área de aterramento com a referência GND com a intenção de reduzir a interferência de ruídos eletromagnéticos e garantir a integridade do sinal transmitido pelas trilhas. O projeto final gerou uma PCB de simples face.

3.3.3 *Firmware* do microncontrolador

O desenvolvimento do *firmware* para o microcontrolador foi realizado na linguagem de programação C padrão da ESPRESSIF. Para isso, foi utilizado um conjunto de ferramentes baseadas no *framework* ESP-IDF juntamente com a plataforma de desenvolvimento de código aberto *PlatformIO* no editor de código *Visual Studio Code*.

3.3.3.1 Fluxograma geral das tarefas do *firmware*

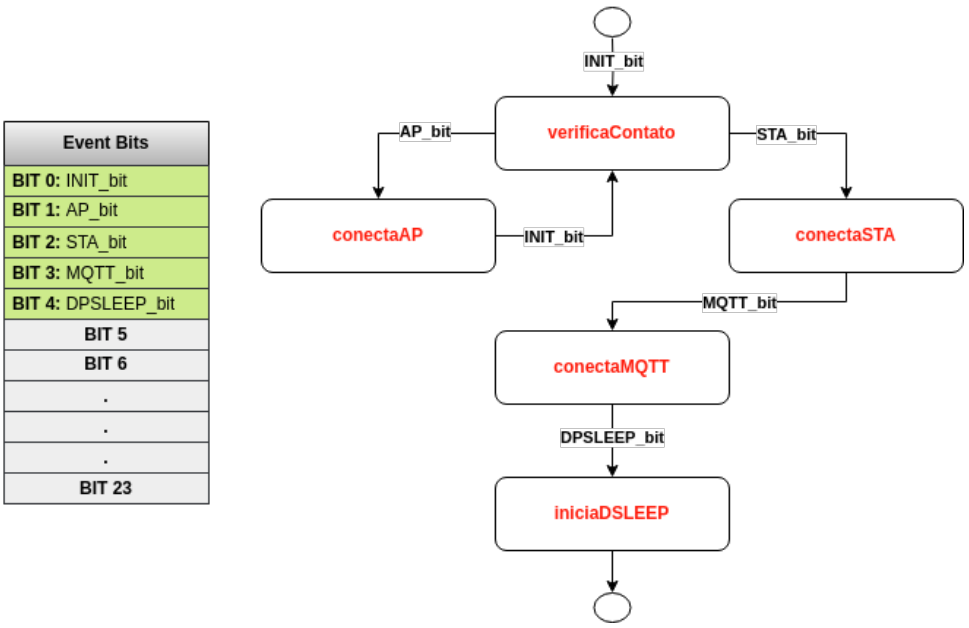
O *firmware* foi implementado utilizando o *freeRTOS* para definições de tarefas implementadas como *Event Groups*. A inicialização ocorre na função principal `MAIN`, onde as tarefas são definidas e o bit `INIT_Bit` é setado. Esse bit permite a inicialização da tarefa `verificaContato` a partir da qual será possível fazer verificações na memória não volátil (NVS) e sobre dados de data e hora, para então definir se o dispositivo irá iniciar o modo AP ou STA.

Caso o bit setado seja o `AP_Bit`, a próxima tarefa executada será a `conectaAP`, responsável por permitir ao usuário definir os dados de configurações iniciais (*setup*) do dispositivo e retornar para a `verificaContato` ao finalizar. Na situação em que o bit setado na `verificaContato` seja o bit `STA_Bit`, a próxima tarefa será a `conectaSTA`; essa é responsável por conectar o dispositivo na rede *Wi-Fi* de acordo com as informações definidas nas configurações iniciais, e atualizar a data e hora do dispositivo via protocolo NTP.

Ao finalizar a tarefa `conectaSTA`, será acionado o bit `MQTT_Bit` para que a tarefa `conectaMQTT` seja inciada. Essa tarefa permite a comunicação MQTT, publicando no tópico especificado pelo usuário, o estado do contator e a hora da aquisição desse estado em formato JSON. A última tarefa, `iniciaDSLEEP`, é chamada através do bit `DPSLEEP_Bit`, setado na tarefa anterior. A função da `iniciaDSLEEP` é realizar as definições de interrupção para saída do modo de sono profundo DSM, e iniciá-lo.

Os bits com a função de iniciar as tarefas implementadas como *Event Groups* voltam para nível lógico baixo ao finalizar a própria tarefa. O fluxograma das tarefas é apresentado na figura 15.

Figura 15 – Fluxograma Geral das Tarefas



Fonte: Elaborado pelo autor.

3.3.3.2 Fluxograma da função principal

A função principal tem como finalidade inicial definir as GPIO's a serem utilizadas:

- `REED SWITCH : GPIO_NUM_2`
- `PUSH BUTTON : GPIO_NUM_3`
- `LED1 : GPIO_NUM_9`
- `LED2 : GPIO_NUM_8`

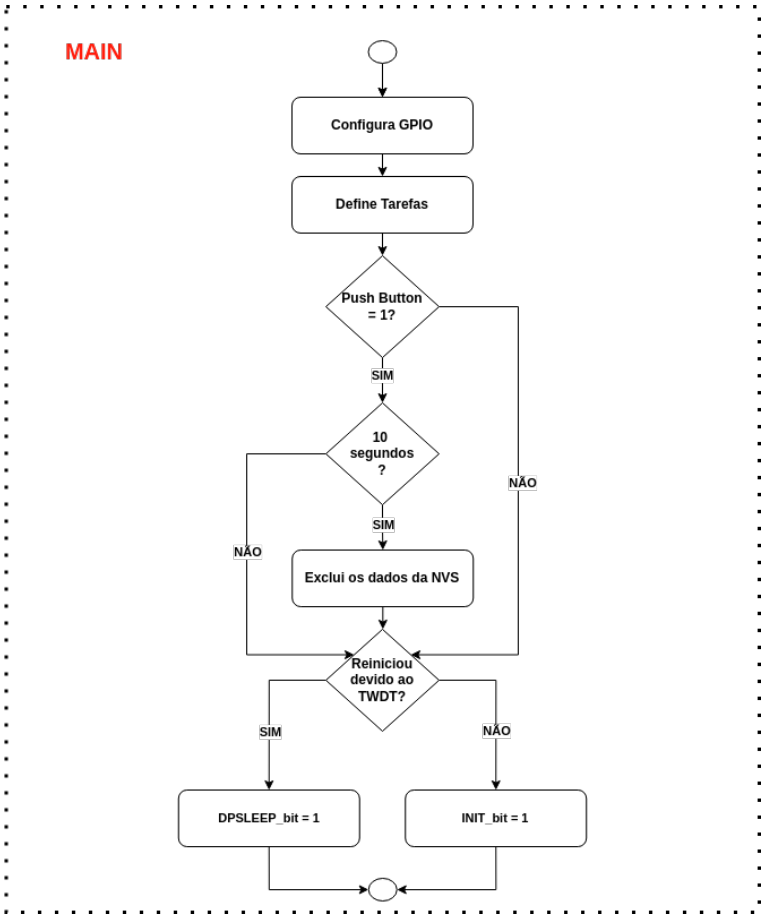
Apesar das portas utilizadas para o *Reed Switch*, *Push Button*, LED1 e LED2 serem, respectivamente, D0, D1, D9 e D8, para o *firmware* é necessário definir as portas relativas às suas identificações no SoC, que são `GPIO_NUM_2`, `GPIO_NUM_3`, `GPIO_NUM_9` e `GPIO_NUM_8`. O *Reed Switch* e o *Push Button* foram definidos como portas de entrada. Os LEDs foram definidos como saída. Dessa forma, o LED1 irá ligar ao energizar o dispositivo, indicando durante sua utilização, que o mesmo está energizado.

Após as definições de GPIO, são declaradas as tarefas como *Event Groups*. Essas tarefas irão aguardar um determinado bit para que sejam iniciadas, com exceção da tarefa

PiscaLED, responsável apenas por ascender e apagar o LED2 a cada 1 segundo, a qual é a única que será executada sem a necessidade de aguardar um bit.

Após declarar as tarefas, a função principal verifica se o *Push Button* está pressionado. Caso esteja, a terefa deverá aguardar no máximo 10 segundos, ou que o botão não esteja mais pressionado, para seguir em frente. Se o tempo pressionado for de 10 segundos, os próximos passos são: apagar os dados da memória não volátil e verificar se a razão da inicialização do ESP32-C3 foi devido a uma interrupção do TWDT (*Task WatchDog Timer*). Caso não seja pressionado ou o tempo pressionado seja inferior a 10 segundos, o programa segue sem apagar os dados da NVS. Após a verificação da razão de inicialização, caso o motivo tenha sido o TWDT, o dispositivo irá entrar em sono profundo; caso contrário, irá modificar o bit `INIT_bit` para 1, fazendo a tarefa `verificaContato` iniciar.

Figura 16 – Fluxograma da Função Principal *MAIN*



Fonte: Elaborado pelo autor.

3.3.3.3 Fluxograma da Tarefa verificaContato

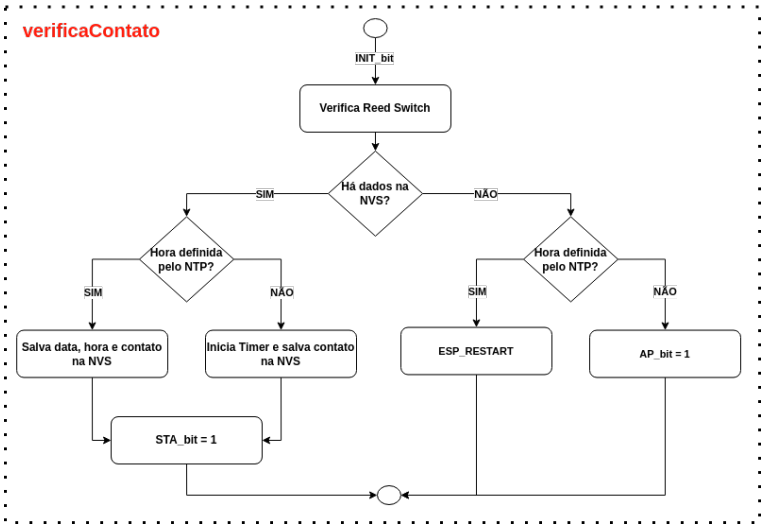
A primeira tarefa inicia quando o bit `INIT_bit` vai para nível lógico alto. Essa tarefa é responsável por verificar o estado atual do *Reed Switch* e definir quais os próximos passos que o sistema deverá tomar de acordo com algumas condições. A tomada de decisão depende de duas informações: se as configurações iniciais do dispositivo estão salvas na memória não volátil, e se há registros de atualização de data e hora no RTC. Caso seja verificado que há algum dado salvo em uma chave da NVS chamada SSID, indicando que a memória já foi inicializada com esse dado pelo usuário, o programa verifica se o RTC está atualizado. A confirmação de atualização do RTC é feita analisando se a variável chamada `temHoraSalva`, definida na própria memória do RTC, é igual a 1. Sendo verdadeiro os retornos das duas verificações, o programa salva o horário atual e o estado do *Reed Switch* na NVS, nas chaves `rtc_time` e `contato` respectivamente. Em seguida o bit `STA_bit` vai para nível lógico alto.

Caso não existam dados salvos no RTC, mas exista dado na chave SSID, o estado do *Reed Switch* é salvo na NVS e é capturado o tempo decorrido desde o início do programa até o momento atual, através da função `esp_timer_get_time()` salva na variável `tempoInicial`. Essa variável será utilizada na tarefa `conectaSTA` para estimar o tempo decorrido do momento da captura até o momento da aquisição da data e hora atualizada. Por último, o bit `STA_bit` vai para nível lógico alto.

Se não existirem dados salvos na chave SSID e nem no RTC, indicando que a NVS foi apagada ou ainda não foram feitas as configurações iniciais pelo usuário, o bit `AP_bit` vai para nível lógico alto.

Por último, caso não exista dado na chave SSID, mas existam dados de data e hora salvos no RTC, o ESP32 irá reiniciar. Essa é uma situação improvável, a qual pode indicar algum problema na leitura ou escrita dos dados.

Figura 17 – Fluxograma da Tarefa verificaContato



Fonte: Elaborado pelo autor.

3.3.3.4 Fluxograma da Tarefa conectaAP

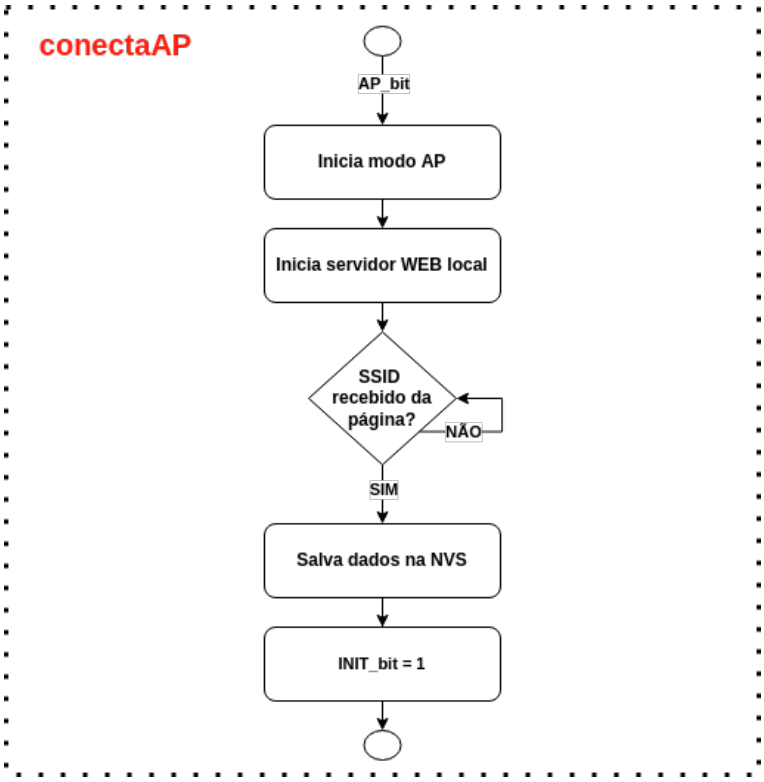
Caso o bit `AP_bit` vá para nível lógico alto, a tarefa `conectaAP` é iniciada. O primeiro passo dessa tarefa é iniciar o ESP32 no modo AP através da função `wifi_init_softap()`. Além disso, são iniciados os manipuladores dos eventos relacionados à conexão *wireless* do dispositivo. Após isso, o servidor WEB é iniciado, e o programa aguarda a confirmação do envio dos dados de configuração do dispositivo, pelo usuário. Para isso, o usuário deve se conectar na rede Wi-Fi gerada pelo ESP32, denominada "SmartContact", sendo a senha igual a "12345678"; em seguida, o mesmo deverá acessar à página de configuração com o seguinte endereço: http://192.168.4.1.

Uma vez enviados os dados, esses são salvos na memória não volátil nas respectivas chaves:

- Nome da rede Wi-Fi = "ssid";
- Senha da rede Wi-Fi = "password";
- *Broker* MQTT= "broker";
- Tópico MQTT = "topic";
- Tempo máximo DSM = "maxtime";

Após salvar os dados, o programa retorna para a tarefa `verificaContato` mudando o bit `INIT_bit` para nível lógico alto.

Figura 18 – Fluxograma da Tarefa conectaAP



Fonte: Elaborado pelo autor.

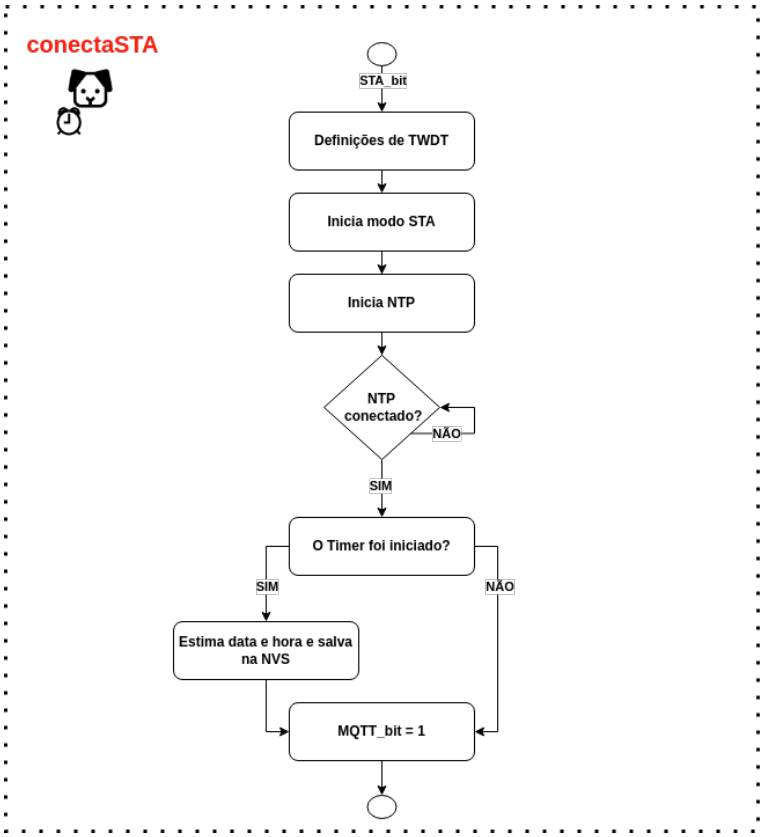
3.3.3.5 Fluxograma da Tarefa conectaSTA

Caso o bit `STA_bit` vá para nível lógico alto, a tarefa `conectaSTA` inicia. O TWDT é definido para 60 segundos e iniciado em seguida. Dessa forma, caso a tarefa demore mais que 60 segundos para finalizar, o ESP32 irá reiniciar como medida de segurança para evitar a tentativa de conexão em momentos que houver falha na rede ou no servidor NTP.

Após a definição do TWDT, o Wi-Fi é iniciado com a função `wifi_connection()`, conectando na rede informada pelo usuário. Em seguida, a comunicação com o servidor NTP é feita através da função `esp_sntp_init()`. O ESP32 aguarda a sincronização de data e hora, e quando a mesma é finalizada, a variável `temHoraSalva` recebe o valor 1.

Como dito anteriormente, para os casos em que exista dado de SSID na NVS mas o RTC não tenha data e hora salvas, o tempo da captura do estado do *Reed Switch* é estimado pela função `esp_timer_get_time()`. Logo, caso a variável `tempoInicial` possua um valor salvo, esse valor é subtraído pelo valor de tempo atual capturado pela mesma função (`esp_timer_get_time()`), e essa diferença é utilizada para estimar o momento exato da captura do estado do *Reed Switch*, salvando o valor estimado na NVS. A finalização dessa tarefa ocorre alterando o bit `MQTT_bit` para nível lógico alto.

Figura 19 – Fluxograma da Tarefa conectaSTA

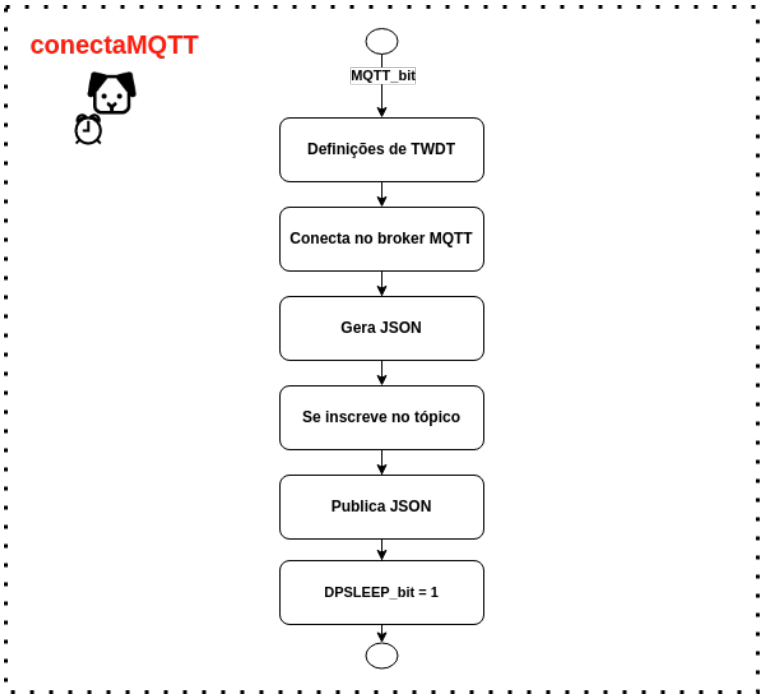


Fonte: Elaborado pelo autor.

3.3.3.6 Fluxograma da Tarefa conectaMQTT

A tarefa `conectaMQTT` inicia quando o bit `MQTT_bit` vai para nível alto. As definições de TWDT são feitas seguindo a mesma ideia e parâmetros feitos na tarefa `conectaSTA`. As funções dessa tarefa são: realizar a conexão do dispositivo com o *broker* MQTT definido pelo usuário; gerar arquivo formatado em JSON contendo estado do *Reed Switch* e data da aquisição do estado; se inscrever no tópico passado também pelo usuário; publicar JSON gerado, no tópico em questão, e por último, mudar o estado do bit `DPSLEEP_bit` para 1.

Figura 20 – Fluxograma da Tarefa conectaMQTT



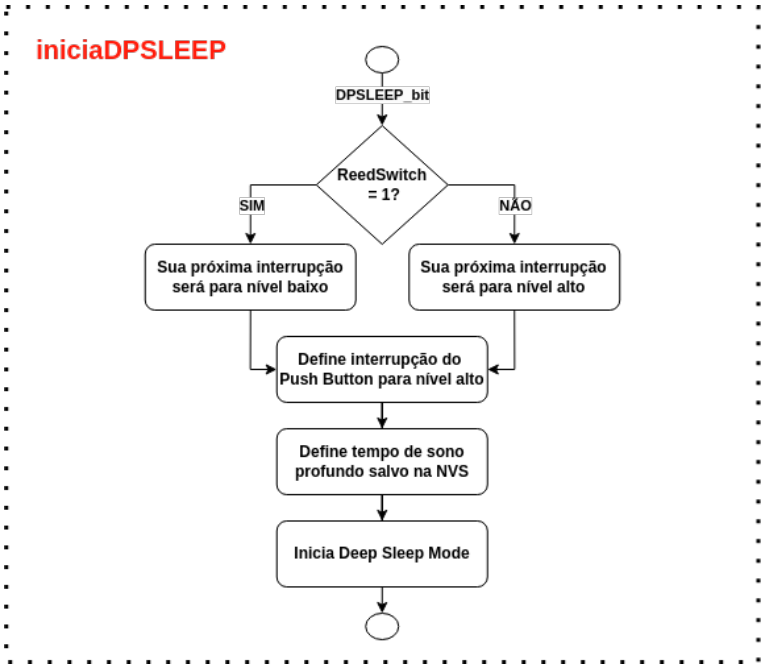
Fonte: Elaborado pelo autor.

3.3.3.7 Fluxograma da Tarefa iniciaDPSLEEP

A última tarefa é responsável por definir as interrupções que farão o ESP32 retornar do modo de sono profundo, e em seguida iniciar esse modo. No entanto, como uma das interrupções é retornar do DSM quando houver variação no estado do *Reed Switch*, a definição dessa interrupção irá depender do estado mais atual do sensor. Logo, se o *Reed Switch* estiver em nível lógico alto, a interrupção deverá ser definida para quando o sensor estiver em nível lógico baixo, e o oposto caso contrário. Com relação ao *Push Button*, a interrupção ocorrerá sempre que o mesmo estiver em nível alto.

Além dessas interrupções, o dispositivo também deverá retornar do sono profundo quando o tempo máximo, definido pelo usuário, for atingido. Caracterizando uma mensagem de *Keep-Alive*. Após essas definições, o dispositivo inicia o modo de economia de energia.

Figura 21 – Fluxograma da Tarefa iniciaDPSLEEP



Fonte: Elaborado pelo autor.

3.4 Projeto Mecânico

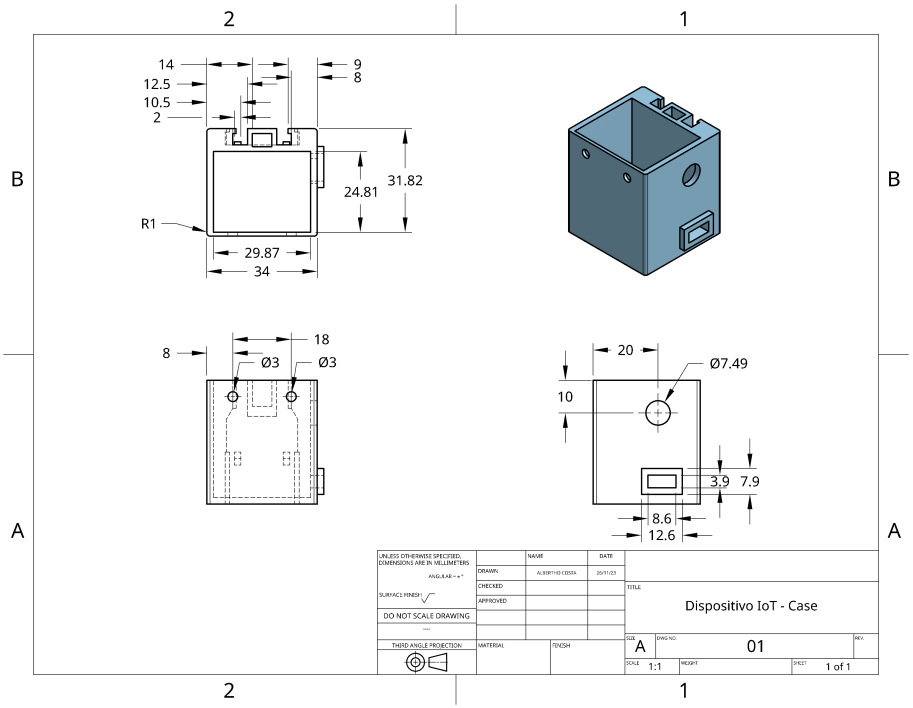
O projeto mecânico foi desenvolvido com a intenção de permitir o encaixe frontal em um contator tripolar da marca *Schneider* modelo LC1E2510. Devido à compatibilidade de encaixe entre esse contator com o bloco de contato auxiliar frontal LADN11, esse bloco foi utilizado como referência para a elaboração do projeto mecânico.

Figura 22 – Bloco de Contato Auxiliar frontal Tesys LADN11

Fonte: Acesso em 02-12-2023. Disponível em: <https://www.newark.com/pt-BR/schneider-electric/ladn11/contactor-auxiliary-contact/dp/31C9311>

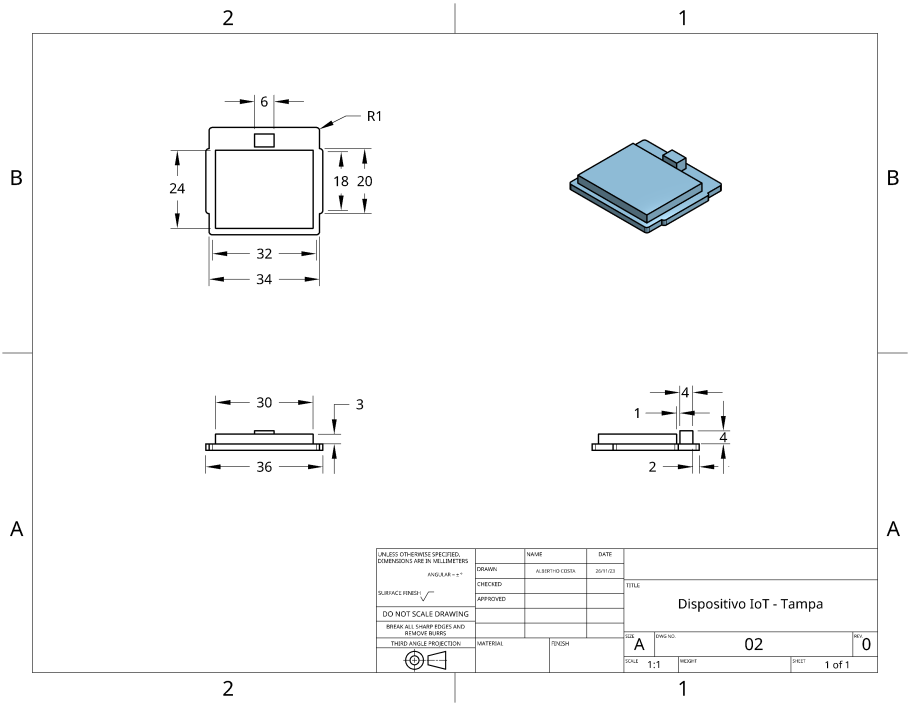
A partir da análise dimensional entre o bloco auxiliar e o contator, foram projetadas a *case*, que comporta os componentes do dispositivo, e uma tampa para a *case*. As Figuras 23 e 24 apresentam os desenhos técnicos desenvolvidos.

Figura 23 – Desenho técnico da *case* do dispositivo IoT



Fonte: Elaborado pelo autor.

Figura 24 – Desenho técnico da tampa da *case*



Fonte: Elaborado pelo autor.

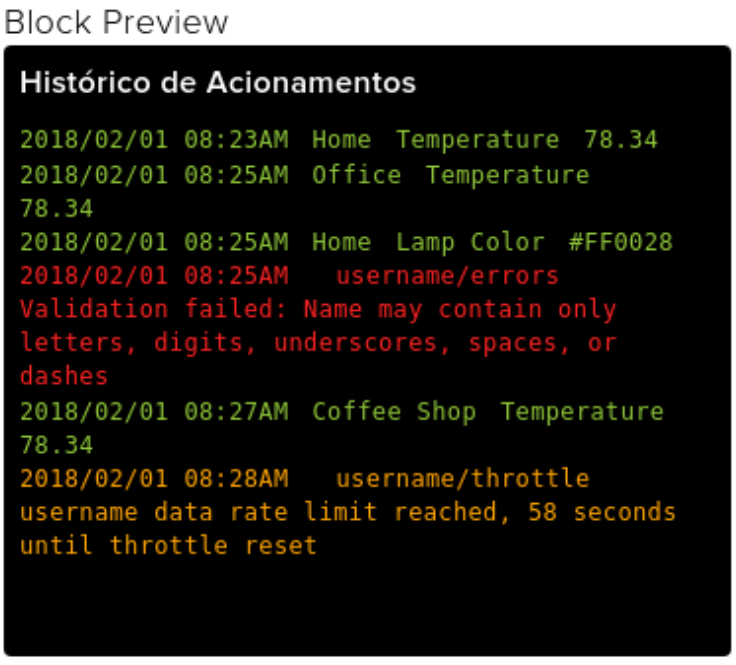
3.5 Plataforma IoT

A plataforma AdafruitIO, fornecida pela Adafruit *Industries*, tem se destacado como uma ferramenta acessível para a implementação de soluções IoT. Ela oferece uma gama de recursos de gerenciamento de dispositivos IoT que simplificam o desenvolvimento de projetos IoT, facilitando a integração dos dados dos dispositivos com aplicações em nuvem. No âmbito deste trabalho, a plataforma AdafruitIO foi escolhida devido ao *broker* MQTT e seu serviço de *dashboards* para monitoramento dos dados.

Neste projeto, foi criado um *Feed* na plataforma AdafruitIO denominado "contator". Esse *Feed* atua como um repositório para os dados enviados pelo dispositivo IoT. Para facilitar a interpretação e análise dos dados, foi desenvolvido um *dashboard* na plataforma, intitulado "Dispositivo IoT para Monitoramento em Tempo Real de Contator".

No *dashboard* criado, foi acrescentado o bloco do tipo *Stream*, o qual é capaz de permitir a visualização de uma mensagem em formato JSON. Após a inclusão do bloco, é necessário vincular com o *Feed* "contator"criado anteriormente. Esse painel proporciona uma visão intuitiva do estado do contator, permitindo a observação instantânea de quaisquer alterações.

Figura 25 – Visualização prévia do bloco *Stream* na plataforma Adafruit IO.



Fonte: Elaborado pelo autor.

Por fim, foram realizados os projetos eletrônico e mecânico do dispositivo IoT SmartContact, incluindo a programação do SoC ESP32-C3, com o objetivo de publicar o

dado do contator (estado) em alguma aplicação com broker MQTT através da Internet. A plataforma Adafruit IO foi utilizada para validar o projeto do dispositivo e monitorar os dados recebidos. Os resultados são apresentados no próximo capítulo.

4 Resultados

Este capítulo apresenta os resultados do projeto do dispositivo IoT SmartContact. O dispositivo foi projetado com o SoC ESP32-C3 para permitir a conexão via protocolo Wi-Fi com alguma rede do ambiente instalado, a fim de publicar o dado do estado do contator em algum broker MQTT.

O protótipo do dispositivo SmartContact foi fabricado, montado e instalado em um contator de uma bancada de teste do Laboratório de Conectividade e Soluções Industriais (ConecteLab) da UFRN. Além disso, foram realizadas as etapas de configuração inicial (*setup*) e monitoramento dos dados via *dashboard* na plataforma Adafruit IO.

4.1 Montagem e Instalação

O contator utilizado para a aplicação em questão se trata do contator tripolar Tesys E, do fabricante Schneider, instalado em uma bancada de teste do laboratório ConecteLab. Um pequeno imã posicionado na região central da parte móvel externa foi instalado no contator. A figura 26 apresenta o contator com o imã.

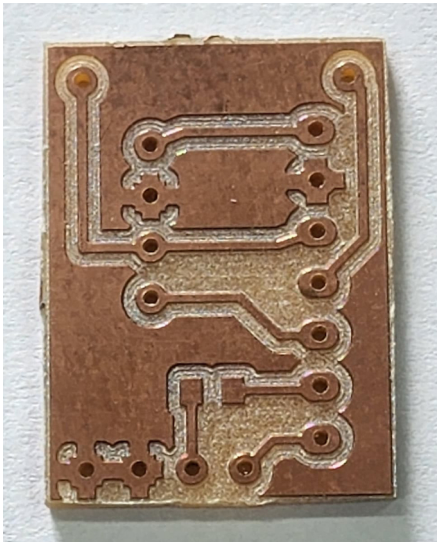
Figura 26 – Contator Schneider Tesys E com imã.

Fonte: Elaborado pelo autor.

A primeira etapa de fabricação do protótipo do dispositivo foi a confecção da placa de circuito impresso. Os arquivos gerbers do projeto do layout da PCB, apresentado na

figura 14, foram gerados a partir da EasyEDA e enviados para fabricação. A figura 27 apresenta a PCB fabricada.

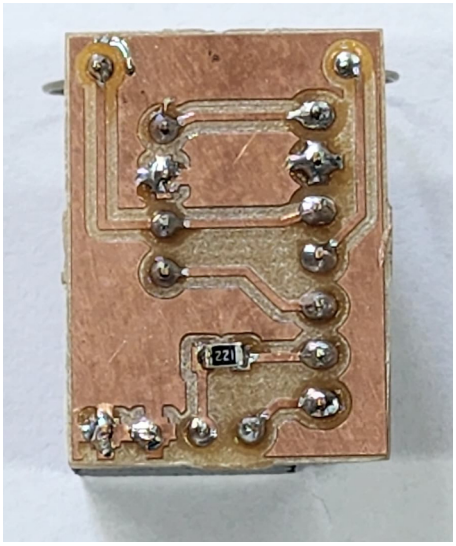
Figura 27 – Lado inferior da PCB antes da soldagem.



Fonte: Elaborado pelo autor.

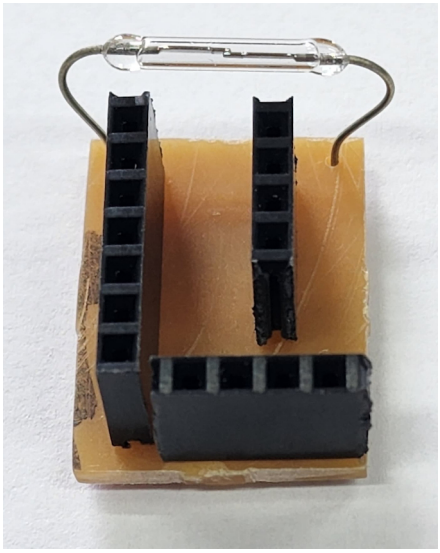
Após a confecção, foram soldados os componentes em seus respectivos locais. As figuras 28 e 29 apresentam as vistas inferior e superior, respectivamente, da placa com os componentes soldados.

Figura 28 – Lado inferior da PCB.



Fonte: Elaborado pelo autor.

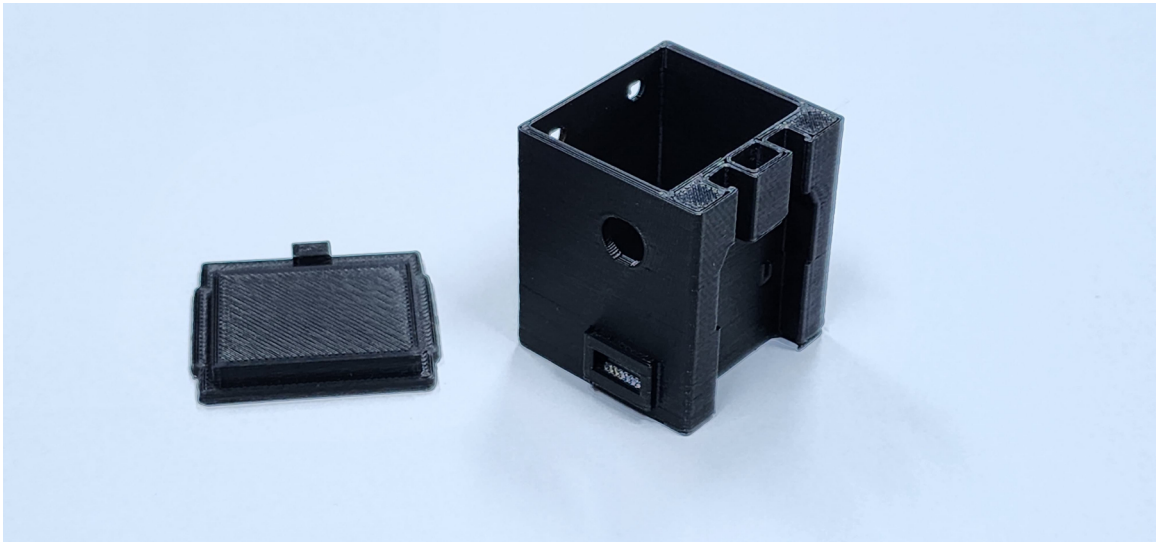
Figura 29 – Lado superior da PCB.



Fonte: Elaborado pelo autor.

A partir dos projetos mecânicos da *case* e da tampa do dispositivo, apresentados nas figuras 23 e 24, foram gerados os arquivos STL para enviar para fabricação das peças em 3D. Vale salientar que essa peças foram fabricadas apenas para validar os testes do protótipo do SmartContact. Para aplicações comerciais, o material a ser utilizado na fabricação das peças devem ser considerados de acordo com as Normas Brasileiras ou NBR para fabricação de dispositivos contatores e blocos auxiliares. A figura 30 apresenta as peças impressas em 3D.

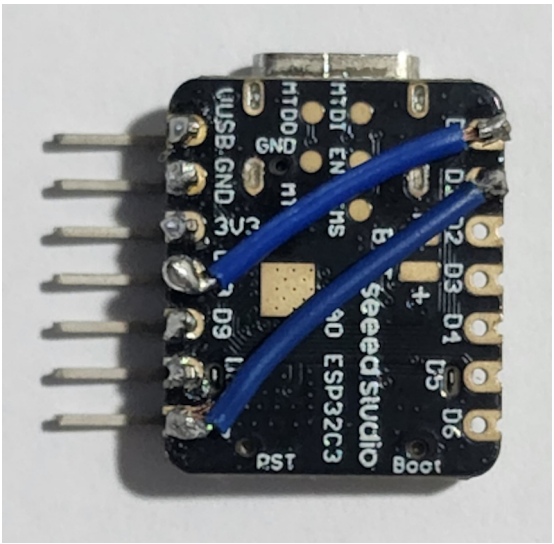
Figura 30 – Case e tampa do dispositivo impressas em 3D.



Fonte: Elaborado pelo autor.

O DevKit XIAO ESP32C3 foi montado na PCB lateralmente. As portas D0 (GPIO2) e D1 (GPIO3) foram utilizadas para a leitura das interrupções externas dos, respectivamente, *Reed Switch* e *Push Button* por se tratarem de RTC GPIO. Dessa forma, como o DevKit foi montado lateralmente, foi realizado dois pequenos *jumper* em ambas as GPIO para as portas D10 (GPIO10) e D7(GPIO20).

Figura 31 – XIAO ESP32C3 com as conexões (*Jumpers*).



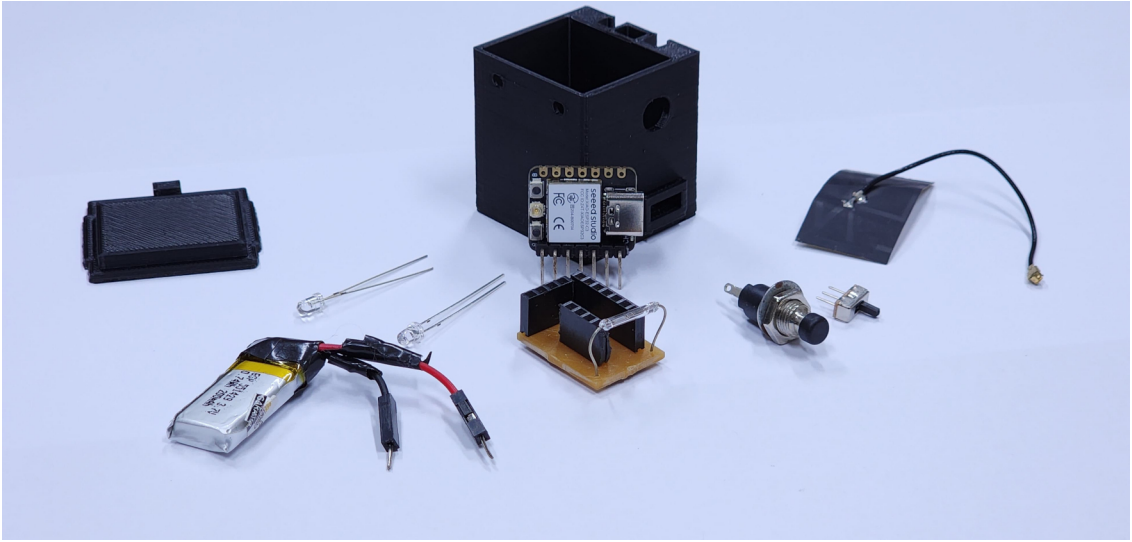
Fonte: Elaborado pelo autor.

De acordo com o requisito RF12, o dispositivo deve ser alimentado através de uma

bateria. Logo, foi utilizada no projeto uma bateria recarregável de 3,7V e 200mAh.

A figura 32 apresenta todos os componentes do dispositivo SmartContact, antes da realização da montagem.

Figura 32 – Componentes do dispositivo SmartContact.



Fonte: Elaborado pelo autor.

Com todos os componentes prontos, foi realizada a montagem e a conexão do dispositivo no contator.

Figura 33 – Dispositivo SmartContact instalado.

Fonte: Elaborado pelo autor.

4.2 Configurações de *Setup*

Como dito anteriormente, o requisito RF04 indica que o dispositivo deve fornecer ao usuário um meio de configuração para as definições iniciais, ou *Setup*. Essa implementação é realizada através de uma página WEB construída em HTML.

Veja na figura 34 a página gerada para o usuário.

Figura 34 – Página WEB para configurações de *Setup*.

Fonte: Elaborado pelo autor.

4.3 Leitura dos dados no Adafruit IO

Para a utilização do *dashboard* criado na plataforma Adafruit IO, é preciso definir o endereço do *broker* e o tópico de acordo com a seguinte formatação:

- *Broker*: `"mqtt://username:active_key@io.adafruit.com"`
- *Tópico*: `"username/feeds/feed_key"`

Ao substituir os termos *username*, *active_key* e *feed_key* pelos valores encontrados nas configurações da própria conta na plataforma, a comunicação entre dispositivo e o Adafruit IO pode ser realizada.

É interessante lembrar que o dispositivo SmartContact foi projetado para publicar em qualquer broker MQTT. No entanto, com o objetivo de validar, os dados foram publicados no broker da Adafruit IO.

Figura 35 – Leitura dos Dados na Adafruit IO

Fonte: Elaborado pelo autor.

A figura 35 mostra um exemplo do recebimento dos dados na plataforma, com o arquivo JSON formatado para enviar os dados do *status* (Json_Status) e da data (Json_RTC). No teste realizado na figura em questão, o dispositivo foi configurado para permanecer no modo de economia de energia por 20 segundos, a não ser que exista uma mudança de estado (Ligado para Desligado ou o contrário). Note que existe uma pequena variação em segundos, do tempo capturado entre os envios, desconsiderando a interrupção pela mudança de estado do contator. Essa diferença pode estar ocorrendo devido às atualizações recorrentes do horário pelo servidor NTP.

5 Conclusões

Neste trabalho foi desenvolvido um dispositivo IoT denominado SmartContact capaz de adaptar um componente crucial em sistemas industriais, o contator eletromecânico. Essa adaptação se refere a torná-lo um disposito conectado, sendo possível monitorá-lo em tempo real, através do protocolo de comunicação MQTT. O dispositivo desenvolvido detecta as variações de acionamentos provenientes do contator através de um sensor magnético *Reed Switch*. Os dados referentes ao estado do contator (podendo esse ser Ligado ou Desligado) e o horário da aquisição do estado são modificados para uma formatação JSON, e enviados para a plataforma IoT AdafruitIO.

O modelo mecânico desenvolvido para ser aclopado frontalmente ao contator se mostrou muito eficaz, realizando um encaixe seguro e sem folgas. O mesmo foi fabricado através de manufatura aditiva (impressão 3D), o que mostrou ser uma forma de fabricação precisa e eficiente para essa aplicação.

O sensor *Reed Switch*, utilizado para capturar o estado do contator, se mostrou eficaz na maioria das interrupções geradas pela mudança no estado do contator. Devido a necessidade de inserir um pequeno imã na parte móvel externa do contator, a escolha desse imã pode influenciar diretamente na leitura do sensor, podendo esse, não detectar a variação do campo magnético em certos momentos.

O DevKit XIAO ESP32C3 utilizado se mostrou eficiente para o *firmware* implementado. Mesmo com a utilização de *free*RTOS, as tarefas desenvolvidas não proporcionaram problemas de concorrências de processamento, devido à lógica sequencial em que foram feitas. Dessa forma, um único *core* de processador foi o suficiente, tornando o SoC ESP32-C3 ideal para esse projeto.

Em um futuro projeto, alguns acréscimos e modificações podem vir a serem feitos, como: o envio do *status* da bateria para o *broker* MQTT; uma verificação mais detalhada nos dados enviados pelo usuário, ao realizar as configurações iniciais (*setup*); mais ações tomadas pelo *hardware* para a prevenção de falhas de conectividade e de processamento; detectar a conexão entre dispositivo e contator; a fabricação de uma PCB mais adequada para o projeto, a qual possa suportar mais GPIO do DevKit utilizado; coletar mais dados na plataforma IoT para validar a eficiência nos envios dos dados; armazenar os dados capturados para as situações em que ocorrer problemas na comunicação entre SmartContact e *broker* MQTT; por último, avaliar uma outra forma de identificar o estado do contator, podendo ser um outro tipo de sensor, ou até mesmo a fabricação de uma peça móvel que capture esse estado de forma mecânica.

Além dos acrécimos a serem considerados em pespectivas futuras, para uma possível

validação do dispositivo SmartContact no setor comercial, o seu desenvolvimento deve ser baseado nas normas que tratam as definições dos requisitos técnicos sobre contatores e blocos auxiliares, como a IEC 60947-4-1.

Referências

BABAYIGIT, B.; ABUBAKER, M. Industrial internet of things: A review of improvements over traditional scada systems for industrial automation. *IEEE Systems Journal*, IEEE, 2023. Citado na página 13.

BARROS, E.; CAVALCANTE, S. Introdução aos sistemas embarcados. *Artigo apresentado na Universidade Federal de Pernambuco-UFPE*, p. 36, 2010. Citado na página 19.

BENEDITO, J. et al. *Indústria 4.0: conceitos e fundamentos*. [S.l.]: Edgard Blücher Ltda, 2018. Citado na página 22.

ELECTRIC, S. *Principais Componentes em Comandos Elétricos*. 2021. Acessado em 18 de novembro de 2023. Disponível em: <https://www.se.com/ww/en/about-us/company-profile/history/telemecanique.jsp#:~:text=Telemecanique%20invented%20the%20first%20contactor,for%20industrial%20control%20and%20automation.> Citado na página 17.

ESPRESSIF. *SoC ESP32C3 ESPRESSIF*. 2023. Acessado em 24 de novembro de 2023. Disponível em: <https://www.espressif.com/en/products/socs/esp32-c3>. Citado 2 vezes nas páginas 35 e 36.

FREERTOS.ORG. *Event Bits (or flags) and Event Groups*. 2023. Acessado em 02 de novembro de 2023. Disponível em: <https://www.freertos.org/FreeRTOS-Event-Groups.html#API>. Citado na página 22.

HOJAIJ, D. *Principais Componentes em Comandos Elétricos*. 2023. Acessado em 21 de novembro de 2023. Disponível em: <https://blog.se.com/br/eletricistas/2021/06/14/principais-componentes-em-comandos-eletricos/>. Citado 2 vezes nas páginas 16 e 17.

LIMA, F. R.; GOMES, R. Conceitos e tecnologias da indústria 4.0: uma análise bibliométrica. *Revista Brasileira de Inovação*, v. 19, 2021. Citado na página 13.

MIRANDA BRUNO DOURADO, R. S. d. O.; CARMINATI., A. Analysis of freertos overheads on periodic tasks. *Anais do XX Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, 2021. Citado na página 21.

MQTT.ORG. *MQTT: The Standard for IoT Messaging*. 2022. Acessado em 20 de novembro de 2023. Disponível em: <https://mqtt.org/>. Citado na página 23.

SANTOS, G. *Contator: O Que É, Como Funciona e Tipos*. 2022. Acessado em 18 de novembro de 2023. Disponível em: <https://www.automacaoindustrial.info/contator/>. Citado na página 17.

SCHWAB, K. *The Fourth Industrial Revolution*. [S.l.]: "Crown Business", 2017. Citado 2 vezes nas páginas 13 e 18.

SONI, D.; MAKWANA, A. A survey on mqtt: a protocol of internet of things (iot). *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, v. 20, 2017. Citado na página 23.

STUDIO, S. *XIAO ESP32C3 Getting Started*. 2022. Acessado em 24 de novembro de 2023. Disponível em: <https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/>. Citado na página 33.

VOGEL BAHTIJAR, e. a. What is an open iot platform? insights from a systematic mapping study. *Future Internet*, v. 12, n. 4, p. 73, 2020. Citado na página 25.

ZURITA, M. E. P. V. Projeto de sistemas embarcados. *Artigo apresentado na Universidade Federal do Piauı, Curso de Engenharia Elétrica, Campus Universitário Ministro Petrônio Portela*, p. 9, 2011. Citado 2 vezes nas páginas 19 e 20.