UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ANDERSON LUIZ DE SOUZA KMETIUK

GERENCIAMENTO DE SISTEMA CFTV CONTRA FALHAS DE FORMA AUTÔNOMA OU REMOTA

CURITIBA

2024

ANDERSON LUIZ DE SOUZA KMETIUK

GERENCIAMENTO DE SISTEMA CFTV CONTRA FALHAS DE FORMA AUTÔNOMA OU REMOTA

AUTONOMOUS OR REMOTE CCTV SYSTEM FAILURE MANAGEMENT

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador (a): Paulo Denis Garcez da Luz

CURITIBA

2024

4.0 Internacional

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

ANDERSON LUIZ DE SOUZA KMETIUK

GERENCIAMENTO DE SISTEMA CFTV CONTRA FALHAS DE FORMA AUTÔNOMA OU REMOTA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 12/Setembro/2024

Paulo Denis Garcez da Luz
Doutorado
Universidade Tecnológica Federal do Paraná

Eduardo Nunes dos Santos
Doutorado
Universidade Tecnológica Federal do Paraná

Luiz Fernando Copetti
Mestrado
Universidade Tecnológica Federal do Paraná

CURITIBA

2024

Dedico este trabalho a Deus, que me presenteia todos os dias com energia e vida, que me dá forças e coragem para atingir os meus objetivos e que é merecedor de toda honra e glória.

AGRADECIMENTOS

A realização deste trabalho foi possível graças a muito esforço e dedicação, além do apoio e colaboração de várias pessoas, a quem expresso minha sincera gratidão.

Primeiramente, agradeço a Deus por me conceder saúde, força, sabedoria e garra para superar os desafios ao longo desta jornada acadêmica.

Agradeço ao Professor Dr. Paulo Denis Garcez da Luz, meu orientador, por sua incansável dedicação, paciência, orientação e apoio constante ao longo de todo o processo do trabalho de conclusão de curso.

Agradeço imensamente à minha família, especialmente aos meus pais, pelo amor, paciência e incentivo incondicional. Aos meus amigos, por sempre estarem presentes para me apoiar nos momentos em que pensei em desistir.

Aos professores do Departamento de Engenharia Eletrônica, cujas aulas foram fundamentais para a realização deste trabalho, e aos membros da banca examinadora por suas valiosas contribuições.

A todos que, de alguma forma, contribuíram para a realização deste trabalho, meu muito obrigado.

Não é o que você tem, ou quem você é, ou onde você está, ou o que você está fazendo que o deixa feliz ou infeliz. É o que você pensa sobre. (CARNEGIE, Dale; *How to Win Friends and Influence People*, 1936).

RESUMO

A proposta desse trabalho é desenvolver um protótipo que será acoplado a um sistema de Circuito Fechado de Televisão (CFTV) que realizará o gerenciamento contra falhas de forma autônoma ou remota. Dentre essas falhas do sistema existem problemas no *nobreak*, no *switch* de rede, nas câmeras, a falta de *internet*, entre outras. Assim, o equipamento fará o monitoramento de todos os equipamentos a fim de detectar algum travamento. Se isto acontecer o sistema deve reiniciar o equipamento a fim de destravá-lo. O objetivo principal a ser atingido é a economia de tempo e dinheiro nas resoluções de problemas técnicos, principalmente relacionado ao deslocamento de equipes ao local da falha. Esse projeto será destinado a empresas, prédios residenciais e comerciais.

Palavras-chave: CFTV; circuito fechado de TV; gerenciamento autônomo; ESP32; Orange Pi; acesso remoto; Linux.

ABSTRACT

The main goal of this thesis is to develop a prototype that will be embedded into a Closed Circuit Television (CCTV) system and manage it against failure autonomously or remotely. Among these system failures, we may encounter issues with the uninterruptible power supply (UPS), network switch, cameras, lack of internet, among others. Therefore, the equipment will monitor all the devices to detect any freezes.If this occur, the system should reboot the device and unlock it. The main objective to be achieved is to save time and money solving technical issues, especially related to the dispatch of teams to the site of the failure. This project will be targeted at companies, residential and commercial buildings.

Keywords: *CCTV; Closed Circuit Television; autonomous management; ESP32; Orange Pi; remote access; Linux.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BLE	*Bluetooth Low Energy*
CC	Corrente Contínua
CFTV	Circuito Fechado de Televisão
COVID-19	*Coronavirus Disease of 2019*
CPU	*Central Processing Unit*
GPIO	*General Purpose Input Output*
GND	*Ground*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
LAN	*Local Area Network*
LED	*Light Emitting Diode*
MAC	*Media Access Control*
OSI	*Open Systems Interconnection*
OTA	*Over-The-Air*
PCI	Placa de Circuito Impresso
PoR	*Power on Reset*
REST	*Representational State Transfer*
SBC	*Single Board Computer*
SMD	*Surface Mount Device*
SoC	*System on Chip*
TBJ	Transistor Bipolar de Junção
TCP	*Transmission Control Protocol*
UART	*Universal Asynchronous Receiver / Transmitter*
URL	*Uniform Resource Locator*
WAN	*Wide Area Network*
Wi-Fi	*Wireless Fidelity*

LISTA DE SÍMBOLOS

μ	Micro
k	Kilo
Ω	Ohm

SUMÁRIO

1 INTRODUÇÃO

Este projeto visa a criação de um sistema de monitoramento de circuito fechado de televisão (CFTV). Para isso, será desenvolvido um protótipo que será acoplado ao sistema de câmeras, ao sistema de rede, que inclui *modems*, *switches* e *routers* de rede, e ao sistema de energia, onde está situado o *nobreak*. Este protótipo realizará o monitoramento e gerenciamento do sistema de forma autônoma ou remota.

Figura 1 - Componentes de um sistema de CFTV

Fonte: Autoria própria (2024)

A Figura 1 apresenta a representação de todos os periféricos desse sistema de CFTV. Assim, o número 1 corresponde ao sistema de câmeras, o número 2 ao *router*, o número 3 ao *hub* de rede, o número 4 ao *switch* de rede, o número 5 ao *nobreak* e, por fim, o número 6 ao *modem.*

1.1 Contextualização do tema

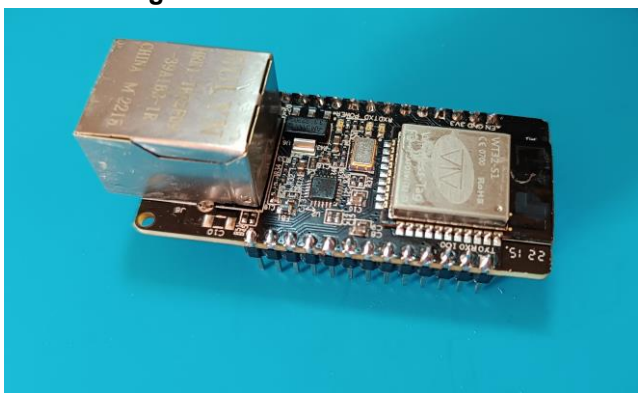
Para a realização do controle das rotinas e monitoramento do sistema CFTV, é necessário o desenvolvimento de um *hardware* que possua a integração com um banco de dados MySQL, a comunicação com um servidor para a geração de um *dashboard* e o envio de comandos e o desenvolvimento de um *firmware* com os acionamentos necessários para o controle autônomo ou remoto do sistema.

Para isso, serão desenvolvidas duas placas de circuito impresso (PCI), a primeira versão com um Arduino Nano (ARDUINO, 2024) acoplado a um módulo de *ethernet* ENC28J60 e a segunda versão utilizando o módulo WT32-ETH01 (WIRELESS-TAG TECHNOLOGY CO. LTD, 2019). Este módulo é derivado do kit ESP32-DevKitC e utiliza seu core ESP32-WROOM-32D acoplado a um módulo de *ethernet* LAN8720 de forma integrada.

A escolha do módulo WT32-ETH01 foi devido à necessidade de um conector RJ45 para a conexão de *internet* via cabo, visto que o sistema possui dois *modems* e um dos objetivos a serem atingidos desse projeto é a conexão simultânea *Wi-Fi* e *ethernet* para o monitoramento simultâneo desses *modems*. Essa conexão simultânea só poderá ser estabelecida na PCI com o módulo WT32-ETH01, pois o Arduino não é compatível com a conexão Wi-Fi.

Este projeto foi idealizado em parceria com a empresa *Installdat*, que foi responsável por fornecer o banco de dados, as rotinas *Linux* da *Orange Pi* e o *site* com um *dashboard*.

Figura 2 - Módulo WT32-ETH01



Fonte: Autoria própria (2024)

1.2 Principais funcionalidades

1.2.1 Monitoramento autônomo

O sistema é capaz de realizar o monitoramento contínuo de forma autônoma dos dispositivos conectados. Para isso, serão utilizados sensores e módulos integrados para detectar variações ou falhas nos dispositivos monitorados e serão realizados acionamentos de relés, retirando o fornecimento de energia da

alimentação de 110V/220V com o propósito de efetuar um *reset* forçado de cada dispositivo monitorado, de forma automática para corrigir os problemas.

Além disso, o sistema realizará a constante comunicação com o banco de dados da aplicação, para inserir dados em tempo real em um *dashboard* presente no *site* da aplicação remota, que mostrará as principais estatísticas do sistema e falhas ocorridas. Existe também uma área de comandos remotos para a execução de destravamentos forçados pelo usuário.

1.2.2 Monitoramento remoto

A funcionalidade do monitoramento remoto permite que os usuários acompanhem as principais estatísticas, em tempo real, dos dispositivos conectados ao sistema através de uma interface *web* amigável e intuitiva. Além disso, é possível enviar comandos por essa interface, possibilitando o controle do sistema à distância.

1.2.3 Comunicação WT32-ETH01 – *Orange Pi*

A comunicação do módulo WT32-ETH01 e o *Orange Pi,* que atua como um *Single Board Computer* (SBC), é essencial para o acesso em tempo real do sistema pelo usuário. O módulo WT32-ETH01 é responsável pelo controle dos relés, leitura dos sensores e comunicação com a interface *web.*

Já o *Orange Pi*, possui um Linux embarcado que possibilita o acesso remoto, assim é possível realizar atualizações de *firmware* do WT32-ETH01 remotamente e rotinas de gerenciamento do sistema.

A escolha de um *Orange Pi* foi devido ao fato deste módulo ser bem mais compacto que um *Raspberry Pi* e também ter um custo menor.

1.2.4 Atualização *Over-The-Air* (OTA)

A funcionalidade de atualização Over-The-Air (OTA) permite a atualização remota do *firmware* do módulo WT32-ETH01 utilizando a conexão *Wi-Fi,* LAN *ou ethernet*. Essa funcionalidade é extremamente útil em ambientes onde o acesso físico ao *hardware* é difícil, permitindo a manutenção e implementação de melhorias sem interrupções do serviço.

1.2.5 Conexão simultânea *Wi-Fi – Ethernet*

A conexão simultânea de *Wi-Fi e ethernet* possibilita a conexão do módulo a dois modems de operadoras de internet diferentes, um via cabo e outro via *Wi-Fi*, assim tendo redundância de acesso à internet. Essa configuração oferece uma maior confiabilidade ao sistema, pois possibilita que ele continue em funcionamento mesmo se houver falha em um dos modems. Assim é possível identificar com facilidade se algum deles estiver sem conexão e realizar o gerenciamento para a correção de problemas.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo principal é desenvolver um protótipo que será acoplado a um sistema de circuito fechado de TV (CFTV) e realizará o gerenciamento contra falhas de forma autônoma ou remota. Para isso, será desenvolvida uma placa de circuito impresso (PCI) que integrará os circuitos de controle com o microcontrolador WT32-ETH01 e um SBC *Orange Pi.* Em princípio o método de destravamento será retirar o fornecimento de energia da tomada 110V/220V, executando assim um *reset* forçado do equipamento travado.

1.3.2 Objetivos Específicos

- Desenvolvimento do *hardware* que realizará a integração entre o WT32-ETH01, a *Orange Pi* e o sistema de CFTV.
- Desenvolvimento do *firmware* do WT32-ETH01 que controlará o sistema.
- Integração com o banco de dados MySQL.
- Comunicação com o servidor para gerar um *dashboard*.

1.4 Justificativa

O monitoramento remoto em condomínios residenciais têm aumentado ao longo dos anos, principalmente devido a pandemia de COVID-19 (FERNANDES, 2020). Com a implantação das portarias eletrônicas, é possível ter apenas um funcionário que administra vários conjuntos de apartamentos.

Este arranjo de câmeras também pode estar disposto de forma simples, atendendo a necessidade de microempresários, como podemos notar no trabalho de conclusão de curso de Bravo (2015), onde foi implantado um sistema simples de câmeras IP para uma loja de roupas.

Esta tecnologia de monitoramento remoto é muito ampla e pode ser utilizada em outras áreas. Como abordado nos trabalhos de conclusão de curso de Victor (2019) e Claro (2011), podemos utilizar essa tecnologia para controlar os sinais vitais de pacientes sem a necessidade de termos um profissional no local. Isso se estende também a dispositivos que auxiliam os pacientes idosos a não esquecerem de tomar os remédios na hora certa com um dispositivo que libera comprimidos e que pode ser customizado remotamente, como foi abordado no trabalho de Santos (2019).

A partir dessa demanda por sistemas remotos, este projeto propõe uma solução de utilidade prática para o monitoramento de falhas de equipamentos de forma autônoma ou de forma remota, com potencial de venda para microempresas e condomínios.

O objetivo é monitorar os equipamentos que estão conectados a esse sistema de monitoramento contra falhas e travamentos. Bem como, problemas de internet e falta de energia. Este protótipo poderá resolver esses problemas de forma rápida e sem a necessidade do deslocamento de uma equipe técnica, ao local da falha. Assim, haverá uma economia de tempo e de dinheiro para a resolução de diversos problemas técnicos.

2 REFERENCIAL TEÓRICO

2.1 WT32-ETH01

WT32-ETH01 é um microcontrolador SoC (System on Chip), que possui Wi-Fi integrado de 802.11 b/g/n, Bluetooth versão 4.2 do tipo *Dual-Mode*, ou seja, possui o Bluetooth clássico e Bluetooth *low energy* (BLE), além de uma variedade de periféricos. É um sucessor avançado do chip 8266, destacando-se pela implementação de dois núcleos que operam em diferentes frequências, até um limite de *clock* de 240 MHz (BABIUCH, FOLTÝNEK, SMUTNÝ, 2019; MAIER, SHARP, VAGAPOV, 2017).

2.2 HTTP

O protocolo HTTP (*HyperText Transfer Protocol*) é baseado em um modelo de requisição e resposta para a transferência de arquivos *web*. Cada transferência consiste em uma requisição realizada pelo cliente, requisitando um arquivo de um servidor *web*, e uma resposta do servidor com o arquivo requisitado, ou uma mensagem de erro (MAH,1997).

Desenvolvido no início dos anos 1990, HTTP é um protocolo extensível que evoluiu ao longo do tempo. É um protocolo da camada de aplicação que é enviado por uma conexão TCP (*Transmission Control Protocol)* e que é utilizado não somente para buscar documentos HTML (*HyperText Markup Language*), mas também imagens ou vídeos (MOZILLA, 2024).

2.3 TCP

TCP (*Transmission Control Protocol)* é um protocolo de rede que permite a conexão e troca de dados entre dois dispositivos distintos. Este protocolo garante que os pacotes de dados serão recebidos na mesma ordem que foram enviados (MOZILLA, 2024).

Os protocolos HTTP e TCP possuem interação direta, pois as requisições HTTP necessitam que seja estabelecida uma conexão persistente entre o servidor e o dispositivo que realiza a requisição. Cada conexão TCP estabelecida pode ser utilizada para realizar pelo menos uma requisição HTTP ao servidor (MAH,1997).

2.4 Modem

O modem é um dispositivo eletrônico que modula um sinal digital em uma onda analógica e que demodula o sinal analógico e o converte novamente para o formato digital original, criando uma comunicação entre dois pontos (TECHTUDO, 2013).

Figura 3 - Modem

Fonte: (TECHTUDO, 2013)

2.5 Roteador

O roteador, ou *router*, é um dispositivo de rede que opera na camada de rede número 3 do modelo OSI (*Open Systems Interconnection*), responsável por encaminhar pacotes de dados entre diferentes redes. Ele age como intermediário, utilizando uma tabela de roteamento para determinar o melhor caminho para os pacotes de dados, garantindo a comunicação eficiente entre redes locais (LANs) e redes estendidas (WANs). Roteadores são essenciais em ambientes onde é necessário interligar redes distintas, mantendo-as isoladas, mas permitindo a troca de informações de forma controlada (TECHTUDO, 2013; ELETRONET, 2024).

Figura 4 - Roteador

Fonte: (TECHTUDO, 2013)

2.6 *Hub* de rede

O *hub* é um dispositivo de rede que atua como um ponto central de conexão para vários dispositivos em uma rede local. Sua função principal é conectar computadores e permitir a transmissão de informações entre eles. Entretanto, o *hub* opera retransmitindo os pacotes de dados recebidos para todas as suas portas de saída, sem a capacidade de filtrar ou direcionar o tráfego com base em endereços IP ou MAC. Essa falta de segmentação resulta em um tráfego intenso e inseguro, pois os dados são expostos a todos os dispositivos conectados (TECHTUDO, 2013; ELETRONET, 2024).

Figura 5 - Hub de rede

Fonte: (TECHTUDO, 2013)

2.7 *Switch* de rede

O *switch* de rede opera na camada de enlace de dados do modelo OSI. Criado para superar as limitações dos *hubs*, ele gerencia o fluxo de dados em redes locais (LAN) de forma eficiente, utilizando o endereço MAC para encaminhar pacotes diretamente ao destinatário correto. Isso reduz o tráfego na rede e minimiza a latência, garantindo maior velocidade e qualidade na transmissão de dados. Além disso, existem dois tipos de *switches*: os gerenciáveis e os não gerenciáveis, oferecendo diferentes níveis de controle e monitoramento remoto da rede. (TECHTUDO, 2013; ELETRONET, 2024).

Figura 6 - Switch de rede

Fonte: (TECHTUDO, 2013)

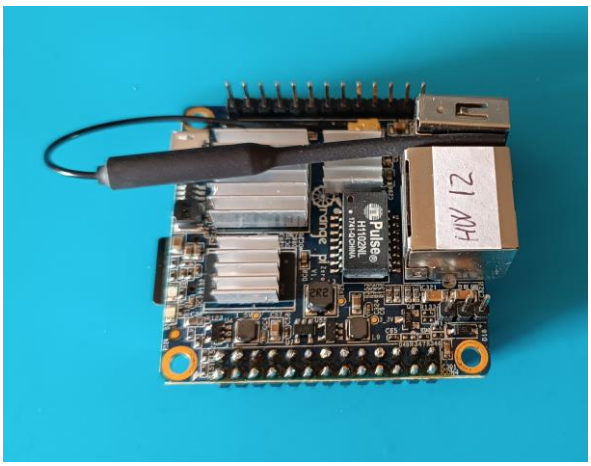
2.8 *Single Board Computer*

Um *Single Board Computer* (SBC) é um computador completo em uma única placa que integra processador, memória e entradas e saídas (GPIOs). Os SBCs são projetados de maneira diferente dos computadores de mesa (*desktop*) ou portáteis (*laptops*).

As principais vantagens dos SBCs incluem a integração eficiente de seus recursos, a facilidade de produção, além de serem mais leves, compactos e mais eficientes no consumo de energia.

Os SBCs são amplamente utilizados em aplicações embarcadas, controle de processos, sistemas robóticos complexos e outras aplicações que demandam processamento intensivo (TECNOBLOG, 2022; TECHOPEDIA, 2017).

Figura 7 - Orange Pi SBC



Fonte: Autoria própria (2024)

2.9 *Nobreak*

O *nobreak* é um equipamento que protege e mantém em funcionamento dispositivos eletroeletrônicos durante oscilações ou quedas de energia elétrica, isso ocorre pois ele possui uma bateria interna. Além disso, ele conta com um sistema de estabilização inteligente para evitar queima e o mau funcionamento dos aparelhos conectados (INTELBRAS, 2020).

Figura 8 - Nobreak

Fonte: Weg

3 DESENVOLVIMENTO

Para o desenvolvimento inicial, optou-se por utilizar um *Arduino Nano* (ARDUINO, 2024), visto que já havia experiência com a placa e era necessário esperar a encomenda do módulo WT32-ETH01 ser importada. Essa decisão foi tomada a fim de acelerar o desenvolvimento inicial.

Outra vantagem é que a biblioteca de comandos básicos do *Arduino*, conhecida como *Arduino Framework*, também foi implementada para os microprocessadores de dois núcleos Tensilica Xtensa LX6 (MAIER, SHARP, VAGAPOV, 2017).

Devido à facilidade do desenvolvimento e popularidade da plataforma *Arduino* e por ela ser uma plataforma de código aberto (*open-source)*, a empresa Espressif resolveu desenvolver um plugin chamado *Arduino core* (ESPRESSIF SYSTEMS, 2024) para todo o seu portfólio de microcontroladores, que hoje são os Xtensa e os Risc-V, que possibilita o desenvolvimento utilizando a *Arduino IDE* ou qualquer outra plataforma compatível (BABIUCH, FOLTỲNEK, SMUTNỲ, 2019).

Para a portabilidade futura bastaria trocar as bibliotecas de *internet* e *watchdog timer,* pois os controladores e *drivers* são diferentes. Além de adicionar as bibliotecas exclusivas do WT32-ETH01 como a de *Wi-Fi* e OTA (*Over-The-Air*, biblioteca utilizada para atualizar o *firmware* pelo IP da placa).

3.1 Projeto de *Hardware*

3.1.1 Prototipagem

Inicialmente, foram realizados alguns testes simples de acionamento de relés para entender como seria idealizada a placa de circuito final. Para o acionamento dos relés foi utilizado um transistor bipolar de junção (TBJ) na função de chaveamento e na bobina do relé, um diodo de roda livre. Como pode ser visualizado na Figura 10.

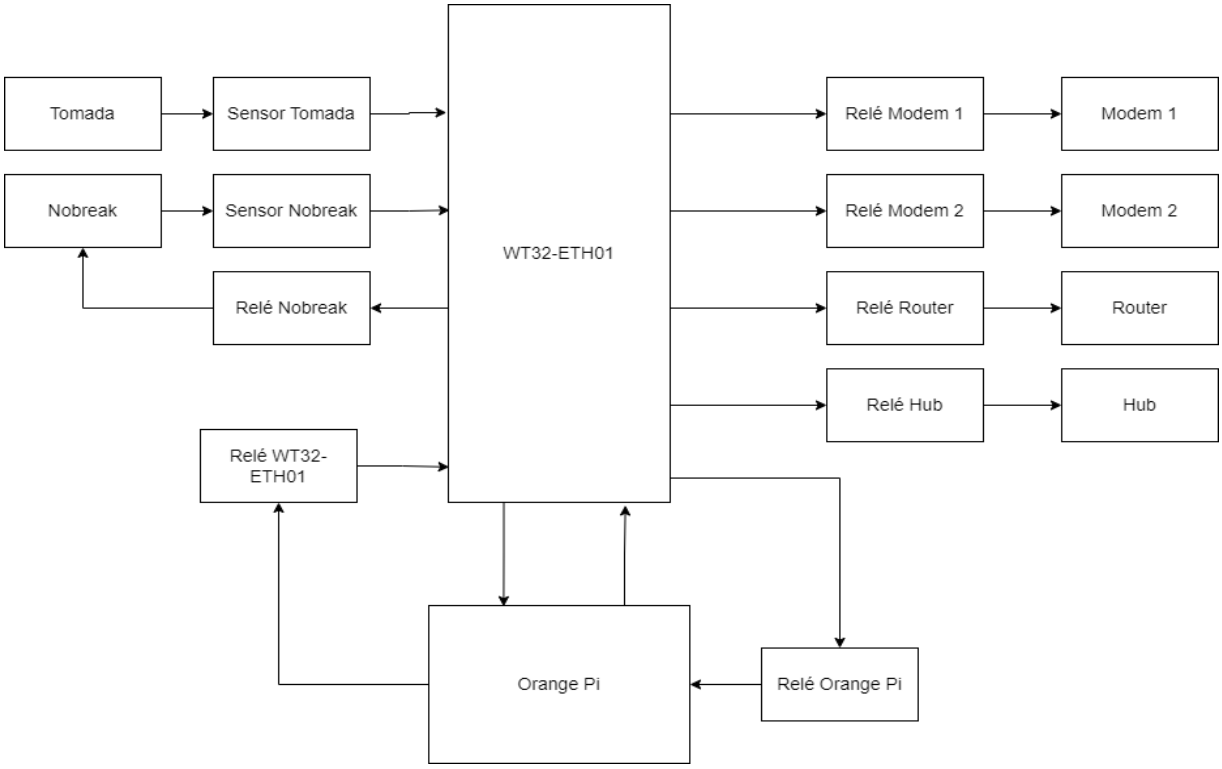
Em seguida, foi idealizado um circuito de *reset* do microcontrolador. Para isso, foi utilizada a técnica de *Power on Reset* (PoR), listado na Figura 16. Essa escolha foi feita, pois o PoR garante uma tensão estável e adequada para a realização do *reset* do circuito.

Por fim, o circuito final foi idealizado utilizando o *software EasyEDA,* pois nesse *software* pode ser realizado o desenvolvimento do circuito e da PCI, além de ser possível encomendar a placa profissional feita pela empresa JLCPCB.

3.1.2 Design do circuito

O circuito principal é composto de diversos blocos que são essenciais para o seu funcionamento. A seguir, será explicada a função de cada um desses blocos de forma concisa. Na Figura 9 temos uma representação simples de cada um desses blocos para a melhor compreensão do sistema.

Figura 9 - Diagrama de blocos do sistema



Fonte: Autoria própria (2024)

3.1.2.1 Acionamento dos relés

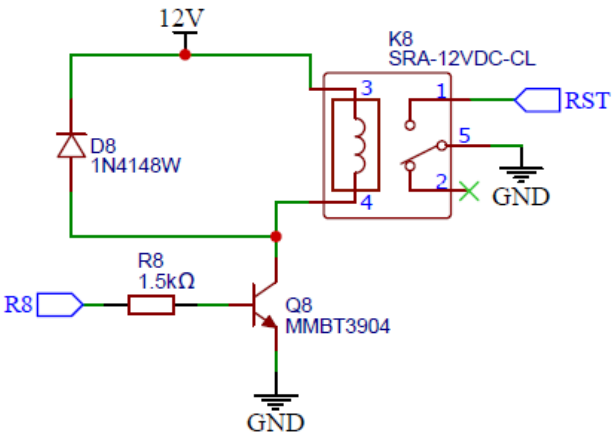
Para o acionamento dos relés, foi utilizado um circuito de *driver* de corrente com um transistor bipolar, como é ilustrado na Figura 10. O transistor bipolar de junção é utilizado como elemento de comutação do relé, e que é controlado por

corrente. Ele funciona em modo chave, fornecendo a corrente necessária no terminal do coletor para a alimentação correta do relé.

Há também um diodo de roda livre conectado em paralelo à carga indutiva do relé, na orientação reversa. Esse diodo serve para proteger o transistor e os demais componentes do circuito contra picos de tensão gerados pela carga indutiva do relé quando a bobina do relé é desenergizada. Assim, durante a operação normal do relé, o diodo está polarizado reversamente e não conduz corrente. E, quando a carga indutiva é desenergizada, o diodo fica polarizado diretamente e absorve a descarga do componente indutivo, protegendo o restante do circuito.

No caso do *nobreak*, como as versões mais comuns não possuem um sistema que liga o *nobreak* no caso em que a bateria acabou e a energia da tomada retornou, é necessário realizar a substituição do botão liga/desliga pelo circuito do relé para que o sistema consiga controlar esse acionamento, evitando assim que necessite de uma intervenção humana.

Figura 10 - Circuito de acionamento dos relés



Fonte: Autoria própria (2024)

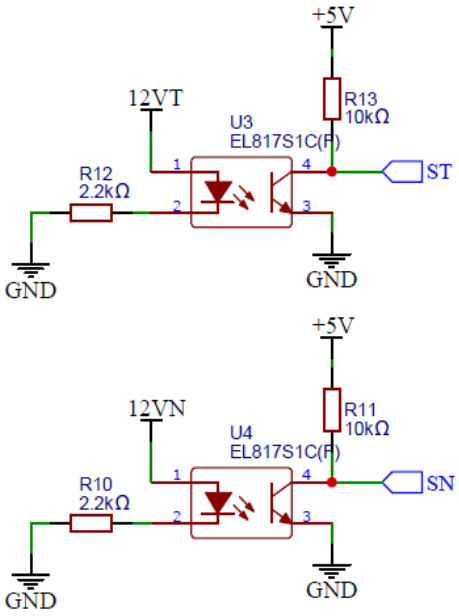
3.1.2.2Optoacoplador

O optoacoplador é um componente que possui um LED e um fototransistor. Quando o LED é acionado ele emite luz infravermelha que aciona o fototransistor. Ele conduz corrente elétrica proporcional à intensidade de luz recebida (ELECTRICITY & MAGNETISM, 2023). Este componente está sendo utilizado como

sensor de tensão para realizar a leitura da fonte do *nobreak* e da fonte da tomada, assim é possível alterar o valor da fonte de entrada sem alterar o circuito de saída do optoacoplador.

Estes circuitos optoacopladores da fonte do *nobreak* e da fonte da tomada estão representados na Figura 9. O sinal ST, do primeiro fotoacoplador, representa a leitura da tensão da tomada que será lido e gerará estatísticas para o *dashboard*. Já o sinal SN, do segundo fotoacoplador, serve para que o circuito autônomo de ligar o *nobreak* funcione. A tomada de decisões para cada uma das leituras pode ser visualizada na Tabela 1.

Figura 11 - Circuito dos optoacopladores



Fonte: Autoria própria (2024)

Tabela 1 - Tabela Verdade Optoacopladores

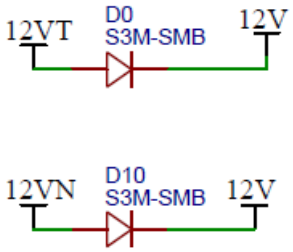
ST	SN	Resultado
0	0	Sistema desligado (sem energia).
0	1	Tomada sem energia. Manda o *status* para o *dashboard*.
1	0	*Nobreak* sem energia. Comando autônomo para reiniciá-lo.
1	1	Tomada e *nobreak* funcionando. Sistema OK.

Fonte: Autoria própria (2024)

3.1.2.3 Diodo para isolamento de Vcc

Este diodo atua como isolante elétrico entre diferentes fontes de tensão que fornecem 12V para que não haja passagem de corrente de uma fonte para outra, realizando o isolamento dos terminais Vcc. Para isso, foram utilizados diodos de 5A, visto que as fontes são de 3A. O circuito pode ser observado na Figura 12.

Figura 12 - Circuito para isolamento de tensão



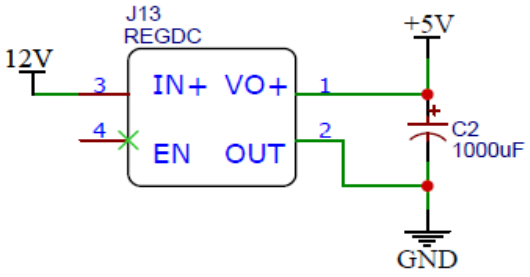
Fonte: Autoria própria (2024)

3.1.2.4 Regulador de tensão

O regulador de tensão escolhido para o projeto, é uma fonte chaveada que reduz a tensão de 12V para 5V. Esse módulo foi escolhido devido ao alto consumo do *Orange Pi*, que necessita de 3A. Por essa razão, não foi utilizado um regulador linear. E como a fonte chaveada realiza um chaveamento em alta frequência, ela acaba gerando um *ripple* um pouco maior que um regulador linear, assim foi utilizado um capacitor de 1000µF a fim de reduzi-lo.

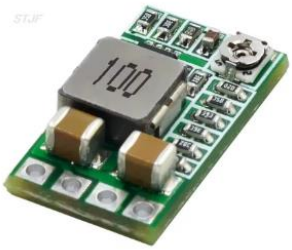
Este circuito pode ser visualizado na Figura 13 e a foto do módulo nas figuras 14 e 15.

Figura 13 - Circuito do regulador de tensão



Fonte: Autoria própria (2024)

Figura 14 - Regulador de tensão (frente)



Fonte: STJF

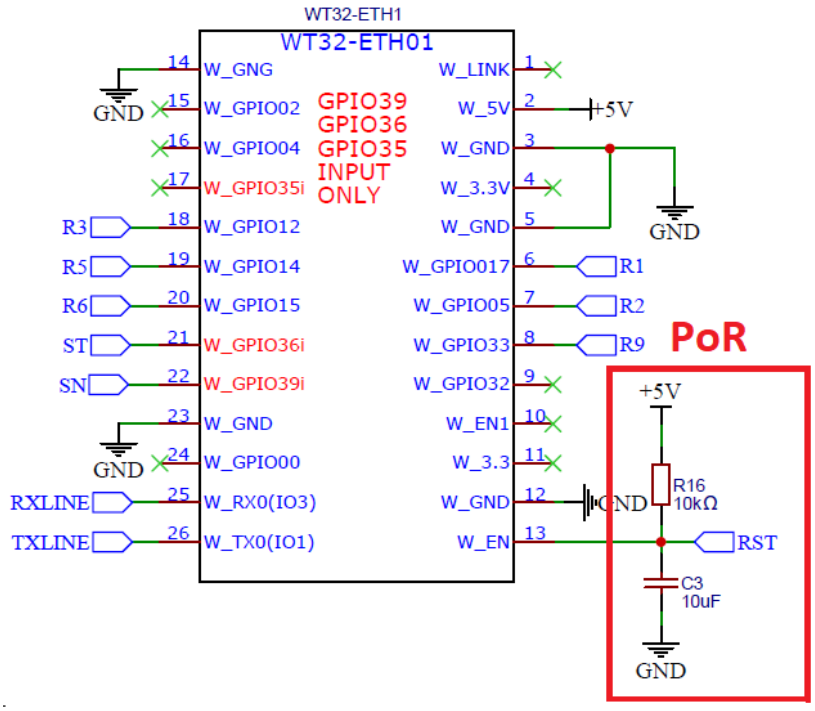
Figura 15 - Regulador de tensão (verso)

Fonte: STJF

3.1.2.5Power-On-Reset

O circuito PoR garante que o microcontrolador acoplado inicie corretamente, pois esse circuito executa um *reset* forçado na CPU fazendo com que ela só execute o *software* quando a tensão de alimentação atingir um nível mínimo e o *clock* estiver estável. Assim, este circuito proporcionará um *reset* consistente para o microcontrolador. Além disso, esse circuito protege o microcontrolador de picos de energia que possam causar travamentos no momento em que o circuito é energizado. Portanto, esse circuito é essencial para eliminar travamentos que possam ser causados por problemas de energia. Pode-se visualizar esse circuito na Figura 16.

Figura 16 - Circuito PoR

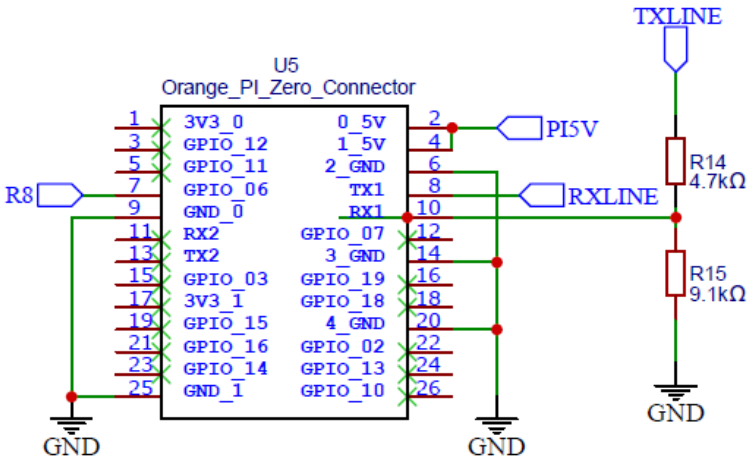


Fonte: Autoria própria (2024)

3.1.2.6Comunicação dos módulos com o Orange Pi

Para realizar a comunicação do módulo WT32-ETH01 com o *Orange Pi* não foi necessário nenhum ajuste, pois ambos trabalham com 3,3V. Porém, para o circuito do Arduino, foi necessária a criação de um divisor de tensão, pois a porta RX do Orange Pi opera com o nível de tensão de 3,3V e a porta TX do Arduino opera a 5V. Por isso, foram escolhidos os valores de resistor de 4,7kΩ e 9,1kΩ. Este divisor resistivo tem como finalidade reduzir o sinal de 5V para 3,3V, como se pode observar na Figura 17. A foto do módulo *Orange Pi* utilizado encontra-se anteriormente na Figura 7.

Figura 17 - Circuito Orange Pi



Fonte: Autoria própria (2024)

3.1.3 Placa de Circuito Impresso (PCI)

Para o desenvolvimento do circuito da placa foi utilizado o *software EasyEDA* onde foi idealizado o circuito esquemático e convertido para circuito impresso. O circuito esquemático completo está presente no ANEXO A.

A PCI foi feita utilizando componentes SMD e duas camadas. Para facilitar a localização dos componentes na placa também foi criada uma camada do tipo *silk*, indicando onde cada componente seria soldado. Nesta camada *silk,* foram adicionados o nome da placa, do autor e a versão atual do projeto.

Quando o desenvolvimento estava completo foi feita a encomenda da placa profissional pelo *site* da empresa JLCPCB. Como a PCI possuía diversos relés e alguns conectores, o que deixaria a encomenda mais cara, optou-se por encomendar apenas a placa com os componentes SMD soldados e realizar a solda do restante manualmente, utilizando um ferro de solda.

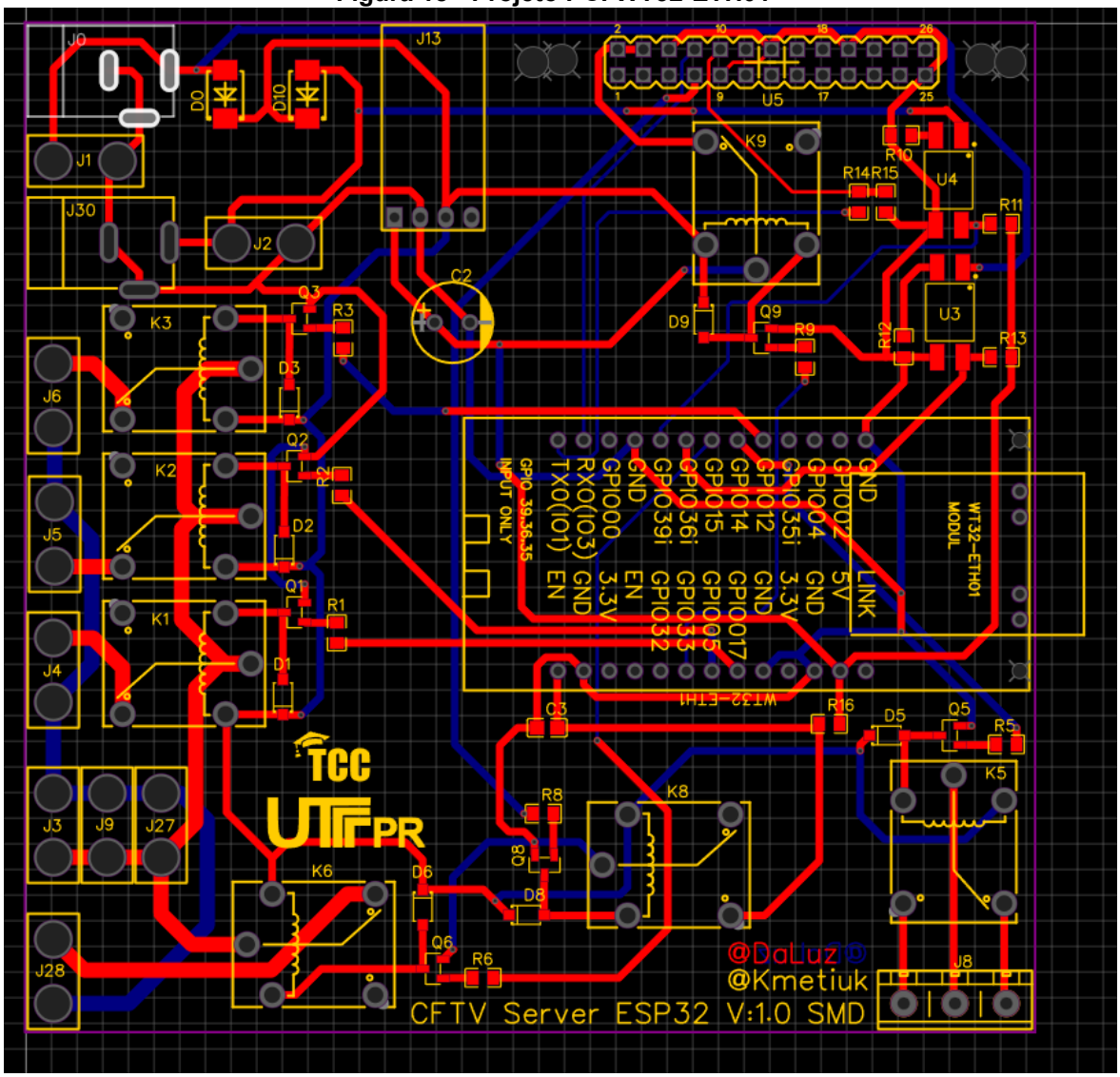
Como essa placa foi idealizada para ser comercializada, optou-se por realizar duas versões. Dessa forma, é possível gerenciar o custo de produção baseado no preço de cada um dos microcontroladores principais.

A primeira placa utiliza um módulo WT32-ETH01 (WIRELESS-TAG TECHNOLOGY CO. LTD, 2019). O circuito pode ser visualizado na Figura 18 e o resultado final da PCI nas figuras 19 e 20.

A segunda placa utiliza um *Arduino Nano* (ARDUINO, 2024) conectado a um módulo de *Ethernet* ENC28J60. O circuito pode ser visualizado na Figura 21 e o resultado final da PCI nas figuras 22 e 23.

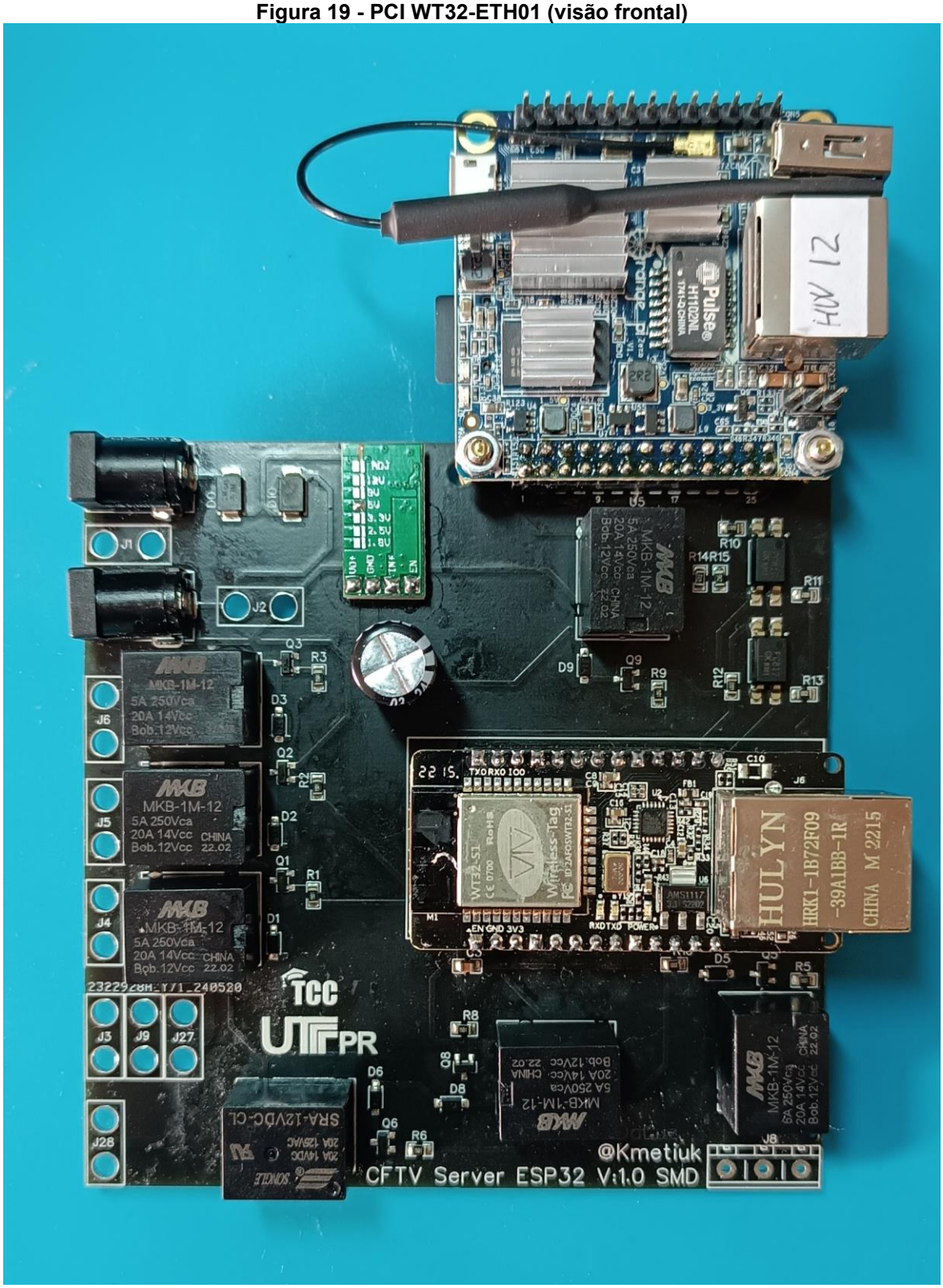
Ambas as placas utilizam um *Orange Pi Zero* como *Single Board Computer* (SBC) com *Linux* instalado.

Figura 18 - Projeto PCI WT32-ETH01



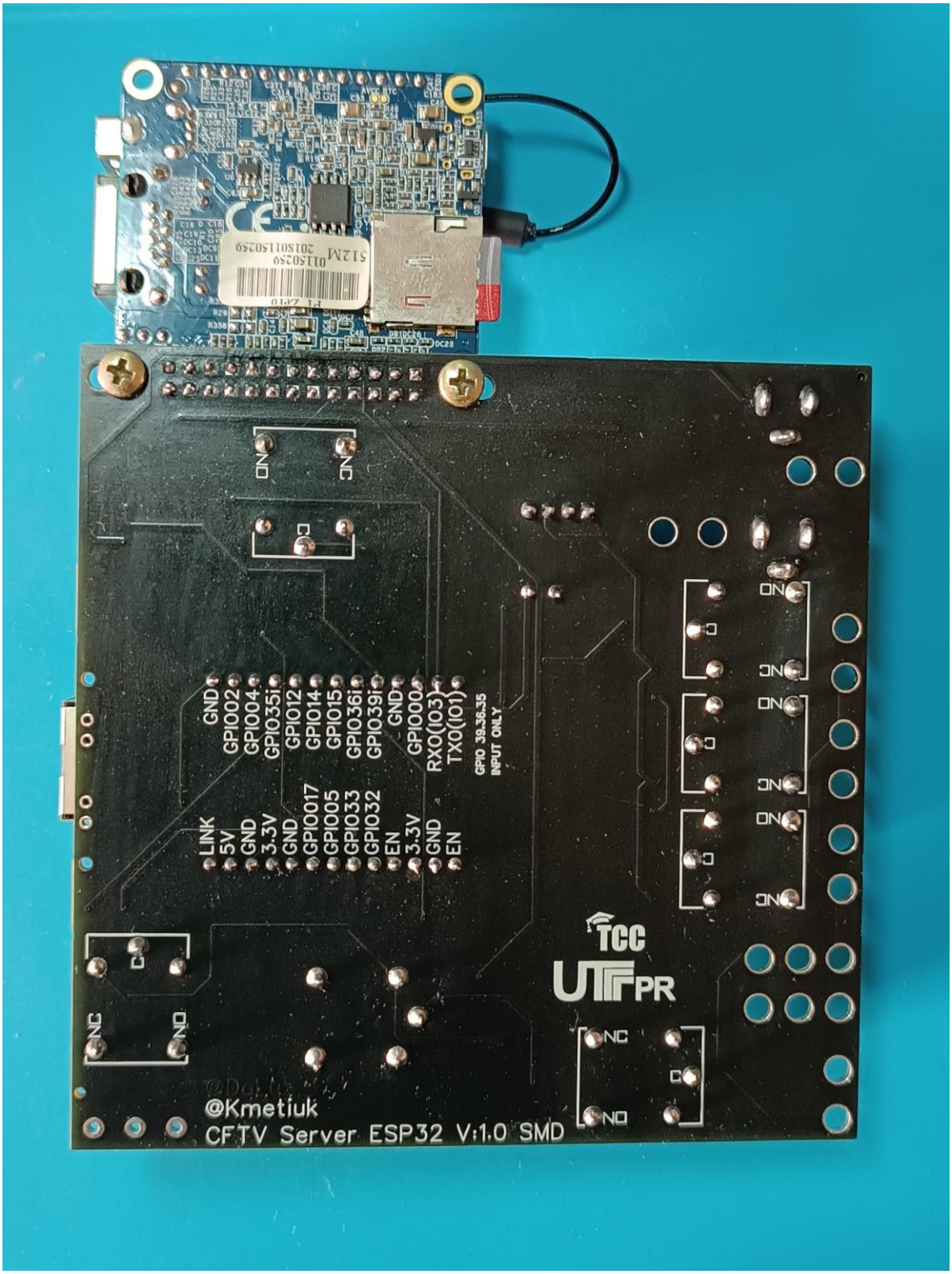
Fonte: Autoria própria (2024)

Figura 19 - PCI WT32-ETH01 (visão frontal)



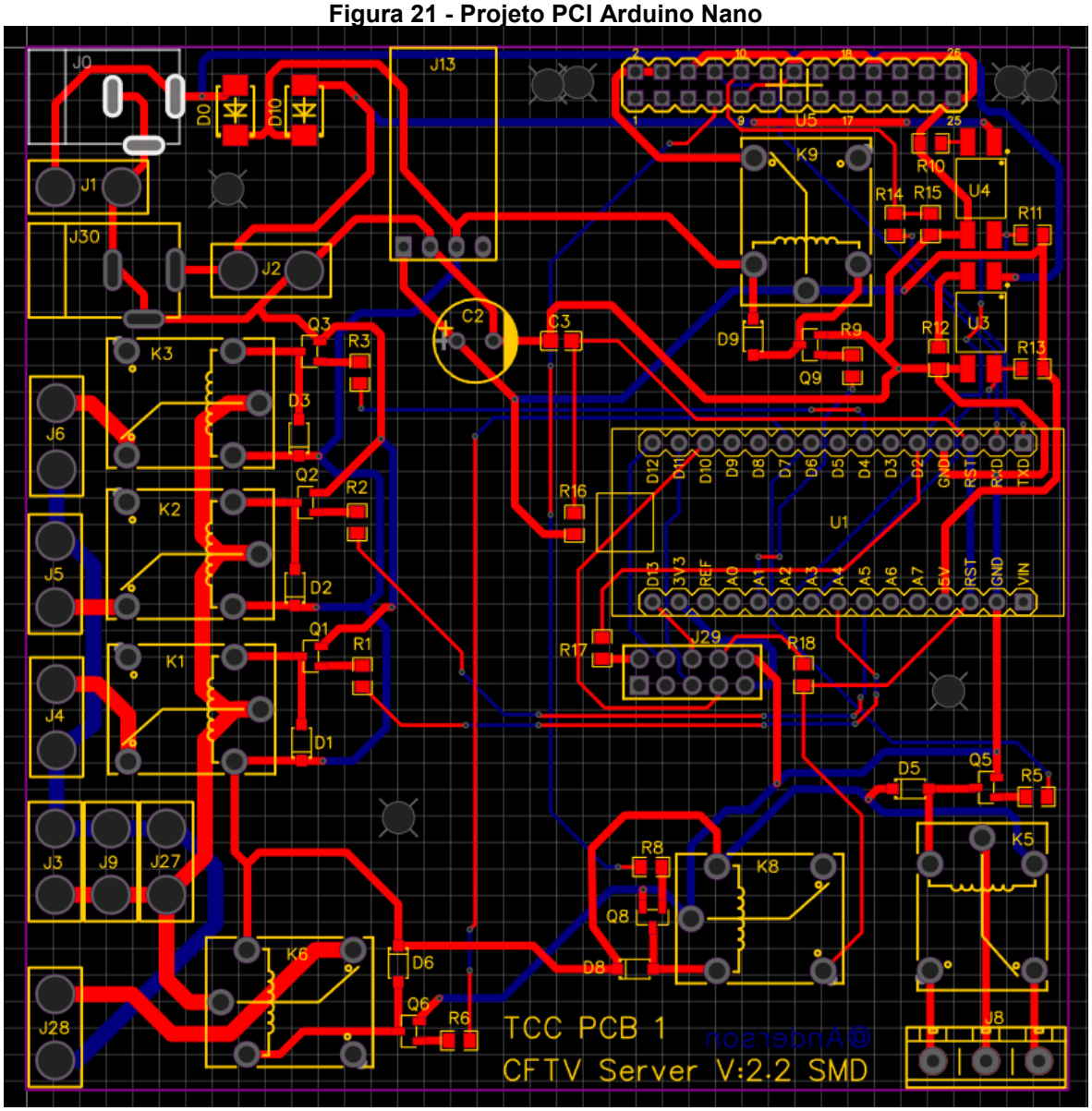
Fonte: Autoria própria (2024)

Figura 20 - PCI WT32-ETH01 (visão traseira)



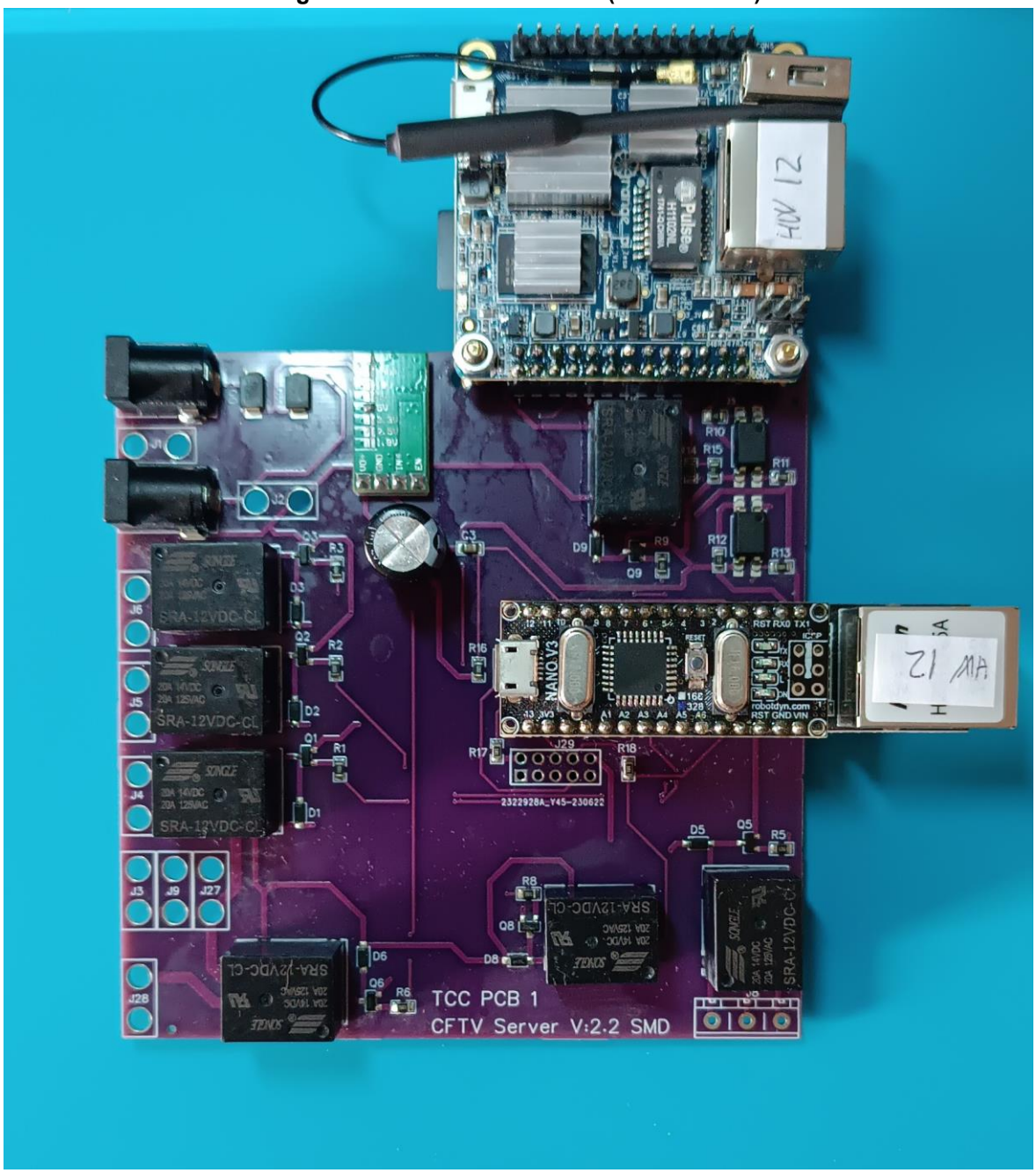
Fonte: Autoria própria (2024)

Figura 21 - Projeto PCI Arduino Nano



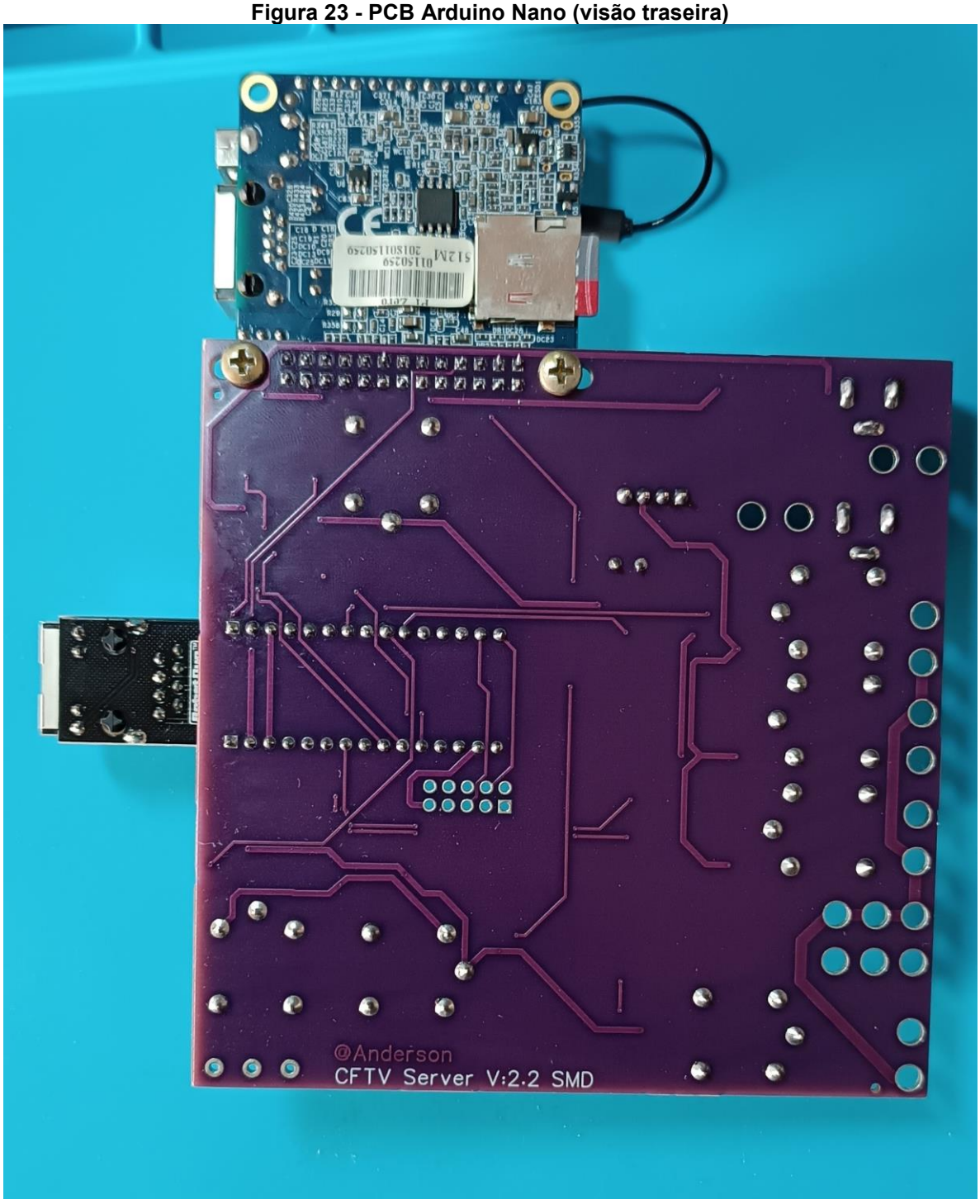
Fonte: Autoria própria (2024)

Figura 22 - PCI Arduino Nano (visão frontal)



Fonte: Autoria própria (2024)

Figura 23 - PCB Arduino Nano (visão traseira)



Fonte: Autoria própria (2024)

3.1.4 Montagem do protótipo

Na fase de prototipagem foi desenvolvida uma PCI personalizada para atender às especificações do projeto. No entanto, surgiu um desafio adicional: a necessidade de projetar uma caixa para abrigar e proteger o circuito. Para isso, era

necessário desenvolver uma caixa que fosse compacta e ao mesmo tempo confiável.

Entretanto, devido à falta de conhecimento e prazo limitado para o desenvolvimento de uma estrutura para abrigar o circuito, optou-se por adquirir uma caixa pronta e adaptá-la. Assim, a adaptação da caixa envolveu ajustes específicos para garantir que todos os componentes ficassem devidamente alojados e acessíveis, proporcionando ao mesmo tempo proteção e funcionalidade ao circuito. O primeiro protótipo junto com as fotos do mesmo e a primeira versão da PCI pode ser visualizado a seguir, nas figuras 24, 25, 26, 27, 28 e 29.

Figura 24 - Protótipo (visão de cima)

Fonte: Autoria própria (2024)

Figura 25 - Protótipo (visão frontal)

Fonte: Autoria própria (2024)

Figura 26 - Protótipo (visão traseira)

Fonte: Autoria própria (2024)

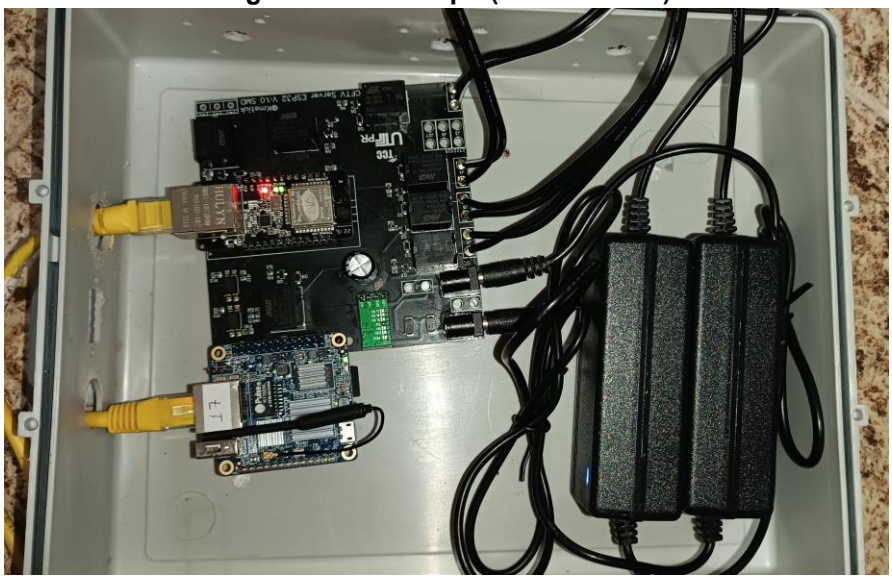
Figura 27 - Protótipo (visão lateral direita)

Fonte: Autoria própria (2024)

Figura 28 - Protótipo (visão lateral esquerda)

Fonte: Autoria própria (2024)

Figura 29 - Protótipo (visão interna)



Fonte: Autoria própria (2024)

3.2 Projeto de *Software*

3.2.1 Processo de desenvolvimento do software

Para o desenvolvimento do *software* foi utilizada a técnica de "*super loop*" e a temporização utilizando *software timers*. Essa abordagem permite que o programa seja executado de forma sequencial, sem que hajam atrasos consideráveis nas subrotinas. Não foi necessária a utilização de um sistema de tempo real, uma vez que não existem acionamentos críticos que exijam uma precisão extrema.

Inicialmente, foram realizados alguns testes simples de acionamento de relés. Após esses testes, foram criados alguns comandos que seriam utilizados para acionar os dispositivos que seriam conectados aos relés do circuito. Para esses acionamentos iniciais, os comandos foram enviados pela serial a partir do terminal *serial* da IDE (*VS Code*).

Quando todos esses acionamentos de comandos estavam consistentes e com as temporizações corretas, foi realizada a leitura do *nobreak* para um dos acionamentos autônomos do circuito. Neste caso, o módulo WT32-ETH01 realiza o gerenciamento do acionamento do relé caso o *nobreak* esteja sem energia para reiniciá-lo.

Assim que todos esses comandos básicos estavam implementados, foi realizada a implementação do *watchdog timer* para o gerenciamento de travamentos. Em seguida, foi implementada a atualização *Over-The-Air* (OTA) e a conexão *Wi-Fi*. Dessa forma, tornou-se mais fácil de atualizar o *firmware* da placa e acelerou o desenvolvimento, pois bastava enviar o novo *firmware* pelo IP da placa conectado à rede.

Por fim, foi implementada a integração com a internet para receber os comandos do site e realizar os acionamentos corretos. Como os comandos básicos já haviam sido implementados, bastou apenas reutilizá-los quando uma requisição era recebida do servidor.

Dessa forma, o módulo realizaria a tarefa requisitada e enviaria a resposta para o servidor, confirmando que todos os comandos foram atendidos. Caso os comandos falhassem, o servidor continuaria a enviar o comando até ser respondido, o que pode ocorrer devido à instabilidade da internet, ocasionando perda de pacotes na comunicação entre o servidor e o microcontrolador.

Por isso são necessárias algumas etapas de verificação para garantir que o comando foi atendido. Quando esse comando é executado e a resposta é verificada, ele é retirado da fila de comandos.

3.2.2 Descrição do funcionamento

Para a explicação do funcionamento do projeto serão utilizados três modos de operação diferentes: o modo autônomo, a requisição pelo site e o acesso remoto.

3.2.2.1 Modo autônomo

Para esse modo de operação o WT32-ETH01 realizará a leitura dos sensores para verificar se existe alguma falha no sistema, como falta de internet, falta de energia ou algum travamento. Se ocorrer algum desses problemas, ele realizará o acionamento dos relés para reiniciar algum dos componentes do sistema.

Se por algum motivo o WT32-ETH01 travar, o *Orange Pi* realizará o destravamento dele. De forma análoga, o contrário também pode ocorrer, assim o WT32-ETH01 destravará o *Orange Pi*.

3.2.2.2 Requisição pelo site

Para esse modo de operação utiliza-se a interface *web*. Nela podem-se realizar requisições para o sistema que as enviará para o banco de dados MySQL e o sistema irá atendê-las por ordem de chegada e confirmação de que outra tarefa já terminou de ser executada.

Para a realização dessas requisições será utilizado a *API REST*, pois é uma alternativa que facilitará a comunicação com os servidores *web* para a realização de requisições em *HTTP*, como abordado por Kim (2022).

3.2.2.3 Acesso remoto

Para a realização do acesso remoto será utilizado o sistema operacional Linux que será implementado no *Orange Pi*. A função do *Orange Pi* nesse sistema é a de *Single Board Computer* (SBC). Essa é uma alternativa de baixo custo e que possui várias aplicações no campo de controle e comunicação *online*, como abordado nos artigos de Alee, Rahman e Ahmad (2011), Doerr (1978) e Fayazi, Colter e Youbi (2022).

Dessa forma, podem-se enviar comandos remotamente para reiniciar os componentes do sistema. Bem como, realizar a atualização do *firmware* e a configuração dos equipamentos.

Assim, não haverá necessidade de enviar um técnico para configurar uma câmera nova. Basta o usuário acoplar a câmera ao sistema que ela poderá ser configurada de forma remota.

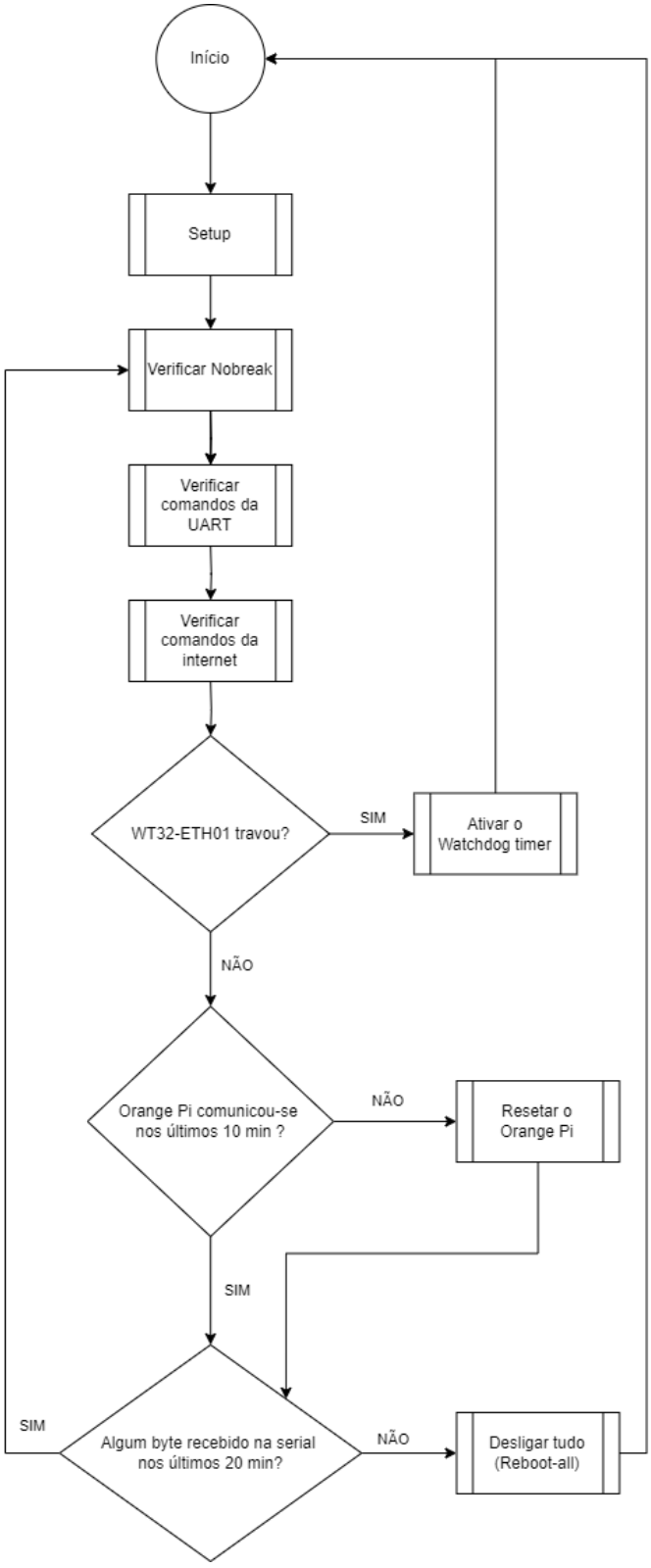
3.2.3 Fluxogramas

3.2.3.1 Principal

Primeiramente no fluxo principal do código ocorre o *setup* de todas as GPIOs e bibliotecas do WT32-ETH01. Em seguida, ocorre uma verificação do estado do *nobreak*, que pode ser reiniciado, de acordo com o fluxograma presente na Figura 31. Em seguida ocorre uma verificação dos comandos da porta *serial* (UART), que também pode ocorrer um tratamento do comando, de acordo com o fluxograma presente na Figura 33. Posteriormente, são verificados os comandos vindos do site,

que são tratados de acordo com o fluxograma da Figura 34. A seguir, o *watchdog timer* verifica se o módulo WT32-ETH01 travou, se o contador do *watchdog* não for reiniciado quer dizer que o sistema travou e ele ativará o *reset* do sistema. Na próxima condição, verifica-se se a *Orange Pi* está travada, caso contrário é realizado o reset. Por fim, a última verificação é a respeito dos bytes que são recebidos na serial, se não há recebimento de nenhum dado, quer dizer que o sistema travou e precisa ser totalmente reiniciado.

Figura 30 - Fluxograma do fluxo principal

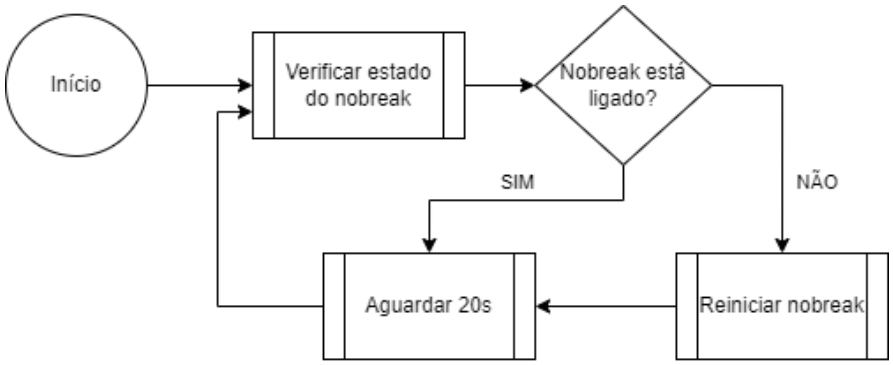


Fonte: Autoria própria (2024)

3.2.3.2Nobreak

Primeiramente, verifica-se o estado do *nobreak*. Se o *nobreak* não estiver ligado, o sistema tentará ligá-lo. Após executar esta tarefa, aguardará alguns segundos para fazer uma nova checagem. Caso contrário, a verificação será realizada a cada 20 segundos. Pode-se visualizar esse fluxo na Figura 31.

Figura 31 - Fluxograma do nobreak

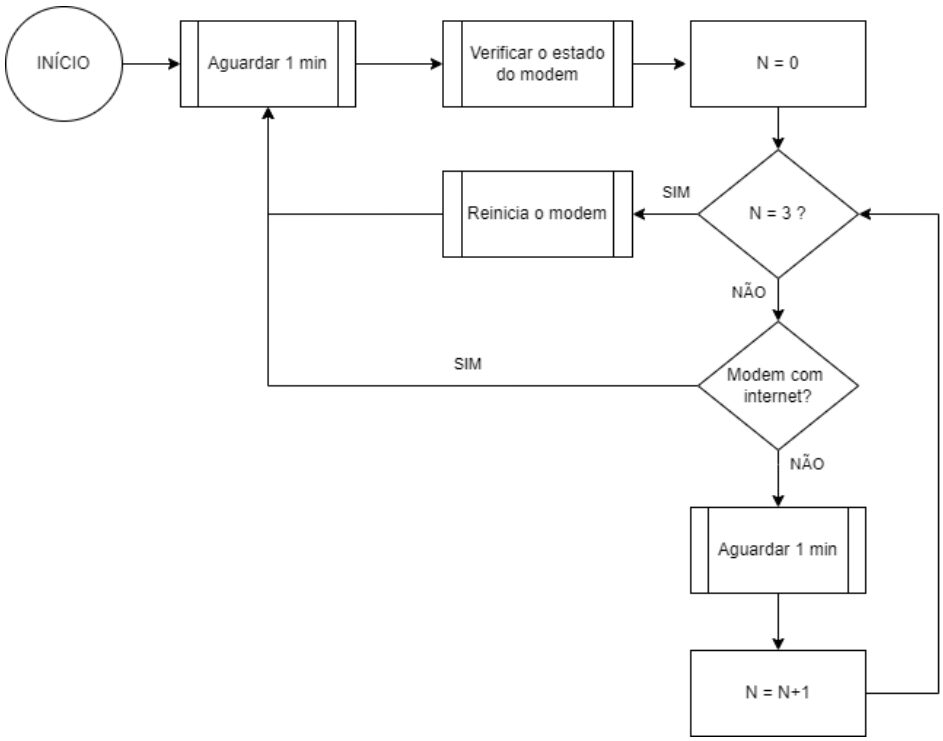


Fonte: Autoria própria (2024)

3.2.3.3Modem

Primeiramente, verifica-se se o modem está com internet. Caso o modem esteja sem internet, realizam-se até 3 requisições de internet, antes de realizar o seu desligamento. Pois é necessário garantir que o modem realmente está sem internet, pois pode existir uma perda de pacotes de rede em vez de uma falha de internet, que pode ser gerada por uma oscilação na conexão. Em seguida, espera-se 1 minuto para que o modem seja totalmente reiniciado, para assim realizar novas verificações. Esse fluxo pode ser visualizado na Figura 32.

Figura 32 - Fluxograma dos modems

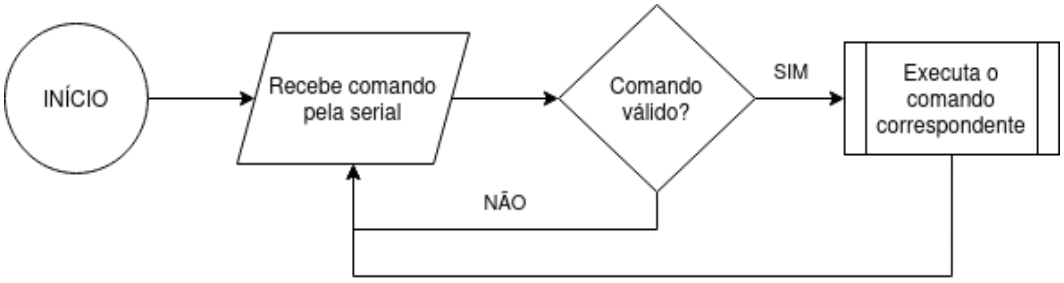


Fonte: Autoria própria (2024)

3.2.3.4UART

O módulo WT32-ETH01 avalia constantemente se existem novos comandos a serem recebidos pela porta *serial* (UART) e valida se os comandos recebidos são comandos válidos. Como pode ser visualizado na Figura 33.

Figura 33 - Fluxograma da UART



Fonte: Autoria própria (2024)

Todos esses comandos utilizam *software timers* construídos com o auxílio da função “millis()” (ARDUINO, 2024) que utiliza um *timer* interno configurado para gerar um incremento de um contador a cada milissegundo como referencial de

tempo, isso permite com que o fluxo principal do programa não seja afetado com rotinas de “delay()”, assim permitindo um fluxo sequencial sem atrasos, esta técnica é conhecida como “*super loop*”.

Para realizar todos os procedimentos de controle concebidos no projeto, foram necessários um total de 9 comandos. Estes comandos constituem um protocolo simples 4 bytes. Eles podem ser enviados pelo SBC via acesso remoto do usuário ou podem ser enviados via *dashboard*. Porém, os comandos presentes na interface do *dashboard* foram reduzidos de forma a simplificar a interface e contendo somente quatro comandos: “CMD0”, “CMD1”, “CMD2” e “CMD3”.

3.2.3.4.1 String de comando “CMD0”

O envio do comando “CMD0”, seja via UART ou via *web* através do *dashboard* ao módulo WT32-ETH01, realiza a reinicialização do *nobreak.* Para isso, o relé do *nobreak* é ativado durante 3 segundos.

3.2.3.4.2 String de comando “CMD1”

O envio do comando “CMD1”, seja via UART ou via *web* através do *dashboard* ao módulo WT32-ETH01, realiza a reinicialização dos *hubs.* Para isso, o relé do *hubs* é ativado durante 10 segundos.

3.2.3.4.3 String de comando “CMD2”

O envio do comando “CMD2”, seja via UART ou via *web* através do *dashboard* ao módulo WT32-ETH01, realiza a reinicialização de ambos os modems, o modem 1 e o modem 2. Para isso, o relé do modem 1 é ativado durante 10 segundos. Em seguida, espera-se 2 minutos para realizar o mesmo processo com o modem 2.

Vale ressaltar novamente que todos esses tempos de espera estão atrelados à função “millis()” (ARDUINO, 2024) para gerar os *software timers*, logo o fluxo do código não é alterado durante os tempos de espera. Outro ponto importante é que o tempo de reinicialização entre os dois modems serve para tentar garantir que o sistema ficará sem internet o menor tempo possível, visto que cada modem necessita um tempo de em torno de 1 minuto para estabilizar após ser reinicializado.

3.2.3.4.4 String de comando “CMD3”

O envio do comando “CMD3”, seja via UART ou via *web* através do *dashboard* ao módulo WT32-ETH01, realiza a reinicialização do *router.* Para isso, o relé do *router* é ativado durante 10 segundos.

3.2.3.4.5 String de comando “CMD5”

O envio do comando “CMD5”, via UART, realiza a reinicialização do modem 1. Para isso, o relé do modem 1 é ativado durante 10 segundos. Assim como na execução do comando CMD2, este modem precisa de um tempo para estabilizar, então as verificações do *status* da *internet* são desativados durante um período de 60 segundos.

3.2.3.4.6 String de comando “CMD6”

O envio do comando “CMD6”, via UART, realiza o processo equivalente à *string* de comando *“*CMD5”, porém nesse caso o processo é realizado para o modem 2.

3.2.3.4.7 String de comando “CMD9”

O envio do comando “CMD9”, via UART, representa o *status* do módulo *Orange Pi.* Quando esse comando é recebido, os tempos de *reset* do *Orange Pi* e do *reset* geral do sistema são reiniciados. O tempo padrão para a reinicialização do *Orange Pi* é de 10 minutos e do *reset* geral do sistema de 20 minutos.

Este comando permite que o sistema esteja em constante atualização, se esse comando não é recebido significa que o bloco do *Orange Pi* não está funcionando apropriadamente, logo precisa ser reiniciado. Porém se esse *reset* é realizado e, mesmo assim, em 20 minutos o sistema não comunica é necessário realizar um *reset* geral.

3.2.3.4.8 String de comando “STOP”

O envio do comando “STOP”, via UART, serve para desativar o *software timer* que reinicializa a contagem do *watchdog timer*, assim quando esse contador atingir o limite, o sistema deverá ser reiniciado automaticamente e será possível verificar se o *watchdog timer* está funcionando corretamente.

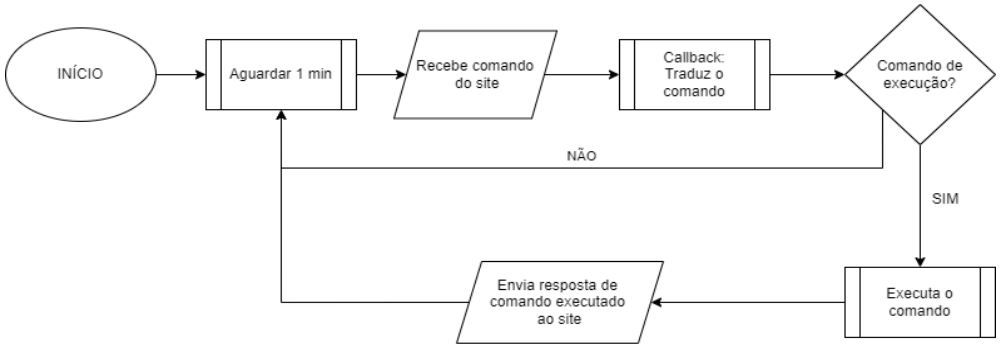
3.2.3.4.9 String de comando “RSTX”

O envio do comando “RSTX”, via UART, serve para forçar a reinicialização do módulo WT32-ETH01 utilizando a função de *reset* nativa deste microcontrolador, chamada “esp_restart()” (ESPRESSIF SYSTEMS, 2024).

3.2.3.5Comandos do site

Para o fluxo de comandos do site, primeiramente é feita uma requisição do tipo *GET* para verificar se existe algum comando na fila e para enviar alguns parâmetros que serão utilizados pelo *dashboard*. Em seguida, a requisição recebida é enviada para uma função de *callback* que realiza a tradução do comando. Se não houver nenhum comando na fila, espera-se o próximo minuto para realizar uma nova requisição. Caso haja um comando a ser executado, ele é enviado para o fluxo de execução. Após a execução, é enviada uma nova requisição do tipo *GET* para o servidor para confirmar que o comando foi atendido. Assim, o servidor removerá o comando da fila e o próximo comando será recebido quando ocorrer a nova requisição *web* via *dashboard*. Os novos comandos do site são verificados a cada minuto. Esse fluxo pode ser observado na Figura 34.

Figura 34 - Fluxograma dos comandos do site



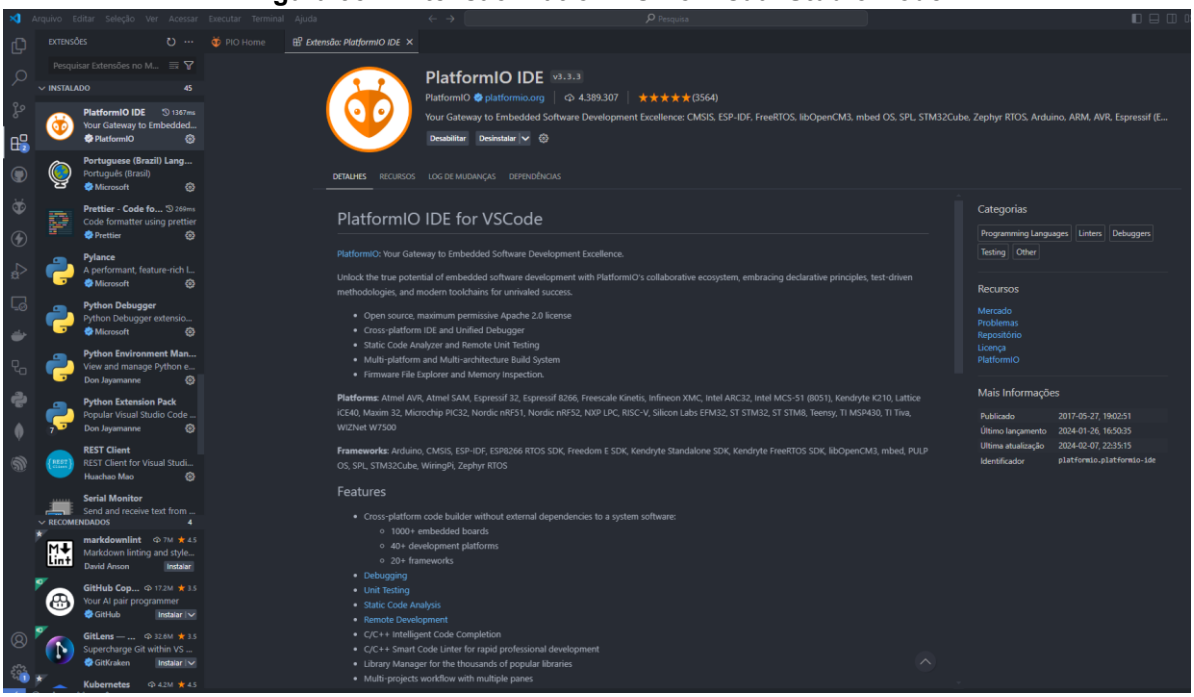
Fonte: Autoria própria (2024)

3.2.4 Configurações do ambiente

O *software* utilizado para gerar o *firmware* do microcontrolador, foi o *Visual Studio Code* com a extensão *PlatformIO IDE*, para o versionamento de código foi utilizado o *Git* e o código foi armazenado no *GitHub.*

Com o *Visual Studio Code* (*VS Code*) já instalado, é necessário instalar a extensão do *PlatformIO IDE* na aba de extensões. Basta procurar por "*PlatformIO IDE*" e clicar em "instalar", assim como na Figura 35. Após a instalação, o *VS Code* precisa ser reiniciado.

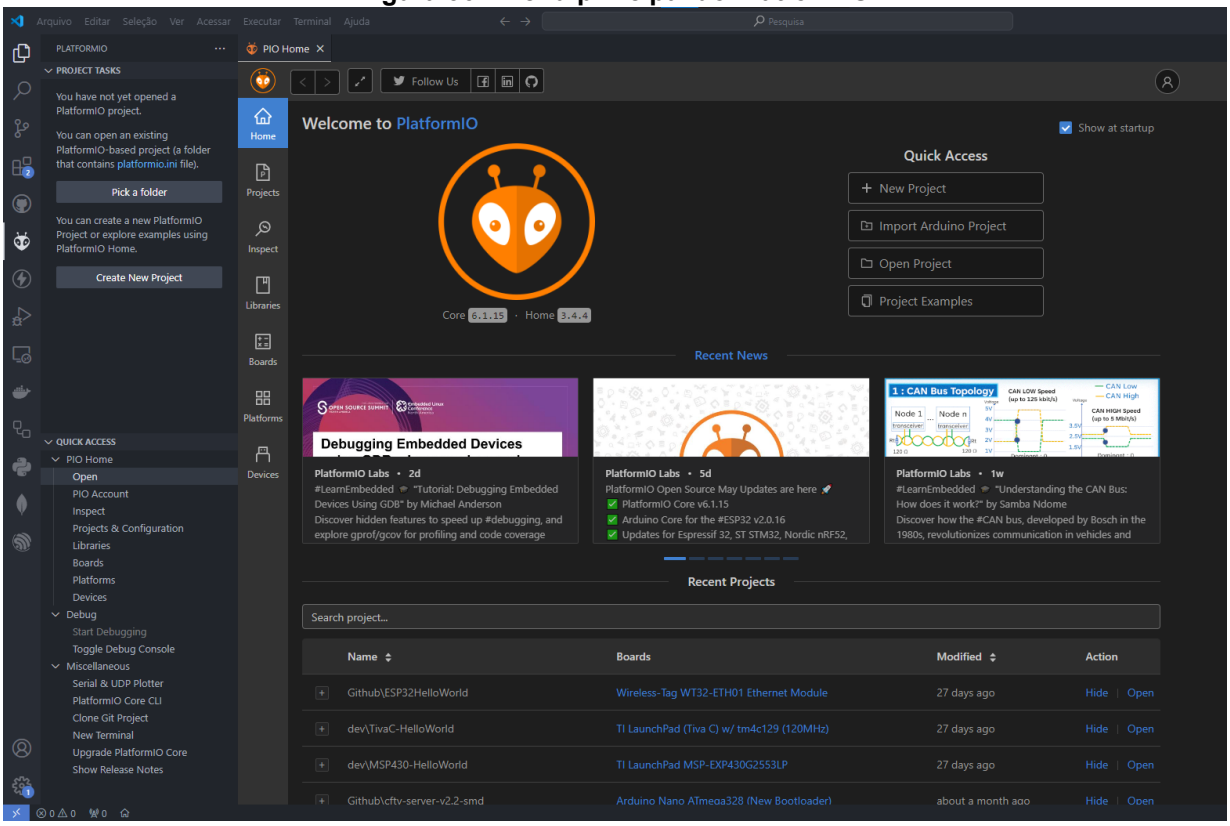
Figura 35 - Extensão PlatformIO no Visual Studio Code



Fonte: Autoria própria (2024)

Para criar um projeto com a extensão do *PlatformIO*, o usuário deve clicar no ícone do *PlatformIO* na barra lateral esquerda. Em seguida, ele deve selecionar "*New Project*", assim como é mostrado na Figura 36. Na janela que abrir, o usuário deve preencher os detalhes do projeto, como nome do projeto, placa de desenvolvimento que está sendo utilizada, *framework* e a pasta onde o projeto será salvo.

Figura 36 - Menu principal do PlatformIO

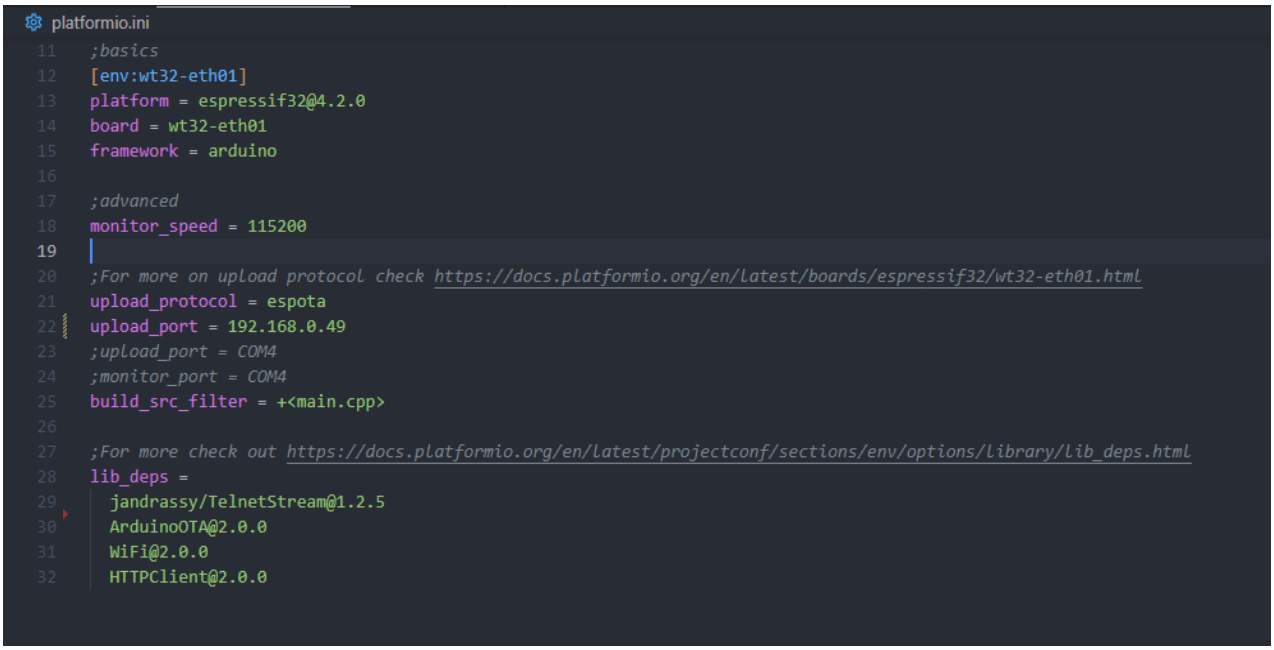


Fonte: Autoria própria (2024)

Com o novo projeto criado, o usuário poderá visualizar a estrutura do projeto no painel lateral, chamado de "*workspace*". Este projeto estará configurado de acordo com as configurações padrões do *PlatformIO*. A estrutura incluirá algumas pastas padrões como "src" para o código fonte, "*test*" para os testes unitários, "lib" para as bibliotecas, entre outras.

Um dos arquivos mais importantes no diretório padrão é o "platformio.ini". Este arquivo contém informações cruciais para o projeto, como nome da placa, plataforma utilizada e *framework*. Além disso, podem ser adicionadas configurações extras como a velocidade da serial, o protocolo para *upload* de código, a porta para *upload* do código e as dependências das bibliotecas com as respectivas versões utilizadas, como pode ser visualizado na Figura 37.

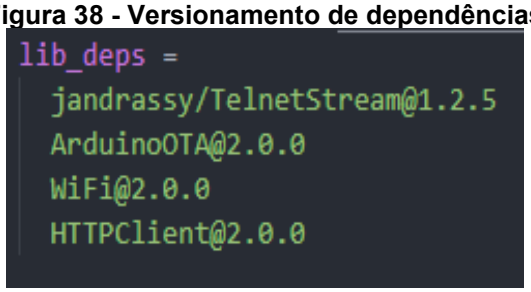
Figura 37 - Configurações principais do Platformio.ini



Fonte: Autoria própria (2024)

A melhor maneira de garantir que o projeto continue funcionando mesmo se alguma das dependências for atualizada, é utilizar o nome da biblioteca seguido de um “@” e o número da versão específica que será utilizada. Assim como é demonstrado na Figura 38.

Figura 38 - Versionamento de dependências



Fonte: Autoria própria (2024)

3.2.4.1 Gravação do *Software*

Como o módulo WT32-ETH01 não possui um gravador integrado. Foi utilizado um conversor USB-UART do modelo CP2102 (Figura 39) para realizar a gravação do *firmware* no WT32-ETH01 até a implementação do método de gravação *Over-The-Air* (OTA). Para isso é necessário conectar os terminais do conversor aos

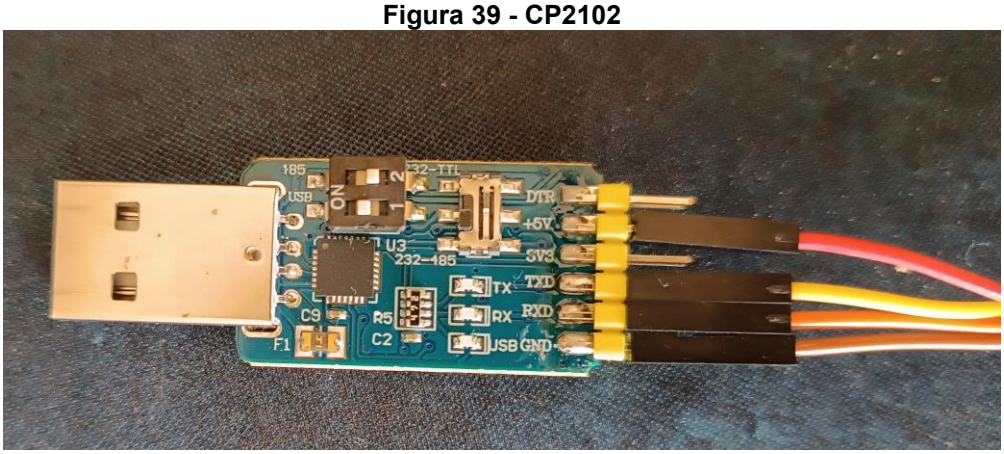
respectivos terminais do módulo WT32-ETH01, atentando para a inversão dos terminais RX e TX das duas placas. É necessário também que o pino IO0 do WT32-ETH01 esteja conectado ao terminal GND. Quando o IO0 está conectado ao GND, o WT32-ETH01 entra em modo de gravação serial. A conexão simplificada dos pinos pode ser visualizada na Tabela 2.

Tabela 2 - Conexões WT32-ETH01 e CP2102

WT32 – ETH01	CP2102
RX	TX
TX	RX
IO0	GND
5V ou 3,3V	5V ou 3,3V
GND	GND

Fonte: Autoria própria (2024)

Figura 39 - CP2102



Fonte: Autoria própria (2024)

Assim que todas essas condições forem satisfeitas, basta conectar o CP2102 na porta USB do computador e verificar se os *drivers* estão de acordo e o dispositivo está sendo reconhecido. Em seguida, utiliza-se o botão "*upload*" na extensão do *PlatformIO.* Assim o código será compilado e carregado no WT32-ETH01.

3.2.4.2 Gravação de *Software* O*ver-The-Air*

Para realizar a gravação *Over-The-Air*, é necessário que o módulo WT32-ETH01 esteja conectado à rede local, utilizando a conexão *ethernet* ou *Wi-Fi,* e que

o código que realiza a ativação do OTA tenha sido previamente gravado na placa utilizando o método listado anteriormente, com o conversor CP2102.

Nas figuras 40, 41 e 42 existe um trecho de código que pode ser utilizado para essa configuração inicial.

Figura 40 - Código exemplo OTA (Parte 1)


```
#include <Arduino.h>

#include <secrets.h> //passwords - This file is in the .gitignore
//just create a file with #define and the value of password and network name

#include <ArduinoOTA.h>
#include <WiFi.h>
#include <TelnetStream.h>

#define LED 14
#define LEDTIME 500

const char ssid[] = SECRET_SSID;    // your network SSID (name)
const char pass[] = SECRET_PASS;    // your network password (use for WPA, or use as key for WEP)

void setup() {
  Serial.begin(115200);

  //WIFI
  Serial.print("Attempting to connect to WPA SSID: ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  // Wait for WiFi connection
  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Failed to connect.");
    while (1) {
      delay(10);
    }
  }

  IPAddress ip = WiFi.localIP();
  Serial.println();
  Serial.println("Connected to WiFi network.");
  Serial.print("Connect with Telnet client to ");
  Serial.println(ip);

  TelnetStream.begin();

  //OTA
  ArduinoOTA.setHostname("ESP32-OTA");
  ArduinoOTA.begin();

  // LED Setup
  pinMode(LED,OUTPUT);
  digitalWrite(LED, LOW);
}
```

Fonte: Autoria própria (2024)

Na Figura 40, o bloco de código número “1” representa as bibliotecas que serão utilizadas nesse exemplo. Para isso, o módulo WT32-ETH01 será conectado à rede utilizando a conexão *Wi-Fi*, com a biblioteca “*WiFi.h*”, e a gravação OTA utilizará a biblioteca *“ArduinoOTA.h”*. Por mais que essa biblioteca possua o nome de *“Arduino”,* ela é compatível com o módulo WT32-ETH01.

No segundo bloco desta figura, enumerado “2”, representa a tentativa do módulo de estabelecer uma conexão *Wi-Fi*, para isso são necessárias as credenciais corretas da rede. Essas credenciais estão definidas no arquivo “secrets.h” que possui os valores de “SECRET_SSID” e “SECRET_PASS” que serão atribuídos na pré-compilação do código para as variáveis “ssid” e “pass”, respectivamente, e estas serão utilizadas pelas funções de conexão da biblioteca

Wi-Fi. Por fim, o bloco de número “3” inicializa a biblioteca do OTA para que ela funcione de acordo.

Além desses três passos, é necessário adicionar um *handler* no programa principal que inicializará a gravação do novo *firmware* utilizando o OTA, como é demonstrado na Figura 41, no bloco de número “4”. Para isso, é aconselhável não utilizar temporizações que possam interromper a execução do fluxo principal de código, como a função “delay()”, assim como é mostrado no código exemplo abaixo do bloco “4”, pois isso pode atrapalhar o *handler* no processo de tratamento de uma requisição de *update* de *firmware* via OTA.

Figura 41 - Código exemplo OTA (Parte 2)

Fonte: Autoria própria (2024)

O último passo é modificar o arquivo “platformio.ini” adicionando o protocolo de *upload* como “espota” e o IP para *upload* do *firmware* na variável “upload_port”, assim como é exemplificado na Figura 42, no bloco de número “5”. O valor de IP do módulo WT32-ETH01 pode ser descoberto acessando as configurações do modem ou adicionando no código um comando para ser enviado pela porta *serial* (UART) do WT32-ETH01 e que será monitorado pelo terminal *serial* da IDE.

Figura 42 - Código exemplo OTA (Parte 3)

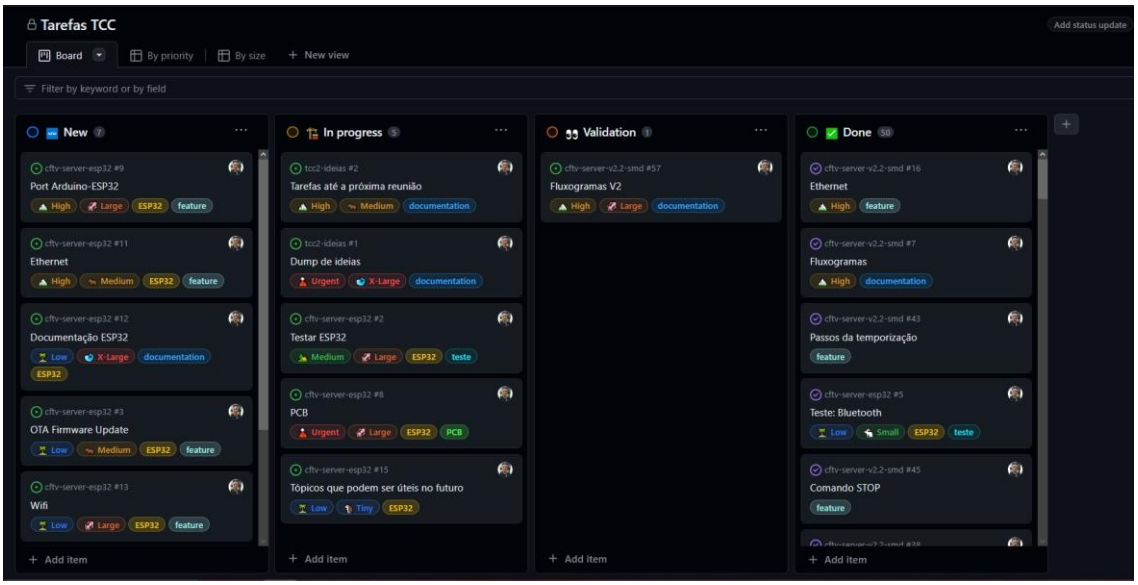
Fonte: Autoria própria (2024)

Para mais detalhes de como utilizar o código exemplo e realizar a gravação OTA, checar o ANEXO B.

3.2.4.3 Organização de tarefas

Para a organização de tarefas foi utilizado o GitHub Projects (Figura 43) onde todas as tarefas foram quebradas em tarefas menores utilizando métodos ágeis e foram colocadas em um *board* e separadas em categorias para a melhor visualização e gerenciamento.

Figura 43 - GitHub Projects



Fonte: Autoria própria (2024)

3.2.5 Desenvolvimento e implementação

O *software* utilizado foi o *Visual Studio Code* com a extensão *PlatformIO IDE*, para o versionamento de código foi utilizado o Git e o código foi armazenado no GitHub.

3.2.5.1 Comandos básicos

Inicialmente, foram implementados comandos básicos com acionamentos simples e leitura de níveis de tensão nas portas do módulo WT32-ETH01. Dessa forma, foi possível obter uma visão mais ampla de como estruturar o código e, ao mesmo tempo, o *hardware* para o desenvolvimento da PCI.

3.2.5.2 Leitura da serial

Para a leitura da serial, optou-se por utilizar a lógica de rotação de bytes em um *buffer* que recebe as mensagens da serial.

Essa lógica funciona da seguinte forma: é utilizado um *buffer* de 4 bytes. Toda vez que um byte novo chega na serial, ele é colocado na última posição e os bytes que já estavam na serial são deslocados 1 byte a esquerda. Dessa forma, quando os 4 bytes de um comando qualquer são recebidos, eles estarão alinhados da forma correta e poderão ser lidos. Essa lógica evita a necessidade de realizar

diversas verificações a cada vez que um byte novo chega. Assim, é feita uma verificação geral a cada vez que um byte é recebido. Nessa verificação, o sistema verifica se a palavra formada no buffer corresponde a algum dos comandos da placa. Caso contrário, ele continuará esperando os próximos bytes. Essa lógica também auxilia na recepção de comandos seguidos, pois eles poderão ser interpretados corretamente e com rapidez.

Outra abordagem que poderia ser utilizada, é enviar um byte específico para o começo de um comando e um byte para o final, porém, essa forma não é tão confiável pois podem haver perdas de bytes na transmissão e se o sistema não identificar o byte específico que sinaliza o início do comando, a placa pode acabar não executando esse comando.

Por esse motivo foi escolhida a forma da qual foi implementada. Assim, qualquer comando que for recebido pela serial será organizado e interpretado.

3.2.5.3 *Nobreak* e tomada

O programa desenvolvido realiza varredura de dois sensores de tensão, presentes na Figura 9, para analisar o estado que se encontra a tomada e o *nobreak.*

Para a tomada é realizado o *upload* de seu estado atual ao *dashboard* do *site,* gerando algumas estatísticas. Já para o *nobreak,* é realizada a verificação e se ele se encontra desligado, o acionamento do respectivo relé é realizado de forma autônoma para a reinicialização do *nobreak*.

Neste caso, foi realizada uma adaptação para o acionamento do botão original de *on/off* do *nobreak,* ou seja, um cabo foi conectado ao relé da PCI e ao botão *on/off* do *nobreak*.

3.2.5.4 *Watchdog*

O *watchdog timer* é utilizado para evitar o travamento do programa. Se alguma rotina do programa principal congelar, o *watchdog* entra em ação para reiniciar o programa principal.

Para isso, o *watchdog* possui um *software timer* interno que é reiniciado a cada 30 segundos. Esse contador indica que o programa está funcionando corretamente. Se esse contador não for reiniciado nesse tempo a rotina do *watchdog*

interpretará que o programa travou em algum ponto e reiniciará o sistema (ESPRESSIF SYSTEMS, 2024).

3.2.5.5 *Over-The-Air* (OTA)

Over-The-Air (OTA) é uma técnica que permite programar o módulo WT32-ETH01 remotamente através de uma conexão *Wi-Fi,* LAN ou e*thernet*. Isso facilita o desenvolvimento, pois não é necessário conectar o módulo fisicamente a um computador para atualizar o *software.* Basta enviar um comando de gravação com o novo *firmware* pelo IP da placa conectado à rede local.

Essa opção também facilita a atualização quando o produto for implementado, especialmente se o kit de monitoramento estiver em um local de difícil acesso (ARDUINO E CIA, 2018). Assim, é possível acessar o sistema remotamente pelo *Linux* do *Orange Pi* e enviar a atualização, sem a necessidade de enviar um técnico para o local.

Para a implementação do OTA foi utilizada a biblioteca ArduinoOTA (ESPRESSIF SYSTEMS, 2024).

3.2.5.6 *Telnet*

Telnet é um protocolo de rede utilizado na *internet* ou em redes locais que permite ao usuário se conectar a outro computador ou dispositivo de rede de forma remota para emitir comandos diretamente através de uma sessão terminal.

Este protocolo, transmite dados em texto puro (*plain text*), o que significa que não possui criptografia, assim tornando-o suscetível a interceptações e ataques e comprometendo sua segurança (TANENBAUM, 2003).

No contexto deste projeto, a comunicação *telnet* foi utilizada para receber os *logs* de eventos da placa, assim o módulo WT32-ETH01 pode ficar desconectado do computador de desenvolvimento. Para isso, foi utilizada a biblioteca *"*TelnetStream" (JANDRASSY, 2024).

Como o objetivo do uso do *telnet* era apenas para o recebimento de logs durante o desenvolvimento, isso não comprometerá a segurança final do projeto. Visto que este protocolo será removido quando for realizada a comercialização do

produto. Assim os logs poderão ser lidos pela *Orange Pi* ou interpretados pelos dados do *dashboard*.

3.2.5.7 *Ethernet* e *Wi-Fi*

Tanto para a biblioteca do *telnet* como a do OTA, o sistema precisa estar conectado à rede. Para isso, são utilizadas as bibliotecas de *Wi-Fi* (ESPRESSIF SYSTEMS,2024) e e*thernet* (ESPRESSIF SYSTEMS,2024).

O desenvolvimento inicial foi realizado utilizando somente o *Wi-Fi*, mas eventualmente foi adicionado a biblioteca de *ethernet* devido ao fato de existirem dois *modems* diferentes e o módulo WT32-ETH01 precisaria se conectar à duas redes diferentes para identificar quando houver falta de internet em algum dos *modems*.

3.2.5.8 *HTTP Client*

Para a requisição de comandos vindos do site foi utilizada a biblioteca *HTTPClient* (ESPRESSIF SYSTEMS,2024), que envia uma requisição do tipo *GET*. A requisição *HTTP (Hypertext Transfer Protocol)* utilizando o método *GET* é um dos métodos mais comuns de solicitar dados de um servidor *web*. Neste método, os dados são passados pela *URL* da requisição como parâmetros. Ao contrário do método do tipo *POST*, a requisição *GET* não possui corpo (*body*) (RANDOM NERD TUTORIALS, 2020).

Essa requisição possui três componentes de retorno. Dentre eles, temos a linha de status, com o código e a mensagem de status, os cabeçalhos da resposta e o corpo da resposta com os dados solicitados. Dessa forma, é necessário filtrar a resposta recebida, buscando apenas a parte do corpo da resposta onde está o comando que começa com o caractere “#”.

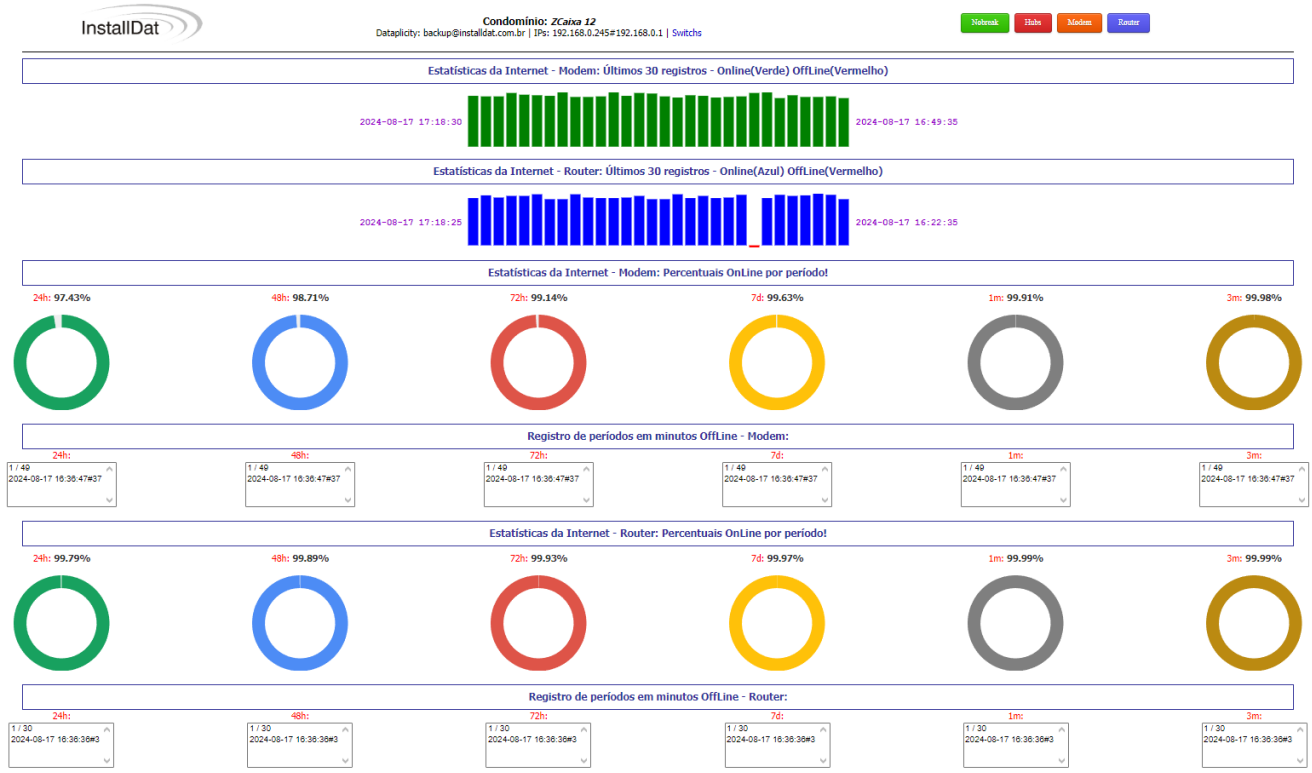
Quando essa resposta é recebida, o módulo WT32-ETH01 interpreta o comando e realiza os acionamentos necessários. Em seguida, envia outra requisição do tipo *GET* em outro endereço de *URL* do servidor para indicar que o comando foi atendido e pode, assim, ser removido da fila de comandos.

Caso as requisições tenham perdas de pacote, o sistema continuará tentando se comunicar com o servidor até que seja executado o comando e o servidor receba um comando que todos os passos foram executados de acordo.

3.2.5.9*Dashboard*

O *dashboard* do projeto foi disponibilizado pela empresa parceira e representa a interface do administrador. Nele pode-se visualizar as estatísticas do sistema e o envio de comandos pela mesma interface.

Figura 44 - Dashboard completo



Fonte: Autoria própria (2024)

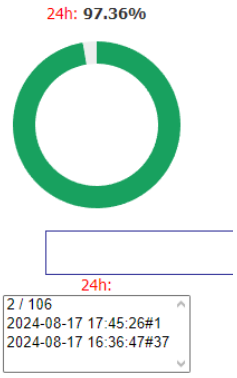
A Figura 44 demonstra o funcionamento do *dashboard*. Pode-se notar que no topo, ao lado da logo da InstallDat e das informações básicas, existem quatro botões coloridos. Nestes botões está escrito "*Nobreak*" (botão verde), "*Hubs*" (botão vermelho), "Modem" (botão laranja) e "*Router*" (botão azul). Cada um desses botões representa um comando presente no *software*. O botão "*Nobreak"* serve para o usuário enviar o comando que acionará o relé do *nobreak* para realizar a reinicialização deste *nobreak*. De forma análoga, o botão "*Hubs"* reinicializa os *hubs*, o botão "Modem" reinicializa os modems e, por fim, o botão "*Router*" reinicializa o *router*.

Logo abaixo da logo do site, existe um bloco de estatísticas de *internet* que representa as estatísticas do módulo WT32-ETH01. A cada minuto é enviada uma requisição do tipo GET por este módulo. Assim, se o módulo não executa essa requisição o gráfico no *dashboard* apresentará uma lacuna em vez de uma barra da cor verde.

Abaixo deste bloco, encontram-se as estatísticas do módulo *Orange Pi,* que funcionam de forma similar às do módulo WT32-ETH01. Porém, dessa vez o gráfico possui barras na cor azul. Como podemos visualizar na figura, existe uma lacuna com uma linha vermelha exibida na parte inferior deste bloco, esta linha significa que o módulo estava *offline* por alguns instantes devido a alguma falha de conexão.

A seguir, são apresentadas as estatísticas por período, divididas em intervalos de 24 horas, 48 horas, 72 horas, 7 dias, 1 mês e 3 meses, respectivamente. Essas estatísticas estão organizadas em dois blocos: o primeiro representa o módulo WT32-ETH01 e o segundo, o *Orange Pi*. Se algum dos módulos do sistema se encontra *offline*, é exibido um registro com a data e hora do acontecimento na seção das tabelas abaixo dos gráficos. Como podemos visualizar na Figura 45.

Figura 45 - Dashboard -
Estatísticas de um período de 24h



Fonte: Autoria própria (2024)

Por fim, quando um comando é enviado pelo usuário utilizando-se os botões, a lista de comandos é registrada na interface com a data e hora em que o comando foi enviado, como na Figura 46. Assim é possível enviar diversos comandos seguidos.

Figura 46 - Dashboard - Botões e comandos

Fonte: Autoria própria (2024)

3.2.6 Testes e validações

Quando foi terminada a fabricação da PCI, foram realizados alguns testes para verificar se todos os componentes estavam funcionando corretamente e se todas as ligações da placa estavam de acordo com o projeto. Após alguns testes de acionamento de relés, verificou-se que todos os componentes estavam funcionando como esperado e que toda a lógica que foi planejada estava correta.

Após essa fase, deu-se continuidade ao desenvolvimento, pois a placa de suporte tornou o manuseio do circuito mais simples e consistente em comparação ao protótipo inicial utilizando uma *protoboard*.

Por fim, quando o *software* estava na fase final, foi realizado o teste de estresse da placa e foram testados diversos cenários em que poderiam ocorrer erros no sistema, para verificar se o conjunto conseguiria lidar com todos esses casos de teste.

Diante disso, foram feitos os últimos ajustes e mais alguns testes para verificar se o sistema conseguiria funcionar por períodos prolongados.

4 RESULTADOS

4.1 Resultados esperados

4.1.1 Tecnológicos

Desenvolvimento de um equipamento que poderá atender às demandas do mercado para a automatização no processo de monitoramento e detecção de falhas. Podendo interagir com os equipamentos de forma autônoma ou remota para uma rápida reinicialização dos equipamentos.

4.1.2 Econômicos

Redução dos gastos com locomoção e trabalho de profissionais que realizariam uma manutenção simples. Sendo necessário apenas o envio dos profissionais quando houver um problema técnico mais grave, como troca de equipamentos queimados, que o sistema não conseguirá resolver sozinho. Além disso, o protótipo possibilitará um ganho na velocidade de atendimento ao cliente.

4.2 Análise dos resultados obtidos

Após a conclusão do desenvolvimento e dos testes, foram realizados testes de funcionalidades individuais de cada bloco de funcionalidade. Em seguida, o sistema foi colocado em funcionamento por 24 horas e a cada hora foram executados os blocos de testes das funcionalidades. Feito isso, foram detectadas pequenas correções a serem feitas a fim de deixar o sistema o mais robusto possível.

Pode-se afirmar que todos os resultados esperados foram atingidos e as funcionalidades propostas no pré-projeto foram implementadas com sucesso.

A integração dos sistemas foi cuidadosamente planejada e executada, resultando em um sistema plenamente funcional. Os testes realizados confirmaram que o circuito eletrônico opera de acordo com o especificado, cumprindo todas as funções designadas.

Além disso, alguns detalhes de construção da PCI foram pensados para que a integração com a caixa fosse facilitada, proporcionando a proteção e manuseio adequados do protótipo. A partir desses resultados obtidos, foi possível confirmar a escolha de utilizar uma caixa pronta em vez de realizar o projeto de uma.

O êxito deste projeto demonstra a importância do planejamento detalhado e do cuidado na execução das tarefas propostas em cada fase do desenvolvimento. É importante ter uma visão do escopo do projeto e planejar o fluxo de tarefas, bem como projetar fluxogramas para a realização de um código consistente. Todos esses cuidados proporcionam uma otimização de tempo e eficácia na implementação do projeto final.

4.3 Sugestões para trabalhos futuros

Embora este projeto tenha alcançado todos os objetivos propostos, sempre existirá a possibilidade de expansões e aprimoramentos bem como a adesão de novas funcionalidades, conforme a demanda do cliente. Com base na experiência adquirida ao longo desta jornada, seguem algumas sugestões para projetos futuros.

4.3.1 Desenvolvimento de uma caixa personalizada

A primeira delas seria a criação de uma caixa personalizada, assim o protótipo poderia ser personalizado de acordo com a demanda do cliente. Isso proporcionaria uma customização estética aprimorada e o projeto poderia ter melhorias de ventilação e otimização de espaço.

Uma das alternativas é investir em ferramentas de *design* 3D e ferramentas de impressão 3D. Outra opção, seria criar uma caixa de mdf para um *design* mais rústico.

4.3.2 Automatização dos testes

Apesar dos testes manuais terem sido eficientes, era necessário monitorar manualmente os resultados. Assim, foram desenvolvidos *scripts* automatizados, que executam os testes de forma autônoma por longos períodos de tempo, simulando um ambiente real.

4.3.3 Melhoria no *hardware*

Novos relés podem ser adicionados para realizar o controle de mais equipamentos. Além disso, é possível incluir pontos de fixação para a placa, otimizar a disposição das trilhas e, possivelmente, reduzir o tamanho da PCI.

4.3.4 Melhoria no *software*

Transformação do fluxo do programa de sequencial para concorrente, utilizando um sistema operacional para realizar o gerenciamento de tarefas. Com isso, o *software* passaria a permitir a execução paralela de processos, otimizando o desempenho e a eficiência no uso dos recursos. Essa abordagem permitiria uma resposta em tempo real, evitando atrasos das rotinas e uma maior flexibilidade para futuras expansões, como a adição de novos módulos.

5 CONCLUSÃO

Este projeto coloca em prática diversos conhecimentos aprendidos ao longo do curso de Bacharelado em Engenharia Eletrônica. Com ele pode-se entender como aplicar os conhecimentos aprendidos em um caso real para a criação de um produto.

Muitos desafios foram enfrentados ao longo desta jornada devido ao fato de a tecnologia estar em constante evolução. Em contrapartida, muitos conceitos novos foram aprendidos, especialmente a integração *web* do sistema. Este conceito não foi estudado a fundo durante a graduação e foi necessária a realização de alguns cursos para o entendimento do conceito. Outro desafio foi a preocupação com a segurança do circuito, realizando o projeto de vários blocos de proteção para evitar que os componentes fossem danificados, visto que existem blocos de corrente alternada e blocos de corrente contínua onde as referências GND não podem se misturar, cada um necessita de um ponto de referência específico.

Além disso, diversas versões de código foram realizados para que o código ficasse consistente com todos os requisitos propostos e testes de *stress* do sistema. Também é importante destacar o apoio e auxílio do professor-orientador Paulo da Luz, que ajudou a corrigir alguns erros, propôs algumas melhorias para o projeto e sempre esteve em constante comunicação realizando diversas reuniões de alinhamento.

Outro aspecto relevante foi o desafio da realização do *hardware*, *software* e escrita do trabalho de conclusão de curso sozinho, ou seja, sem o auxílio de nenhum colega para dividir as tarefas deste projeto. Muitas horas foram dedicadas e muito esforço para a finalização.

Em vista disso, o projeto cumpriu todos os requisitos que foram propostos e deixo aqui um agradecimento a todos os professores do curso de Engenharia Eletrônica e aos colegas que me ajudaram durante essa jornada. Chegar à conclusão do curso não foi uma tarefa fácil.

REFERÊNCIAS

ALEE, N.; RAHMAN, M.; AHMAD, R. B. **Performance comparison of single board computer: A case study of kernel on ARM architecture**. In: Proceedings of the 6th International Conference on Computer Science and Education (ICCSE), 2011, p. 521–524.

ARDUINO. **Arduino Nano.** 2024. Disponível em: <https://store-usa.arduino.cc/products/arduino-nano>. Acessado em: 15 de agosto de 2024.

ARDUINO. **Millis().** 2024. Disponível em: <https://www.arduino.cc/reference/pt/language/functions/time/millis/>. Acessado em: 16 de agosto de 2024.

ARDUINO E CIA. **Como programar o ESP32 por Wifi usando OTA.** 2018. Disponível em: <https://www.arduinoecia.com.br/programar-esp32-wifi-com-ota-over-the-air/>. Acessado em: 19 de junho de 2024.

BABIUCH, M.; FOLTỲNEK, P.; SMUTNỲ, P. **Using the esp32 microcontroller for data processing.** In: IEEE. 2019 20th International Carpathian Control Conference (ICCC). [*S.l.*], 2019. p. 3–4.

BRAVO, Renan Marcel. **Infraestrutura de monitoramento remoto por vídeo utilizando câmeras IP e um dispositivo ARM**. 2015. 53 f. Trabalho de Conclusão de Curso (Especialização) - Universidade Tecnológica Federal do Paraná, Pato Branco, 2015.

CLARO, André Luis Saganski Costa. **Tecnologias sem fio para monitoramento remoto de pacientes.** 2011. 55 f. Trabalho de Conclusão de Curso (Especialização) – Universidade Tecnológica Federal do Paraná, Curitiba, 2011.

DOERR, J. **Low-cost microcomputing: The personal computer and single-board computer revolutions.** Proceedings of the IEEE, v. 66, n. 2, p. 117–130, Feb. 1978.

ELECTRICITY & MAGNETISM. **O que é um optoacoplador?** 2023. Disponível em: <https://www.electricity-magnetism.org/pt-br/o-que-e-um-optoacoplador/>. Acessado em: 25 de junho de 2024.

ELETRONET. **Hub, switch e router: entenda as diferenças.** 2024. Disponível em: <https://www.eletronet.com/blog/hub-switch-router/>. Acessado em: 13 de fevereiro de 2024.

ESPRESSIF SYSTEMS. **ArduinoOTA.** 2024. Disponível em: <https://github.com/espressif/arduino-esp32/tree/master/libraries/ArduinoOTA>. Acessado em 13 de abril de 2024.

ESPRESSIF SYSTEMS. **Arduino core for ESP32 WiFi chip.** 2024. Disponível em: <https://github.com/espressif/arduino-esp32>. Acessado em 13 de abril de 2024.

ESPRESSIF SYSTEMS. **Ethernet.** 2024. Disponível em: <https://github.com/espressif/arduino-esp32/tree/master/libraries/Ethernet>. Acessado em 13 de abril de 2024.

ESPRESSIF SYSTEMS. **HTTPClient.** 2024. Disponível em: <https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient>. Acessado em 13 de abril de 2024.

ESPRESSIF SYSTEMS. **Miscellaneous System APIs**. 2024. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/misc_system_api.html>. Acessado em: 16 de agosto de 2024.

ESPRESSIF SYSTEMS. **Project Configuration – ESP32**. 2024. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/kconfig.html>. Acessado em: 13 de junho de 2024.

ESPRESSIF SYSTEMS. **Watchdogs – ESP32**. 2024. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/wdts.html>. Acessado em: 13 de junho de 2024.

ESPRESSIF SYSTEMS. **WiFi.** 2024. Disponível em: <https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi>. Acessado em 13 de abril de 2024.

FAYAZI, M.; COLTER, Z.; YOUBI, Z. B. -E.; BAGHERZADEH, J.; AJAYI, T.; DRESLINSKI, R. **FASCINET: A Fully Automated Single-Board Computer Generator Using Neural Networks.** IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 41, n. 12, p. 5435-5448, Dec. 2022. DOI: 10.1109/TCAD.2022.3158073.

FERNANDES, Isabela. **Portaria remota se torna opção atrativa para condomínios durante a pandemia.** G1, Sorocaba e Jundiaí, 14/07/2020. Disponível em: https://g1.globo.com/sp/sao-jose-do-rio-preto-aracatuba/mercado-imobiliario-do-interior/noticia/2020/07/14/portaria-remota-se-torna-opcao-atrativa-para-condominios-durante-a-pandemia.ghtml. Acessado em 3 de dezembro de 2022.

INTELBRAS. **Tudo sobre nobreak: o que é, vantagens, proteções e muito mais.** 2020. Disponível em: <https://blog.intelbras.com.br/tudo-sobre-nobreak/>. Acessado em: 17 de agosto de 2024.

JANDRASSY. **TelnetStream.** 2024. Disponível em: <https://github.com/JAndrassy/TelnetStream>. Acessado em 13 de abril de 2024.

KIM, M. et al. **Automated Test Generation for REST APIs: No Time to Rest Yet.** 2022. DOI: 10.1145/3533767.3534401. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.2204.08348&lang=pt-br&site=eds-live&scope=site. Acessado em 29 de novembro de 2022.

MAH, B. **An empirical model of http network traffic.** In: Proceedings of INFOCOM'97. [S.l.: s.n.], 1997.v.2, p.592–600 vol.2.

MAIER, A.; SHARP, A.; VAGAPOV, Y. **Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things.** In: 2017 Internet Technologies and Applications (ITA). [S.l.: s.n.], 2017.p.143-148

MOZILLA. **An overview of HTTP**. 2024. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>. Acessado em 8 de agosto de 2024.

MOZILLA. **TCP**. 2024. Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/TCP>. Acessado em 8 de agosto de 2024.

RANDOM NERD TUTORIALS. **ESP32 HTTP GET and HTTP POST with Arduino IDE (JSON, URL Encoded, Text).** 2020. Disponível em: <https://randomnerdtutorials.com/esp32-http-get-post-arduino/>. Acessado em: 20 de junho de 2024.

SANTOS, Ana Carolina Ribeiro dos; MORIMATSU, Igor de Souza; GUIZO, Vinicius. **Dispositivo com monitoramento remoto que libera comprimidos aos idosos.** 2019. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Tecnológica Federal do Paraná, Curitiba, 2019.

TANENBAUM, A. S. **Computer Networks**. 4th ed. Upper Saddle River: Prentice Hall, 2003. Disponível em: <https://books.google.com.br/books/about/Computer_Networks.html?id=DYQoAQAAMAAJ>.

TECHOPEDIA. **Single-Board Computer.** 2017. Disponível em: <https://www.techopedia.com/definition/9266/single-board-computer-sbc>. Acessado em: 17 de agosto de 2024.

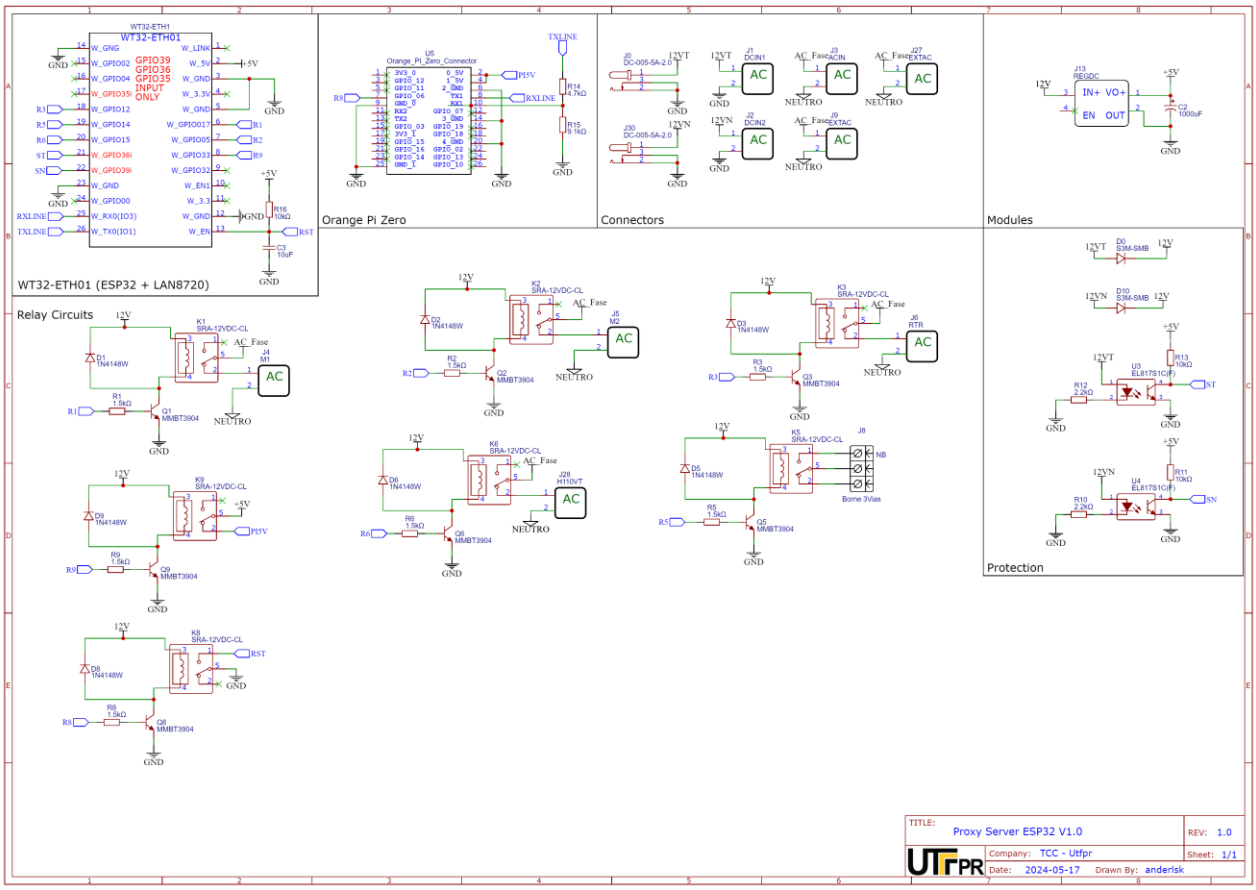
TECNOBLOG. **O que é um single board computer?** 2022. Disponível em: <https://tecnoblog.net/responde/o-que-e-um-single-board-computer/>. Acessado em: 13 de fevereiro de 2024.

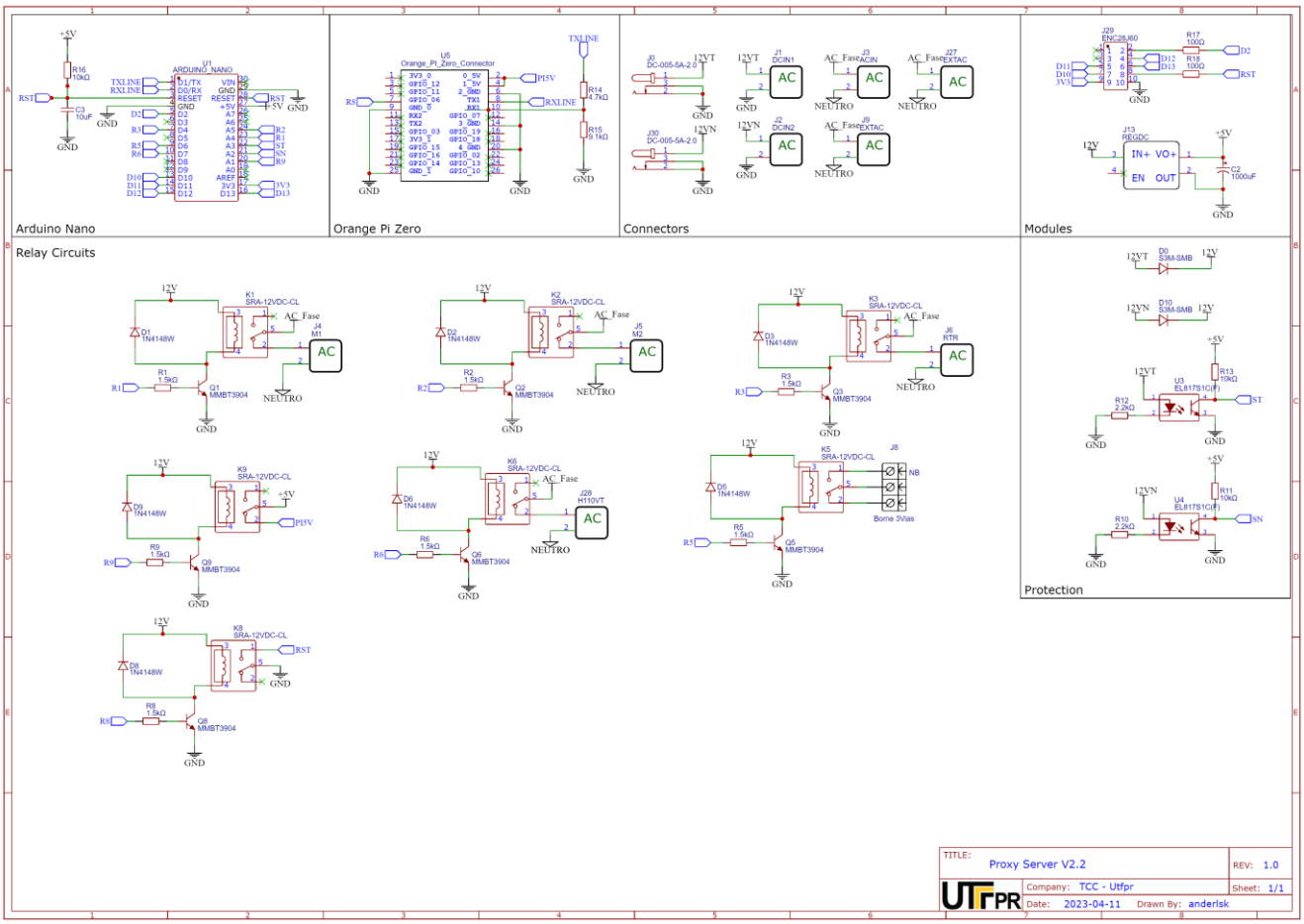
TECHTUDO. **Entenda a diferença entre hub, switch, roteador e modem.** 2013. Disponível em: <https://www.techtudo.com.br/noticias/2013/05/entenda-diferenca-entre-hub-switch-roteador-e-modem.ghtml>. Acessado em: 13 de fevereiro de 2024.

VICTOR, Patrícia Paula da Luz. **Sistema de monitoramento remoto de sinais vitais para pacientes sob supervisão médica.** 2019. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Universidade Tecnológica Federal do Paraná, Curitiba, 2019.

WIRELESS-TAG TECHNOLOGY CO. LTD. **WT32-ETH01 – Datasheet Version 1.1.** 2019. Disponível em: <https://en.wireless-tag.com/product-item-2.html>. Acessado em 22 de junho de 2023.

ANEXO A – Esquemático Completo


WT32-ETH01 (ESP32 + LAN8720)
Orange Pi Zero
Connectors
Modules
Relay Circuits
Protection
TITLE: Proxy Server ESP32 V1.0
REV: 1.0
Company: TCC - Utfpr
Sheet: 1/1
Date: 2024-05-17
Drawn By: anderlsk


Arduino Nano
Orange Pi Zero
Connectors
Modules
Relay Circuits
Protection
TITLE: Proxy Server V2.2
REV: 1.0
Company: TCC - Utfpr
Sheet: 1/1
Date: 2023-04-11
Drawn By: anderlsk

ANEXO B – Exemplo de *firmware* OTA

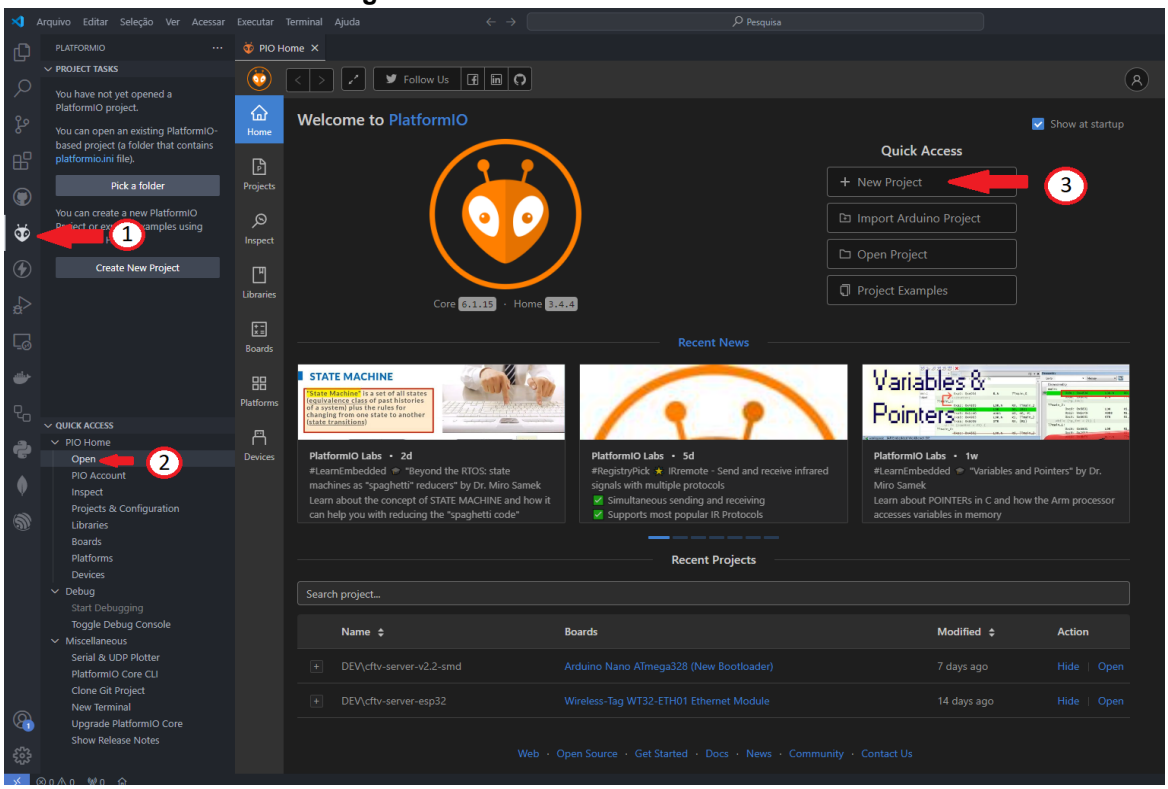
Esta seção abordará a gravação detalhada do *firmware* para a ativação da gravação *Over-The-Air* (OTA).

Primeiramente é necessária a instalação inicial do Visual Studio Code (VSCode) e da extensão PlataformIO, como abordado na seção 3.2.4.

Para a criação do projeto base, basta selecionar a extensão do PlatformIO no VSCode, em seguida, clicar em "*Open*" na seção "*PIO Home*".

Esta é a página inicial do PlatformIO. Para criar um projeto basta clicar em "*New Project*". Todos esses passos estão listados na FiguraB 1 com números em ordem crescente representando a ordem dos cliques.

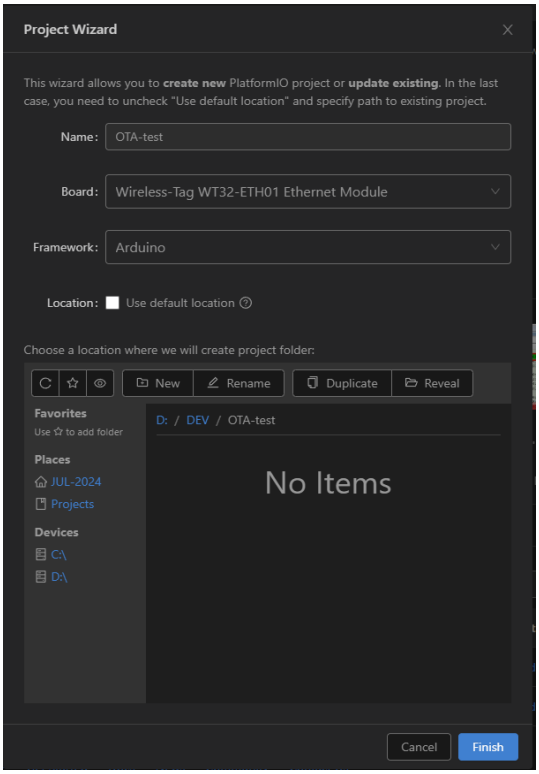
FiguraB 1 - Passos VSCode PlatformIO



Fonte: Autoria própria (2024)

Em seguida, basta preencher as informações básicas do projeto e escolher onde ele será salvo. Como é demonstrado na FiguraB 2.

FiguraB 2 - Configurações de um novo projeto

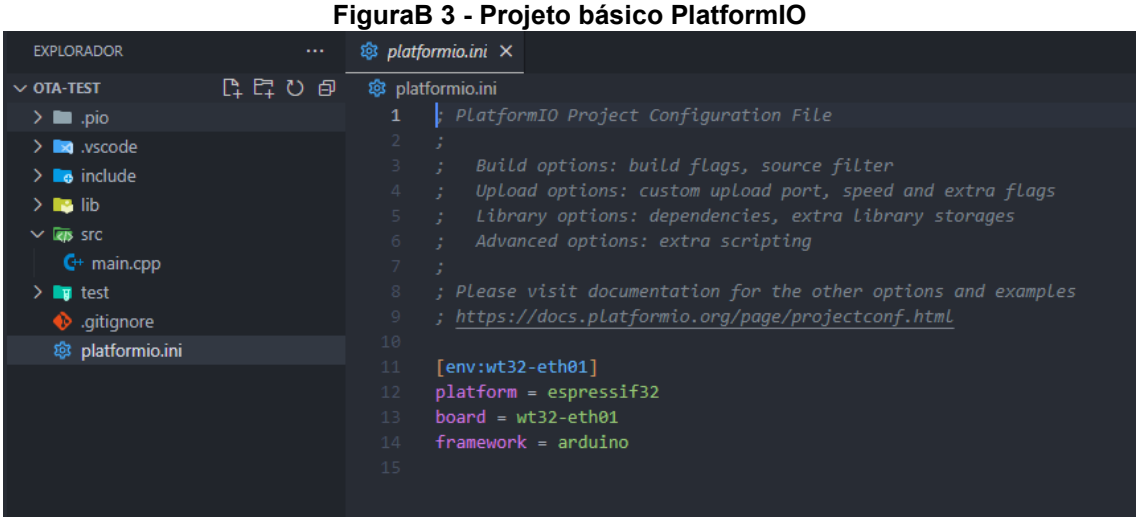


Fonte: Autoria própria (2024)

Como esse projeto foi realizado utilizando o módulo WT32-ETH01 foi escolhida essa opção no campo “*Board*” e como o *framework* utilizado para esse exemplo foi o “Arduino”, essa informação foi colocada no campo “*Framework”*.

Após o preenchimento de todas essas informações o projeto será criado utilizado o padrão de projetos do PlatformIO e virá com algumas informações básicas nas pastas e no arquivo “platformio.ini”. Assim como na FiguraB 3.

FiguraB 3 - Projeto básico PlatformIO



Fonte: Autoria própria (2024)

Para esse exemplo serão feitas algumas mudanças nesse arquivo “platformio.ini”. Pois, para a compilação do código é necessário que todas as dependências estejam atualizadas. A extensão PlatformIO realiza o *download* automático de todas as dependências listadas neste arquivo “platformio.ini”. Basta adicionar as seguintes informações da FiguraB 4.

FiguraB 4 - configurações platformio.ini

Fonte: Autoria própria (2024)

Neste exemplo, será utilizado o seguinte código:

```
#include <Arduino.h>
#include <secrets.h>
```

```
#include <ArduinoOTA.h>
#include <WiFi.h>
#include <TelnetStream.h>

#define LED 14
#define LEDTIME 500

const char ssid[] = SECRET_SSID;
const char pass[] = SECRET_PASS;

void setup() {
Serial.begin(115200);

 //WIFI
 Serial.print("Attempting to connect to WPA SSID: ");
 Serial.println(ssid);
 WiFi.begin(ssid, pass);
 // Wait for WiFi connection
 if (WiFi.waitForConnectResult() != WL_CONNECTED) {
  Serial.println("Failed to connect.");
  while (1) {
    delay(10);
  }
 }

 IPAddress ip = WiFi.localIP();
 Serial.println();
 Serial.println("Connected to WiFi network.");
 Serial.print("Connect with Telnet client to ");
 Serial.println(ip);

 TelnetStream.begin();
```

```
  //OTA
  ArduinoOTA.setHostname("ESP32-OTA");
  ArduinoOTA.begin();

  // LED Setup
  pinMode(LED,OUTPUT);
  digitalWrite(LED, LOW);
}

void loop() {
  //OTA Handler
  ArduinoOTA.handle();

  switch (TelnetStream.read()) {
    case 'R':
    TelnetStream.stop();
    delay(100);
    ESP.restart();
      break;
    case 'C':
      TelnetStream.println("bye bye");
      TelnetStream.stop();
      break;
  }

  //Simple Test
  TelnetStream.println("LED ON");
  digitalWrite(LED, HIGH);
  delay(LEDTIME);
  TelnetStream.println("LED OFF");
  TelnetStream.println("--------");
  digitalWrite(LED,LOW);
  delay(LEDTIME);
```

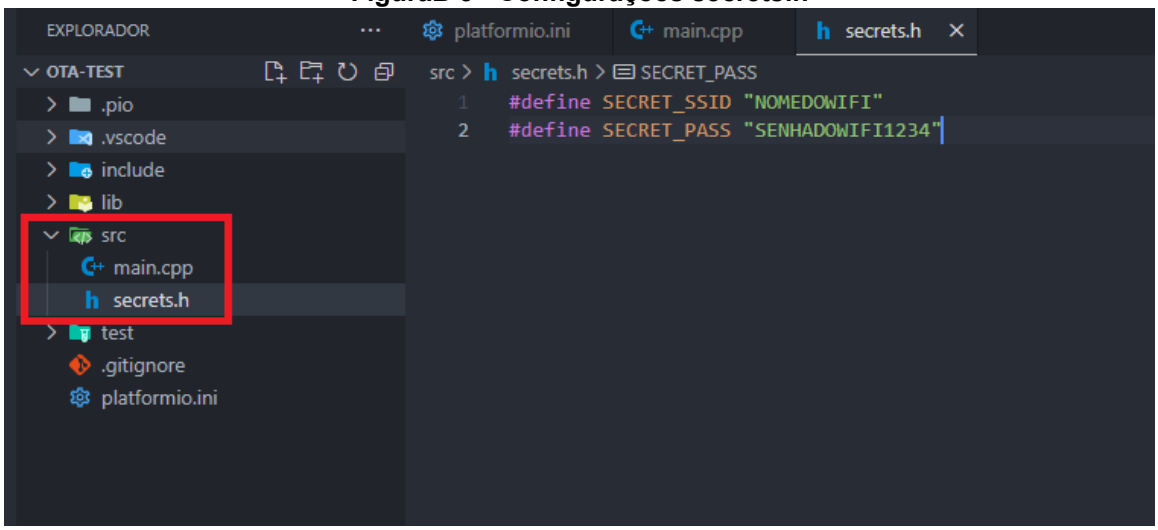
}

Para que o *Wi-Fi* funcione de forma correta, é necessário criar um arquivo “secrets.h” na mesma pasta do código principal “main.cpp” com os seguintes campos:

```
#define SECRET_SSID "nome_da_rede"
#define SECRET_PASS "senha_da_rede"
```

Em seguida, basta trocar “nome_da_rede” pelo nome da rede desejada, lembrando de manter as aspas e “senha_da_rede” pela senha da rede na qual está tentando conectar. Estes passos podem ser visualizados na FiguraB 5.

FiguraB 5 - Configurações secrets.h



Fonte: Autoria própria (2024)

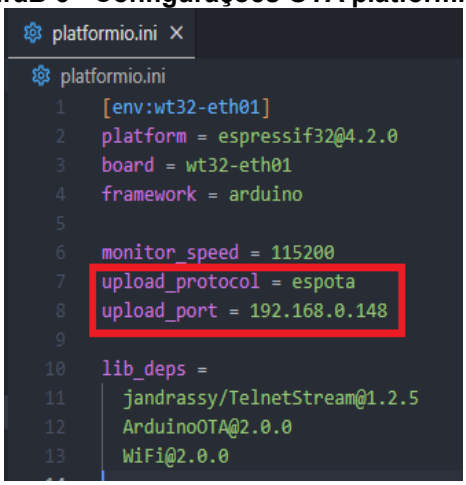
Para o funcionamento adequado do OTA, é necessário que a primeira gravação do *software* seja feita utilizando-se o CP2102, como explicado anteriormente na seção 3.2.4.1.

Após essa primeira gravação é necessário garantir que os trechos de código básicos para a configuração do OTA estarão presentes em todas as atualizações futuras de *firmware*, caso contrário, os passos iniciais do OTA terão que ser executados novamente.

Para iniciar a gravação do OTA utilizando o PlatformIO basta adicionar dois campos no arquivo “platformio.ini”: os campos “upload_protocol” como “espota” e

“upload_port” com o valor do IP que foi atribuído na rede local para o módulo WT32-ETH01. Assim como é demonstrado na FiguraB 6.

FiguraB 6 - Configurações OTA platformio.ini



Fonte: Autoria própria (2024)

Para encontrar o valor de IP, basta acessar o modem local e descobrir qual o número de IP atribuído ao dispositivo WT32-ETH01.

Por fim, para que a gravação OTA funcione é necessário garantir que os trechos de código relacionados ao OTA estejam presentes e que haja uma conexão do tipo *Wi-Fi* ou *ethernet*. Os demais trechos do código exemplo podem ser modificados.