UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ALFREDO SAVI CIUCCIO

INTERFACE DE REDE SEM FIO UTILIZANDO ESP32 COMO COPROCESSADOR

PATO BRANCO

2023

ALFREDO SAVI CIUCCIO

INTERFACE DE REDE SEM FIO UTILIZANDO ESP32 COMO COPROCESSADOR

WIRELESS NETWORK INTERFACE USING ESP32 AS COPROCESSOR

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Gustavo Weber Denardin

PATO BRANCO

2023

ALFREDO SAVI CIUCCIO

INTERFACE DE REDE SEM FIO UTILIZANDO ESP32 COMO COPROCESSADOR

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 01/Dezembro/2023

Adriano Serckumecka
Mestrado em Computação Aplicada
Universidade Tecnológica Federal do Paraná

Diogo Ribeiro Vargas
Doutorado em Engenharia Elétrica
Universidade Federal de Santa Maria

Gustavo Weber Denardin
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2023

RESUMO

A adoção frequente de interfaces de rede *Ethernet* em sistemas embarcados microcontrolados é uma prática amplamente difundida, exemplificada pelo projeto freePMU, especializado na medição de grandezas elétricas em sistemas de potência, utilizando exclusivamente *hardware* com conectividade *Ethernet*. Contudo, essa característica pode se transformar em um desafio, dependendo dos objetivos da aplicação, uma vez que requer uma conexão física com a rede para operar. Diante desse cenário, esse trabalho propõe uma solução que utiliza *System-on-Chips* (SoCs) da fabricante *Espressif* como interfaces de rede sem fio para um dispositivo *host* desprovido dessa funcionalidade sem fio. Essa proposta incorpora as bibliotecas ESP-Hosted e lwIP como elementos fundamentais da solução, visando superar as limitações associadas à dependência da conectividade física, proporcionando maior flexibilidade e mobilidade ao sistema embarcado. Também é importante destacar que essa abordagem é aplicável a uma ampla variedade de contextos que necessitam de conectividade sem fio em microcontroladores, indo além do escopo específico do projeto freePMU.

Palavras-chave: interface de rede sem fio; esp-hosted; lwip; esp32; sistemas embarcados.

ABSTRACT

The frequent adoption of Ethernet network interfaces in microcontrolled embedded systems is a widely practiced approach, as exemplified by the freePMU project, specialized in measuring electrical quantities in power systems, using exclusively hardware with Ethernet connectivity. However, this characteristic can pose a challenge, depending on the application goals, as it requires a physical connection to the network to operate. In this scenario, this study proposes a solution that utilizes System-on-Chips (SoCs) from the manufacturer Espressif as wireless network interfaces for a host device lacking this wireless functionality. This proposal incorporates the ESP-Hosted and lwIP libraries as fundamental elements of the solution, aiming to overcome limitations associated with the dependence on physical connectivity, providing greater flexibility and mobility to the embedded system. It is also noteworthy that this approach is applicable to a wide range of contexts requiring wireless connectivity in microcontrollers, extending beyond the specific scope of the freePMU project.

Keywords: wireless network interface; esp-hosted; lwip; esp32; embedded systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ADC	*Analog-to-Digital Converter*
AP	*Access Point*
API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
AT	*Attention*
BLE	*Bluetooth Low Energy*
BSSID	*Basic Service Set Identifier*
BT	*Bluetooth*
CGI	*Common Gateway Interface*
CPHA	*Clock Phase*
CPOL	*Clock Polarity*
CPU	*Central Processing Unit*
DFT	*Discrete Fourier transform*
DHCP	*Dynamic Host Configuration Protocol*
DMA	*Direct Memory Access*
DS	*Data Ready*
DSI	*Display Serial Interface*
ESP-IDF	*Espressif IoT Development Framework*
GPIO	*General Purpose Input/Output*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
HCI	*Human-Computer Interaction*

HS	*Handshake*
HSE	*High Speed External clock*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IP	*Internet Protocol*
LCD	*Liquid Crystal Display*
lwIP	*Lightweight IP*
MAC	*Media Access Control*
MCU	*Microcontroller Unit*
MISO	*Master In-Slave Out*
MIT	*Massachusetts Institute of Technology*
MOSI	*Master Out-Slave In*
NETIF	*Network Interface*
PDC	*Phasor Data Concentrator*
PMU	*Phasor Measurement Unit*
PPS	*Pulse Per Second*
RAM	*Random Access Memory*
RMII	*Reduced Media Independent Interface*
ROM	*Read-Only Memory*
RTOS	*Real Time Operating Systems*
RTT	*Round Trip Time*
RX	*Receive*
SCADA	*Supervisory Control and Data Acquisition*
SCLK	*Serial Clock*
SDIO	*Secure Digital Input/Output*

SEP	Sistema Elétrico de Potência
SMFS	Sistema de Medição Fasorial Sincronizada
SoC	*System on a Chip*
SPI	*Serial Peripheral Interface*
SS	*Slave Select*
SSH	*Secure Socket Shell*
SSI	*Server Side Includes*
SSID	*Service Set Identifier*
STA	*Station*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TLS	*Transport Layer Security*
TX	*Transmit*
UART	*Universal Asynchronous Receiver/Transmitter*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
UTFPR	Universidade Tecnológica Federal do Paraná
WEB	*World Wide Web*
WIFI	*Wireless Fidelity*
WLAN	*Wireless Local Area Network*
WPA3	*Wi-Fi Protected Access 3*

SUMÁRIO

1 INTRODUÇÃO

O Sistema Elétrico de Potência (SEP) é formado por um conjunto de usinas geradoras, linhas de alta tensão de transmissão de energia e sistemas de distribuição (Zanetta, 2006). Seu principal objetivo é gerar, transmitir e entregar energia elétrica aos consumidores para atender a demanda solicitada segundo critérios de qualidade, confiabilidade, dentre outros. Com o uso cada vez mais intenso desses sistemas, somado com diversos outros fatores, tais como a reestruturação do setor elétrico e as restrições de ordem técnica e ambiental os sistemas de energia elétrica tem enfrentado situações de operação críticas (Andrade, 2008).

Diante disso, o conceito de redes inteligentes *smart grids* surge como solução. A *smart grid* baseia-se na utilização de tecnologias como: automação, computação e comunicação para monitorar e controlar a rede elétrica, permitindo um gerenciamento muito mais eficiente e eficaz em tempo real, diferente do sistema *Supervisory Control and Data Acquisition* (SCADA) (Falcao *et al.*, 2010).

O dispositivo capaz de realizar medições na rede elétrica em tempo real é chamada *Phasor Measurement Unit* (PMU). É um dispositivo de medição dos valores de tensão e corrente, tendo uma referência única e temporal baseada nos sinais obtidos via satélite. Por utilizarem uma fonte eficaz de sincronização *Pulse Per Second* (PPS), fornecida pelo sistema de *Global Positioning System* (GPS), as PMUs viabilizam a realização da medição de grandezas fasoriais em instalações geograficamente distantes, com taxas de amostragem superiores aos sistemas SCADA tradicionais. Essas medidas são processadas e convertidas em fasores, com informações de magnitude, fase e frequência pela PMU e assim enviadas ao *Phasor Data Concentrator* (PDC) (Wohlfahrt, 2019).

O projeto de unidade de medição fasorial, chamada *FreePMU* (FreePMU, 2018), desenvolvido por alunos de graduação e mestrado da Universidade Tecnológica Federal do Paraná (UTFPR) campus Pato Branco, teve como objetivo o desenvolvimento de PMUs com uma arquitetura de baixo custo a nível de distribuição, em virtude dos elevados custos das PMUs comerciais. Contudo, um dos principais problemas encontrados no projeto foi a falta de uma interface de rede sem fio para o envio das medições feitas pela PMU.

Nesse projeto, utiliza-se um microcontrolador STM32 que possui somente suporte a uma interface de rede *ethernet*. Isso é um fator limitante, visto que em vários dos locais em que é desejada a instalação de uma PMU no sistema de distribuição não há conexão cabeada. Portanto, a elaboração de uma interface de rede sem fio para estender as funcionalidades da *FreePMU* é uma das principais melhorias possíveis para o projeto. No entanto, os *kits* que possuem essa interface *Wireless Fidelity* (WIFI) pronta não tem periféricos e capacidade de processamento necessário comparado com o STM32. Por isso a necessidade de criar uma ponte de interface de rede entre a PMU com algum outro microcontrolador que possui WIFI (por exemplo, o módulo ESP32).

Assim, o presente trabalho propôs a implementação do *firmware* necessário para que a *FreePMU* utilizasse um módulo microcontrolador externo possuindo uma interface de rede sem fio, capaz de suportar conexões *Transmission Control Protocol/Internet Protocol* (TCP/IP) com fluxo de dados compatível com o de uma PMU.

1.1 OBJETIVO GERAL

Esse trabalho tem como objetivo geral estabelecer o processo de comunicação entre um módulo microcontrolado com suporte a WIFI com outros microcontroladores, de forma a disponibilizar uma interface de rede sem fio para um microcontroladores sem suporte a esse periférico.

1.2 OBJETIVOS ESPECÍFICOS

- Utilizar a biblioteca ESP-Hosted, fornecida pela *Espressif Systems*, fabricante do ESP32, para integrar o módulo ESP32 como um coprocessador WIFI em um dispositivo *host*;
- Implementar uma *Application Programming Interface* (API) que permita configurar e utilizar os recursos de comunicação sem fio provenientes do módulo ESP32;
- Desenvolver um servidor *Hypertext Transfer Protocol* (HTTP) no modo *Access Point* (AP) da rede WIFI, com a finalidade de gerenciar os parâmetros de conectividade da aplicação;
- Conduzir testes de largura de banda da solução selecionada, com o intuito de determinar a taxa máxima de transferência de dados que pode ser alcançada.

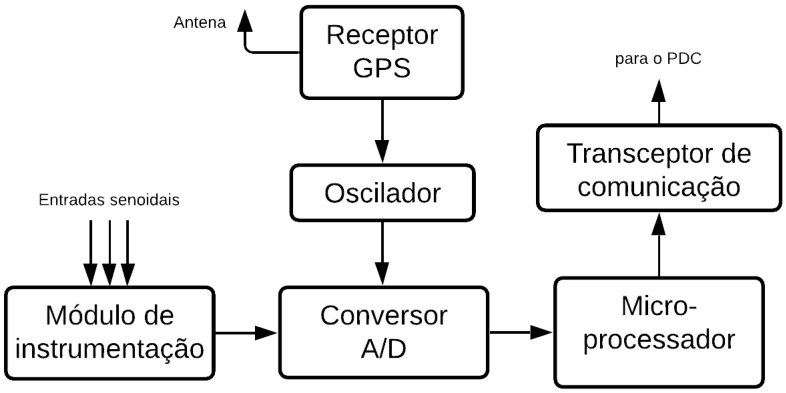
2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos teóricos essenciais para o entendimento deste projeto. Na Seção 2.1 é abordado o funcionamento de uma PMU, detalhes do protocolo de comunicação a ser empregado na interação entre o *host* e seu dispositivo *slave* na Seção 2.2, uma explicação abrangente sobre o conceito e a operação de uma API, fundamental para a interface entre esses dispositivos na Seção 2.3. Além disso, são fornecidas informações sobre os comandos AT, bastante utilizados na configuração de dispositivos na Seção 2.4. Uma exposição abrangente acerca da solução denominada "ESP-Hosted", que estabelece uma interface de comunicação entre um dispositivo desprovido de capacidade de rede sem fio e outro dispositivo que a detém, na Seção 2.5, bem como uma exposição sobre os sistemas operacionais em tempo real, necessários para o desenvolvimento do *firmware* na Seção 2.6. Por fim, na Seção 2.7 é discutida a pilha de protocolos TCP/IP que será utilizada no projeto.

2.1 PMU

Atualmente a mudança da rede elétrica convencional para uma rede elétrica inteligente acontece pela proteção e o controle de monitoramento em tempo real do sistema de medição de uma área ampla e as PMUs são requisitos básicos dessa rede elétrica inteligente (Joshi; Verma, 2021). Segundo Jamil, Rihan e Anees (2014), as PMUs são dispositivos de medição digitais que utilizam um algoritmo de *Discrete Fourier transform* (DFT) juntamente com um sinal de GPS para sincronizar todas as PMUs conectadas em rede Sistema de Medição Fasorial Sincronizada (SMFS), garantindo que todas registrem os fasores no mesmo instante de tempo assegurando uma melhor análise da rede elétrica.

Figura 1 – Estrutura genérica de uma PMU



Fonte: Adaptado de Jamil, Rihan e Anees (2014).

A Figura 1 mostra o diagrama de blocos genérico de uma PMU. O processo operacional da PMU tem início com a recepção do sinal de pulso do GPS, marcando o início de um segundo.

O propósito desse pulso é sincronizar as medições de PMUs distribuídas geograficamente. Ao receber esse sinal, a PMU inicia a amostragem do *Analog-to-Digital Converter* (ADC) e ativa um contador, cuja responsabilidade é estabelecer o momento exato para a aquisição dos próximos períodos de amostragem. É relevante ressaltar que uma PMU possui a flexibilidade de amostrar de 1 a 120 fasores por segundo.

Após a conclusão da amostragem de um período completo, a PMU calcula os fasores e, posteriormente, são encaminhados para a interface de rede, a qual possui a capacidade de transmitir esses dados para o PDC, contribuindo assim para a integração e monitoramento preciso do sistema elétrico.

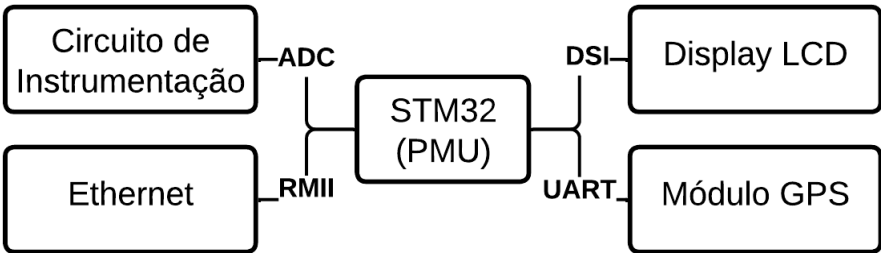
Conforme mencionado, após a realização dos cálculos de fasores, a PMU necessita enviar esses dados para o PDC utilizando um canal de comunicação. Com base nisso, foi lançada a norma IEEE C37.118.2 com o intuito de padronizar essa comunicação entre dispositivos.

A norma IEEE C38.118.2 trata exclusivamente da comunicação em tempo real entre PMU, PDC e outras aplicações, especificando o tipo e formato dos dados a serem recebidos e enviados entre os dispositivos.

2.1.1 Projeto *freePMU*

O projeto da *FreePMU* tem como base o hardware STM32F769NI *discovery kit* e integra módulos externos, como GPS e um circuito de instrumentação, com o propósito de constituir uma PMU de baixo custo. O processo operacional envolve a recepção do sinal de PPS emitido pelo GPS. Posteriormente, a PMU utiliza seu circuito de instrumentação, que emprega três conversores ADC independentes, para adquirir e calcular os fasores, permitindo a monitorização da rede elétrica. Após a conclusão dos cálculos, o microcontrolador STM32 apresenta os valores resultantes no *display Liquid Crystal Display* (LCD) por meio de uma interface *Display Serial Interface* (DSI) e, simultaneamente, envia os pacotes de dados através da interface *Reduced Media Independent Interface* (RMII) para o módulo *ethernet* (Grando, 2016). A Figura 2 ilustra o diagrama de blocos da *FreePMU*, exibindo todos os módulos interconectados.

Figura 2 – Diagrama de bloco geral - *FreePMU*



Fonte: Autoria própria (2023).

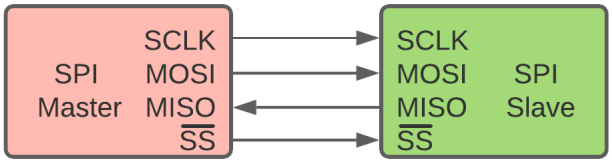
Na Figura 2, observa-se que o projeto *free*PMU emprega diversos protocolos de comunicação, resultando na ocupação de vários pinos do microcontrolador. Devido a essa alocação,

a única interface serial disponível para conectar um dispositivo de rede sem fio é através do barramento *Serial Peripheral Interface* (SPI).

2.2 Protocolo SPI

A interface de barramento SPI é amplamente utilizado para transmitir dados entre dois ou mais dispositivos digitais e de forma síncrona, pois utiliza um sinal de *clock* responsável por sincronizar todos os dispositivos conectados. SPI é um protocolo de comunicação que possui quatro linhas de sinal (Figura 3) e também de mestre único. Isso significa que um dispositivo central, chamado mestre, inicia todas as comunicações com os demais que são denominados escravos (Leens, 2009).

Figura 3 – Mestre conectado a um único escravo (Topologia ponto a ponto)



Fonte: Autoria própria (2023).

O barramento SPI funciona conforme a arquitetura mestre-escravo, em que o mestre é responsável por controlar todas as comunicações efetuadas com seus escravos, e a troca de dados acontece bidirecionalmente, isso é, o SPI permite que os dados sejam transmitidos e recebidos simultaneamente, pois cada *bit* de dado enviado do mestre para o escravo, transfere também um *bit* de dado do escravo para o mestre. Dessa forma, define-se que a comunicação pode ser *full-duplex* (Sacco, 2014).

Abaixo, a explicação dos quatro sinais utilizados no protocolo SPI:

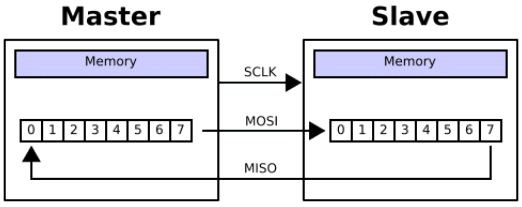
- **Sinal de *Clock* (SCLK):** O sinal é enviado do mestre no barramento *Serial Clock* (SCLK) para todos os escravos. Todos os sinais SPI são síncronos com esse sinal de *clock*;
- **Seleção do escravo (SSn):** Como o SPI utiliza barramento, é necessário selecionar qual escravo o mestre quer comunicar, chamado *Slave Select* (SS);
- **Sinal MOSI:** Linha de dados do mestre para os escravos, chamado *Master Out-Slave In* (MOSI);
- **Sinal MISO:** Linha de dados dos escravos para o mestre, chamado *Master In-Slave Out* (MISO).

2.2.1 Transmissão de dados

Para enviar e/ou solicitar informações dos escravos, o mestre deve ativar o sinal de *clock*, com uma frequência em que ambos dispositivos suportem, e selecionar o escravo habilitando o sinal SS. Normalmente, esse sinal é ativado em nível lógico baixo (*LOW*). Durante a comunicação, os dados são transmitidos simultaneamente (deslocados em série no barramento MOSI) e recebidos no barramento MISO (como mostra a Figura 4). O sinal de *clock* é responsável por sincronizar a mudança e a amostragem dos dados. A mudança pode ser configurada tanto na borda de subida ou descida do sinal de *clock* (Dhaker, 2018).

Como o mestre é responsável por selecionar o sinal SS, o dispositivo escravo só consegue transmitir dados se houver uma solicitação do mestre. Portanto, o dispositivo escravo deve empregar um mecanismo alternativo para notificar o mestre sobre a disponibilidade de dados para leitura. Um exemplo viável seria a utilização de um pino de interrupção.

Figura 4 – Comunicação SPI



Fonte: (Sacco, 2014).

2.2.2 Polaridade e fase do sinal de *clock*

O protocolo SPI permite a configuração das bordas de comunicação do *clock* de acordo com sua polaridade *Clock Polarity* (CPOL) e fase *Clock Phase* (CPHA). Seus modos possíveis são mostrados na Tabela 1.

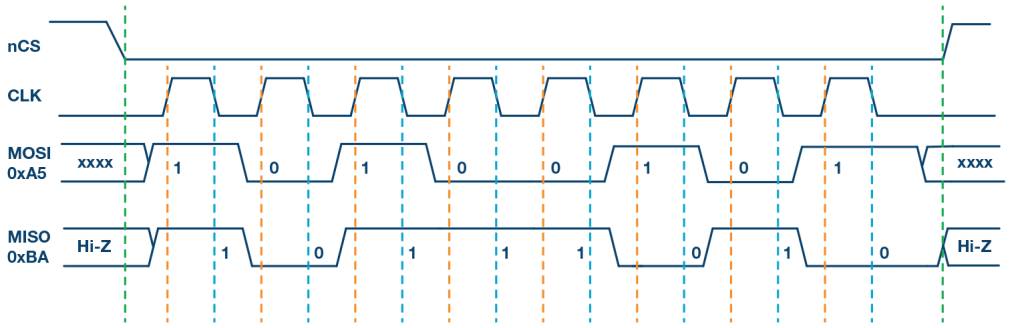
Tabela 1 – Modos de configuração do *Clock*

Modo	CPOL	CPHA	Borda de troca
0	0	0	Subida
1	0	1	Descida
2	1	0	Descida
3	1	1	Subida

Fonte: (Dhaker, 2018).

A Figura 5 mostra um exemplo de comunicação SPI no modo 0 (CPOL e CPHA iguais a zero). No exemplo, o início e o fim da transmissão são indicados pela linha verde pontilhada, os dados são apresentados nas linhas MOSI e MISO, a borda de amostragem é indicada em laranja e a borda de deslocamento indicado pela linha em azul (Dhaker, 2018).

Figura 5 – Exemplo de comunicação SPI - Modo SPI 0



Fonte: (Dhaker, 2018).

No contexto da Figura 5, o início da transmissão é evidenciado pelo sinal "nCS", também conhecido como sinal SS. Neste ponto, o mestre estabelece um nível lógico baixo para esse sinal, selecionando assim o dispositivo escravo e dando continuidade à transmissão de dados. Nessa configuração, os dados são lidos na borda de subida e deslocados na borda de descida do sinal de *clock*.

2.3 API

Uma API define um conjunto de regras definidas que devem ser genéricas e independentes permitindo computadores ou aplicativos se comunicarem-se entre si, aproveitando os dados e funcionalidades de um aplicativo para outro por uma interface sem precisar saber como eles foram implementados (Ibm, 2020).

A utilização de APIs geram uma economia de tempo e dinheiro, visto que ela simplifica o desenvolvimento das aplicações ao reutilizar funcionalidades que a API provê. Também, podem disponibilizar funcionalidades de difícil implementação para o usuário final sem ele precisar entender como funciona e como foi implementado, precisando somente de um guia de referência de como utilizar essas funcionalidades, chamada documentação (RedHat, 2017).

Um exemplo de API com grande dimensão, são os serviços para integração de soluções de pagamento *online*. Com o intuído de não precisar implementar um serviço de pagamento, podendo ser de grande complexidade, a empresa opta por utilizar uma API para pagamentos *online*, ficando responsável somente pela integração da API na aplicação.

Certamente, a API utilizada pelo exemplo é classificada como remota, ou seja, ela não se encontra no mesmo computador que foi efetuada a solicitação do serviço, utilizando a rede de comunicações para realizar essa comunicação entre esses aplicativos.

Portanto, as APIs desempenham papéis diversos. Por exemplo, enquanto a API gráfica opera localmente, manipulando dados diretamente em uma tela na própria plataforma, a API de comandos AT, se destaca por solicitar a execução de funcionalidades em dispositivos externos. Essa distinção ressalta a versatilidade das APIs, que não se limitam a operações internas, mas também facilitam a comunicação e a execução de tarefas em outros dispositivos.

2.4 AT *commands*

De acordo com Agnihotri (2021), os comandos AT, originalmente herdados dos comandos *Hayes*, foram inicialmente projetados para a comunicação entre modems inteligentes e computadores. Esses comandos consistem em uma série de *strings* de texto com instruções pré-definidas, permitindo o controle eficiente do *hardware*. Por exemplo, são capazes de executar ações como atender ou desligar uma chamada em dispositivos *Global System for Mobile Communications* (GSM). Os comandos AT são reconhecíveis pelo início com a instrução AT indicando "atenção". Esses comandos ganharam uma popularidade considerável na interface de comunicação entre processadores/microcontroladores e modems (modulador/demodulador). Essa popularidade se estendeu abrangendo diversas interfaces de rede sem fio. Recentemente, os comandos AT têm sido empregados de maneira extensiva para facilitar a comunicação eficiente entre dispositivos e interfaces sem fio, como *Zigbee*, WIFI, entre outras.

Os comandos AT podem ser classificados em quatro tipos, conforme apresentado na Tabela 2.

Tabela 2 – Classificação dos comandos AT

Classificação	Função	Sintaxe
Test	Verifica se um comando é suportado pelo *hardware*	AT: <nome do comando>=?
Read	Obtém informações das configurações do *hardware*	AT: <nome do comando>?
Set	Modifica configurações do *hardware*	AT: <nome do comando>= valor1, ..., valorN
Execution	Executa uma operação	AT: <nome do comando>= parâmetro1, ..., parâmetroN

Fonte: Adaptado de Agnihotri (2021).

A sintaxe de uma linha de comando é composta por três elementos:

1. **Prefixo**: Consiste nos caracteres “AT” ou “at” para o dispositivo identificar que a mensagem é do tipo *command* AT;

2. **Corpo**: Consiste nos comandos AT;

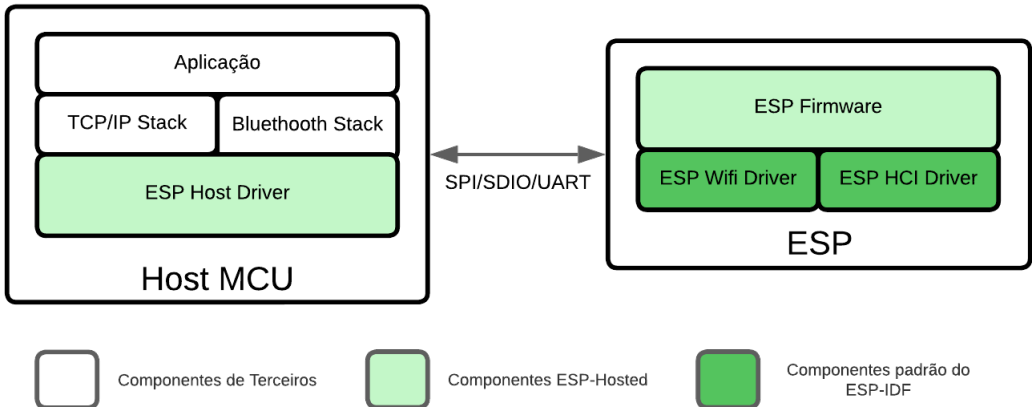
3. **Carácter de término**: Indica o final do comando, podendo ser selecionado pelo usuário. O valor padrão é <CR> (*Carriage Return*) que move o cursor para a linha seguinte na tela.

Os comandos AT demostram que todo o processo de comunicação ocorre no dispositivo remoto, sendo controlado por esses comandos. A implementação dos protocolos de comunicação acontece no modem, enquanto a plataforma *host* desempenha o papel exclusivo de fornecer ou receber os dados.

2.5 Biblioteca Esp-Hosted

De acordo com Esp-Hosted (2023), é uma solução de código aberto, fornecido pela empresa *Espressif Systems*, que fornece uma solução para a utilização de *System on a Chip* (SoC)s e módulos *Espressif* como coprocessador de comunicação. Essa tecnologia possibilita conectividade sem fio (WIFI, *Bluetooth* (BT) e *Bluetooth Low Energy* (BLE)) para microprocessador ou microcontrolador *host*. A Figura 6 exibe um resumo do diagrama de blocos do projeto.

Figura 6 – Diagrama em bloco do ESP-Hosted



Fonte: Adaptado de Esp-Hosted (2023).

Resumidamente, caso uma aplicação necessite de uma interface sem fio, essa solução pode ser empregada para estabelecer uma conexão entre seu dispositivo (denominado dispositivo *host*) e um módulo ou SoC da fabricante *Espressif*, proporcionando, desse modo, acesso à conectividade sem fio. Vale destacar que, em comparação com os comandos AT, a solução ESP-Hosted apresenta uma abordagem distinta, em que o dispositivo remoto com suporte WIFI opera como uma interface de rede para o dispositivo *host* e toda a lógica de comunicação e protocolos de rede torna-se responsabilidade do dispositivo *host*.

Essa solução é oferecida em duas variantes, sendo que, a distinção principal reside no tipo de interface de rede disponibilizada para o dispositivo *host* e também na maneira como o WIFI no SoC/módulo ESP é configurado e gerenciado.

- **Esp-Hosted-NG:** Projetada especificamente para *hosts* que operam no sistema operacional Linux;
- **Esp-Hosted-FG:** Solução de primeira geração (*first generation*) projetada para fornecer uma interface de rede padrão 802.3 (*ethernet*) para o dispositivo *host*.

Nesse projeto, daremos destaque à variante FG, uma vez que o trabalho está centrado em um *Microcontroller Unit* (MCU).

2.5.1 Introdução ao Esp-Hosted-FG

De acordo com a documentação oficial Esp-Hosted (2023) *release* 0.0.5, essa solução tem como foco manter o *software* simples no *host* e fornecer um conjunto de recursos de conectividade, ideal para uso com *hosts* MCU que não possuem interfaces de comunicação complexas, como: *Ethernet*, WIFI e BT/BLE.

Essa versão oferece:

- Interface de rede padrão 802.3 para transmissão e recepção de quadros *ethernet*;
- Interface *Human-Computer Interaction* (HCI) para conectividade BT/BLE;
- Interface de controle para configuração e controle do WIFI do módulo ESP32.

A solução utiliza a pilha *Transmission Control Protocol* (TCP)/*Internet Protocol* (IP) do dispositivo *host* existente. A comunicação entre o dispositivo *host* e o ESP32 com suporte à comunicação sem fio, pode ser realizada pelos periféricos SPI, *Secure Digital Input/Output* (SDIO) ou *Universal Asynchronous Receiver/Transmitter* (UART). Essa realização é possível por meio de uma camada de *software* notavelmente otimizada em termos de recursos.

Recursos de conectividade

A solução ESP-Hosted-FG disponibiliza os recursos resumidos na Tabela 3 para comunicação sem fio *Wireless Local Area Network* (WLAN), e também oferece suporte a BT/BLE para o dispositivo *host*.

Tabela 3 – Tabela de recursos de conectividade

Recursos WLAN	802.11 b/g/n
	WLAN *Station*
	WLAN *Soft* AP
Recursos BT/BLE	ESP32 suporta BR/EDR e BLE 4.2
	ESP32-C2/C3/S3 suporta BLE 4.2 e 5.0
	ESP32-C6 suporta BLE 5.3

Fonte: Adaptado de Esp-Hosted (2023).

Módulos ESP suportados

A solução ESP-Hosted-FG é compatível com os módulos ESP apresentados na Tabela 4.

Tabela 4 – Tabela de módulos ESP suportados

ESP32	ESP32-S2	ESP32-S3	ESP32-C2	ESP32-C3	ESP32-C6

Fonte: Adaptado de Esp-Hosted (2023).

Hosts MCU suportados

Conforme previamente mencionado, essa variante é principalmente destinada à integração em *hosts* baseados em MCU. A portabilidade foi realizada para dois tipos específicos de MCU, conforme documentado na Tabela 5. Contudo, para a integração em um MCU distinto, será necessário realizar o processo de portabilidade correspondente.

Tabela 5 – Tabela de *hosts* suportados

STM32 Discovery Board (STM32F469I-DISCO)
STM32F412ZGT6-Nucleo 144

Fonte: Adaptado de Esp-Hosted (2023).

Relação de Recursos, Conexão e Módulo

Dependendo da escolha do módulo ESP, os recursos disponíveis (WIFI e BT/BLE) podem utilizar comunicações diferentes entre si. Essas diferenças são descritas em detalhes abaixo:

- **Somente SDIO:** Tanto o WIFI quanto o *Bluetooth* utilizam a interface SDIO para o tráfego de dados;
- **SDIO + UART:** O WIFI opera através da interface SDIO, enquanto o *Bluetooth* utiliza a UART para comunicação;
- **Somente SPI:** Tanto o WIFI quanto o *Bluetooth* utilizam a interface SPI para o tráfego de dados;
- **SPI + UART:** O WIFI opera através da interface SPI, enquanto o *Bluetooth* utiliza a UART para comunicação.

Com base nessas informações, a Tabela 6 apresenta um resumo das funcionalidades disponibilizadas pela biblioteca para dispositivos *hosts* MCU, de acordo com a *release* 0.0.5.

Tabela 6 – Tabela de recursos ESP-Hosted

Módulo ESP	Tipo Conexão	Recursos suportados
ESP32	Somente SDIO SDIO e UART Somente SPI SPI e UART	Wifi / Classic-BT / BLE-4.2
ESP32-C6	Somente SPI SPI e UART	Wifi 6 / BLE-5.3
ESP32-S2	Somente SPI	Wifi
ESP32-C3 ESP32-C2 ESP32-S3	Somente SPI SPI e UART	Wifi / BLE-5.0

Fonte: Adaptado de Esp-Hosted (2023).

2.5.2 Arquitetura do sistema

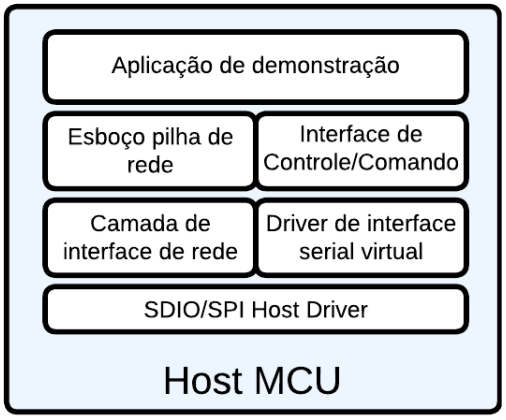
Esta seção ira discutir os blocos de construção da solução ESP-Hosted que tem como base três blocos principais do sistema, de acordo com Esp-Hosted (2023). Os blocos são:

- *Driver* ESP-Hosted-FG e caminho de controle;
- *Firmware* ESP-Hosted-FG;
- Componentes de terceiros.

Driver ESP-Hosted-FG e caminho de controle

Dado que esse trabalho se fundamenta em um MCU e o *driver* está intrinsecamente vinculado à plataforma do dispositivo *host*, a análise se concentrará exclusivamente na variante da arquitetura do sistema em que o *host* é baseado em MCU. Portando, de acordo com Esp-Hosted (2023), os componentes que compõem a estrutura desta solução estão descritos no diagrama de blocos a seguir:

Figura 7 – Diagrama em bloco arquitetura *host* MCU



Fonte: Adaptado de Esp-Hosted (2023).

- ***Driver host* SPI/SDIO:** A solução ESP-Hosted oferece uma pequena camada de interface SPI/SDIO para a comunicação entre o *driver* de *hardware* SPI/SDIO e as interfaces de serial ou de rede. Operam de forma assíncrona, isso é, operam de forma flexível sem depender de um tempo determinado para sua execução. Atualmente, a solução suporta uma capacidade máxima de 1600 *bytes* para transmissão ou recebimento de dados via SPI e 4096 *bytes* no caso de SDIO em uma única transação;

- ***Driver* de interface serial virtual:** A solução ESP-Hosted também oferece uma implementação genérica de interface serial virtual que serve como base para diversos componentes. O componente de interface de controle da solução é construído a partir dessa interface, e o mesmo princípio se aplica à interface HCI, que pode ser desenvolvida com base na interface serial virtual. A interface HCI, por sua vez, proporciona funcionalidades de BT/BLE o qual não será utilizado nesse trabalho;

- **Interface de controle/comando:** A interface é implementada através da interface serial virtual, mencionada acima, utilizada para enviar comandos de controle destinados à configuração e controle das funcionalidades WIFI do módulo ESP32 conectado ao dispositivo *host*. Esta interface é considerada opcional, e se não for usada, é possível empregar o caminho de controle ou funcionalidade BT/BLE na interface UART física conectada ao módulo ESP32. Os detalhes referentes ao design e à implementação do caminho de controle estão disponíveis na documentação específica do mesmo;

- **Camada de interface de rede:** Essa interface é uma camada intermediária que se encontra entre o controlador SDIO/SPI (dispositivo *host*) e uma pilha de rede. Essa camada tem a finalidade de proporcionar flexibilidade, permitindo o uso de qualquer pilha de rede em conjunto com a solução hospedada pelo ESP32. Ela estabelece um conjunto de APIs e estruturas de dados que a pilha de rede precisa seguir para operar de maneira compatível com o *driver* do *host* SDIO/SPI;

- **Esboço da pilha de rede:** Um breve exemplo de uma pilha de rede simples para o desenvolvedor obter uma referência para o desenvolvimento da camada de interface de rede. No entanto, é importante ressaltar que esse exemplo não deve ser considerado como a pilha de rede real a ser utilizada no projeto;
- **Aplicação de demonstração:** Um exemplo prático de uma aplicação de pequeno porte que ilustra as capacidades do ESP-Hosted. Este exemplo explora a interface de controle para controlar e configurar a conectividade WIFI, transmitir e receber dados, bem como enviar solicitações *Address Resolution Protocol* (ARP) para outros dispositivos de rede.

Firmware ESP-Hosted-FG

Essa seção aborda uma breve explicação do *firmware* do módulo ESP32 o qual faz parte da porção da solução denominada ESP-Hosted-FG. É relevante notar que o *firmware* do módulo ESP32 é independente da plataforma do dispositivo *host*.

Conforme evidenciado na Figura 6, o módulo ESP32 foi subdividido em seções que englobam tanto elementos da própria biblioteca ESP-Hosted como componentes provenientes do ESP-IDF, que é um conjunto aberto e gratuito de ferramentas de desenvolvimento e um ambiente de *software* disponibilizado pela *Espressif Systems*, oferecendo uma extensa gama de bibliotecas, *drivers* e ferramentas destinados ao desenvolvimento de *firmwares* (Espressif Systems, 2023b).

Os componentes intitulados ESP-Hosted foram criados pela solução ESP-Hosted e possuem as seguintes responsabilidades:

- Camada de transporte SDIO / SPI;
- *Driver* da interface serial virtual;
- Implementação dos comandos da interface de controle;
- Implementação de uma ponte que interliga os dados entre a interface WIFI, *driver* do controlador HCI e para a plataforma do dispositivo *host*.

Já os componentes intitulados ESP-IDF utilizam a implementação nativa dos recursos que o ESP-IDF oferece para o desenvolvedor, sendo assim, não foram implementados pela biblioteca ESP-Hosted, e possuem as seguintes responsabilidades:

- *Driver* SDIO / SPI;
- *Driver* WIFI;
- *Driver* controlador HCI;
- Camada de protocolo.

Componentes de terceiros

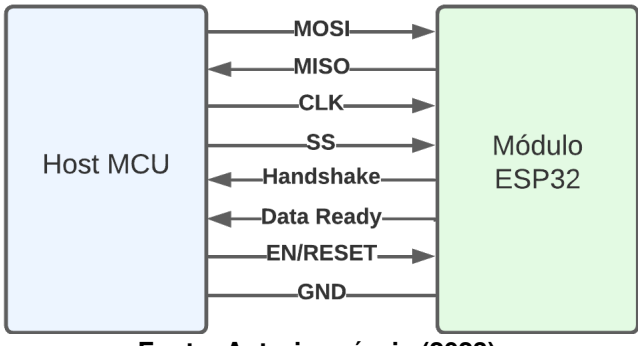
Para o funcionamento completo da solução ESP-Hosted-FG são necessários os componentes de terceiros, como os listados abaixo. A implementação ou adaptação desses componentes de terceiros não está incluída no âmbito do projeto ESP-Hosted e cabe ao desenvolvedor fornecer e/ou integrá-los na aplicação. A implementação destes componentes deve ser realizada somente no dispositivo *host*, conforme representado na Figura 6.

- **Pilha TCP/IP e *Transport Layer Security* (TLS):** Conjunto de protocolos de comunicação, conforme explicado na Seção 2.7;
- **Pilha BT/BLE:** Conjunto de protocolos de comunicação sem fio para BT/BLE (não utilizado no âmbito desse projeto);
- ***Driver* UART:** Controlador encarregado da comunicação serial e responsável por exibir *logs* e depurações do *firmware* no dispositivo *host*;
- ***Protobuf*:** Desempenha a função de serializar os dados que são transmitidos entre os módulos.

2.5.3 Configuração de *hardware*

Neste segmento, serão apresentadas as conexões físicas entre os módulos, juntamente com suas respectivas configurações. A Figura 8 exibe um diagrama abrangendo todas as ligações físicas estabelecidas entre o dispositivo *host* e o módulo ESP32.

Figura 8 – Diagrama em bloco do *hardware*



Fonte: Autoria própria (2023).

Como discutido em detalhes na Seção 2.2, os pinos pertencentes à interface SPI desempenham um papel essencial na transferência de dados entre esses módulos. No entanto, devido suas características, foi necessário a utilização de pinos adicionais para permitir um controle mais abrangente entre os dispositivos. A seguir, será detalhada a funcionalidade dos pinos adicionais, conforme explicado por Esp-Hosted (2023).

- **Pino de *Handshake* (HS):** Em relação ao módulo ESP32, esse pino opera como uma saída digital. Sua principal atribuição consiste em comunicar que o módulo ESP32 se encontra no estado "pronto" para iniciar as transações SPI. É decisivo que o dispositivo *host* aguarde a indicação de prontidão do ESP32 antes de iniciar a transmissão de dados via SPI. Esse pino mantém-se em nível lógico alto durante todo o decorrer da transação SPI;

- **Pino de *Data Ready* (DS):** Em relação ao módulo ESP32, esse pino opera como saída digital. Sua função é notificar o dispositivo *host* que pacotes de dados foram recebidos pela interface de rede sem fio e estão prontos para serem lidos pelo dispositivo *host*. O nível lógico permanece em alto até que o dispositivo *host* tenha finalizado a leitura dos pacotes de dados correspondentes;

- **Pino de EN/*RESET*:** Em relação ao módulo ESP32, esse pino opera como entrada digital. Esse pino tem a responsabilidade de efetuar a reinicialização do módulo ESP32, uma vez que o dispositivo *host* requer essa ação a fim de configurar o referido módulo. Além disso, seu uso é essencial na solução ESP-Hosted baseada em SPI.

É relevante destacar que o documento oficial também aconselha a utilização de fios *jumpers* de alta qualidade e com comprimento limitado, preferencialmente inferiores a 10 centímetros, todos com medidas uniformes. Esse procedimento é particularmente recomendado ao realizar interligações cabeadas entre dispositivos, especialmente devido à alta frequência dos sinais na porta SPI, a fim de mitigar possíveis complicações na transmissão de dados.

2.5.4 Transferência de dados entre *Host* e ESP

Nesta seção, será apresentado como ocorre a transferência de dados entre o dispositivo *host*, atuando como mestre SPI, e o módulo ESP32, desempenhando o papel de *slave*. Conforme destacado por Esp-Hosted (2023), a comunicação é baseada no modo *full duplex* SPI, no qual operações de leitura e escrita ocorrem simultaneamente na mesma transação SPI. Essa sincronia entre as partes é fundamental para assegurar um fluxo contínuo de dados.

Como dito anteriormente, o dispositivo *host* não deve iniciar uma transação antes que o módulo ESP32 esteja pronto para receber os dados. O pino de HS desempenha um papel crucial nesse cenário, indicando ao *host* quando o módulo ESP32 está preparado para a transação SPI.

Para manter um tráfego de dados continuo entre os dispositivos, o módulo ESP32 deve estar sempre pronto para receber os dados do dispositivo *host*. Então, após a conclusão de cada transação SPI, o módulo ESP32 coloca imediatamente na fila a próxima transação SPI.

O protocolo de transferência de dados é definido da seguinte forma:

1. Cada transação SPI consiste em dois *buffers*, o *buffer Transmit* (TX) e o *buffer Receive* (RX). O *buffer* TX contém os dados que o módulo ESP32 pretende transmitir ao dispositivo *host*, enquanto o *buffer* RX é um espaço de *buffer* vazio que, após a conclusão da transação SPI, será utilizado para armazenar os dados recebidos do dispositivo *host*;

2. O módulo ESP32 configura a interface SPI para receber pacotes de no máximo 1600 *bytes*. Isso define a capacidade máxima de dados que o *host* pode enviar a cada transação;

3. Em relação ao *buffer* TX, existem dois casos:

    - Caso o módulo ESP32 não tenha dados para transferir ao dispositivo *host*, um *buffer* TX de tamanho 1600 *bytes* é alocado e definido na transação SPI, porém, o campo de comprimento do pacote da carga desse *buffer* é definido como 0;
    - Caso o módulo ESP32 tenha um *buffer* de dados válidos para ser enviado, o *buffer* TX apontara para esse *buffer* de dados.

4. Depois que os *buffers* de comunicação SPI estão configurados no módulo ESP32, o pino HS é levado para nível logico alto pelo ESP32, indicando ao dispositivo *host* que o módulo ESP32 está pronto para a transação;

5. Caso o *buffer* TX tenha dados validos, o pino de DS também é levado para nível lógico alto pelo módulo ESP32;

6. No âmbito do dispositivo *host*, esse receberá um sinal de interrupção através do pino HS e, após a ocorrência desse evento, será responsável por determinar se pretende ou não realizar a transação SPI. Tal decisão será embasada em:

    a) Se o pino DS estiver em nível lógico alto, o dispositivo *host* realiza a transação SPI;

    b) Ou, se o dispositivo *host* possua dados a serem transferidos para o módulo ESP32, nesse contexto, o dispositivo *host* também procede com a execução da transação SPI;

    c) Se ambas as condições anteriores forem falsas, o dispositivo *host* não realiza a transação SPI.

7. Caso o dispositivo *host* inicie uma transação SPI, ocorre a troca de informações dos *buffers* TX e RX entre os dispositivos, sendo subsequentemente processados por cada um deles;

8. Após a finalização da transação, o módulo ESP32 coloca o pino HS em nível lógico baixo. Caso a transação encerrada contenha um *buffer* TX válido, o pino DS também é colocado em nível lógico baixo pelo módulo ESP32.

Todos esses procedimentos, conforme descrito e implementado por Esp-Hosted (2023), constituem o cerne da solução ESP-Hosted. Essa abordagem engloba a totalidade das trocas de informações entre os dispositivos, resultando em uma comunicação eficaz e resiliente, resistente a falhas e problemas de sincronização.

2.5.5 APIs de caminho de controle

Para aproveitar os recursos oferecidos pela solução ESP-Hosted, é necessário empregar os recursos disponibilizados por sua API. A solução ESP-Hosted disponibiliza uma API destinada ao controle da interface, que consiste essencialmente em funções que aceitam parâmetros específicos e respondem de acordo com a função invocada. As definições técnicas, isso é, as informações necessárias para assegurar o correto funcionamento, devem ser consultadas na documentação oficial correspondente a essa API.

A seguir, são apresentados exemplos de funções fornecidas pela solução ESP-Hosted para os desenvolvedores, conforme mencionado por Esp-Hosted (2023). Os detalhes referentes aos parâmetros e retornos destas funções foram omitidos, sendo apresentados apenas os nomes e suas respectivas descrições.

- **wifi_get_mac**: Obtém o endereço *Media Access Control* (MAC) da interface AP ou *Station* (STA);
- **wifi_set_mode**: Define o modo em que o módulo ESP32 ira funcionar. As opções são: Desativado, Modo STA, Modo AP e Modo AP+STA;
- **wifi_ap_scan_list**: Lista uma série de informações relativas a todos os APs visíveis pelo módulo ESP32;
- **wifi_connect_ap**: Define a configuração do AP à qual a STA do módulo ESP32 deve se conectar;
- **wifi_get_ap_config**: Obtém a configuração do AP à qual a STA do módulo ESP32 está conectada.

As funcionalidades presentes na API da solução ESP-Hosted têm a capacidade de efetuar uma ou mais transações através da interface SPI, o que implica na necessidade de pontos de espera durante sua execução. Com o intuito de aprimorar o desempenho do *firmware* no dispositivo *host* e permitir a alocação do processador para outras tarefas durante esses períodos, a inclusão de um sistema operacional surge como uma decisão essencial.

A relevância dessa eficiência torna-se ainda mais evidente ao considerarmos a demanda por concorrência, viabilizando a operação simultânea de múltiplos clientes ou servidores na interface de rede. Nesse cenário, torna-se indispensável a presença de um sistema operacional para coordenar essas funcionalidades de forma eficaz, assegurando a fluidez, eficácia e estabilidade do sistema como um todo.

Adicionalmente, vale ressaltar que a interface de rede do módulo ESP32 não incorpora a pilha de protocolos TCP/IP. Portanto, para permitir comunicação TCP/IP, deve-se implementar essa pilha de protocolos no dispositivo *host*.

2.6 Sistema Operacional de Tempo Real

Antes de abordarmos um *Real Time Operating Systems* (RTOS), é necessário estabelecer uma definição de sistema operacional. De acordo com Denardin e Barriquello (2019), um sistema operacional é um conjunto de *softwares* que atuam como intermediários entre o *hardware* de um computador e os aplicativos de *software*, desempenhando funções cruciais, como o gerenciamento de tarefas, de memória e de recursos.

Em diversos cenários, o uso de um sistema operacional de uso geral pode parecer uma escolha natural devido sua versatilidade e capacidade de lidar com uma variedade de tarefas. No entanto, é importante ressaltar que a natureza de um sistema embarcado difere de dispositivos de uso geral, como computadores. Os sistemas embarcados são utilizados em cenários específicos como: dispositivos médicos, eletrodomésticos inteligentes, controles de navegação de aeronaves, telecomunicações e etc (Puhlmann, 2014). Dito isso, para atender essas necessidades especificas usa-se o RTOS, definido por Denardin e Barriquello (2019) logo abaixo:

> Sistemas operacionais de tempo real são uma subclasse de sistemas operacionais destinados à concepção de sistemas computacionais, geralmente embarcados, em que o tempo de resposta a um evento é fixo e deve ser respeitado sempre que possível. Esses sistemas são conhecidos na literatura como sistemas de tempo real e são caracterizados por possuírem requisitos específicos de sequência lógica e tempo que, se não cumpridos, resultam em falhas no sistema a que se dedicam. Ressalta-se que o tempo de resposta aos eventos controlados em um sistema de tempo real não deve ser necessariamente o mais rápido possível. A prioridade é o cumprimento dos prazos de todos os eventos controlados pelo sistema. (Denardin; Barriquello, 2019).

De acordo com Prado (2010), para entendermos a importância de um RTOS, precisamos entender o conceito de restrição de tempo. Toda tarefa computacional opera com base em estímulos, sejam eles internos ou externos, realizando o processamento e gerando uma saída. Eventos com restrições de tempo têm um prazo máximo para concluir esse processo, chamado de *deadline* e podem ser classificados como:

- ***Hard Real-Time*:** Possuem requisitos de tempo extremamente rigorosos podendo resultar em consequências graves se as restrições de tempo não forem cumpridas, isto é, o prazo da *deadline* deve ser concluído sem atrasos. Por exemplo, sistema de radar aeroespacial;

- ***Soft Real-Time*:** Possuem requisitos de tempo menos rígidos e podem tolerar pequenos atrasos. Por exemplo, um teclado que gera *inputs* de teclas pressionadas.

No mercado, existem diversos RTOS, sendo o *FreeRTOS* a escolha adotada neste projeto. Conforme indicado pelo site oficial do projeto FreeRTOS (2019), é um sistema operacional de código aberto licenciado sob o *Massachusetts Institute of Technology* (MIT), tornando-o compatível com uma diversificada gama de microcontroladores e arquiteturas, o que lhe confere uma notável versatilidade. Além disso, destaca-se por sua eficiência no consumo de recursos.

2.7 Pilha TCP/IP

Em redes de computadores, o termo "pilha TCP/IP" se refere ao conjunto de protocolos de comunicação que são utilizados para conectar dispositivos em redes, como a Internet. A pilha TCP/IP abrange uma série de protocolos que regulam a forma como os dados são enviados, direcionados e recebidos entre esses dispositivos em uma rede (Parziale *et al.*, 2006). Nesse contexto, a biblioteca *Lightweight IP* (lwIP) será responsável por implementar esse conjunto de protocolos de comunicação nos dispositivos embarcados.

De acordo com Dunkels (2002), lwIP é uma implementação de protocolos TCP/IP que se destaca pela sua eficiência e baixo consumo de recursos. Foi originalmente escrito por Adam Dunkels e continua a ser aprimorado pela comunidade de desenvolvedores. Como citado anteriormente, uma das características principais dessa implementação é a sua capacidade de otimizar o uso de recursos, ao mesmo tempo em que oferece suporte a um conjunto abrangente de protocolos TCP/IP. Essa capacidade o torna adequado para aplicações em sistemas embarcados, que operam sob recursos limitados de memória, tendo como disponibilidade apenas dezenas de *kilobytes* de *Random Access Memory* (RAM) e espaço para aproximadamente 40 *kilobytes* de *Read-Only Memory* (ROM) de código.

Na biblioteca lwIP, encontramos diversos protocolos essenciais, incluindo:

- **IP (*Internet Protocol*):** Incluindo encaminhamento de pacotes em múltiplas interfaces de rede, também chamada de *Network Interface* (NETIF). Em outras palavras, a biblioteca possui suporte para direcionar pacotes de dados entre diferentes NETIFs;

- **ICMP (*Internet Control Message Protocol*):** Para manutenção e depuração de rede;

- **DHCP (*Dynamic Host Configuration Protocol*):** Para configurações de rede dinâmica;

- **TCP (*Transmission Control Protocol*):** Incluindo controle de congestionamento (quando detectado congestionamento, será reduzido a taxa de transmissão para evitar perda de pacotes), estimativa *Round Trip Time* (RTT) (responsável por medir a latência da rede para ajustar a taxa de transmissão apropriada) e recuperação rápida/retransmissão rápida (responsável pela eficiência da transmissão e recuperação de dados utilizando algoritmos de recuperação);
- **HTTP (*Hypertext Transfer Protocol*):** Para serviços *World Wide Web* (WEB).

2.7.1 Integração do lwIP

Para realizar a integração do lwIP em uma aplicação, é necessário realizar duas portabilidades. A primeira, embora não obrigatória, diz respeito à portabilidade do RTOS. O lwIP pode ser utilizado em aplicações sem um RTOS, mas isso é um cenário específico. Por padrão, ao ser baixado, o lwIP oferece portabilidade para o *free*RTOS, *windows* 32 *bits* e sistemas baseados em UNIX. No entanto, se a intenção for utilizá-lo com outro RTOS, essa portabilidade deverá ser implementada manualmente pelo desenvolvedor. A segunda portabilidade é voltada para o *hardware*. Em outras palavras, é necessário adaptar o lwIP para funcionar em um *hardware* específico. Isso envolve a configuração dos *drivers* de *hardware*, ajustes nas interfaces de rede, como *ethernet*, WIFI ou outras interfaces, e configurações específicas para garantir que o lwIP funcione de forma otimizada no ambiente de *hardware* específico (Lwip, 2023).

Arquivo de configuração lwIP

Outro componente importante para a integração do lwIP em uma aplicação é o arquivo de configuração do lwIP, denominado "Lwipopts.h". Este arquivo é de natureza personalizável e permite ajustar totalmente as opções e configurações para atender às necessidades específicas da aplicação (Lwip, 2023). Em termos simples, o "Lwipopts.h" é um arquivo de configuração que contém macros que podem ser ativadas ou desativadas, possibilitando que o desenvolvedor module o comportamento do lwIP de acordo com os requisitos da aplicação. Na Listagem 1, há alguns exemplos e explicações de macros utilizadas no arquivo "Lwipopts.h".

2.7.2 Protocolo HTTP

O protocolo HTTP representa um padrão de comunicação na internet, projetado para a transferência de dados entre um navegador (cliente) e um servidor WEB. Esse protocolo é essencial para a obtenção de recursos, notadamente documentos *HyperText Markup Language* (HTML). Sua operação é fundamentada na arquitetura cliente-servidor, na qual as requisições são iniciadas pelo destinatário, geralmente um navegador WEB, e respondidas por um servidor.

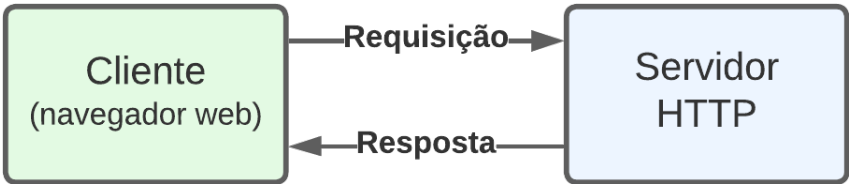
Listagem 1 – Exemplo de um arquivo de configuração lwIP - Lwipopts.h

```
1  #ifndef LWIP_LWIPOPTS_H
2  #define LWIP_LWIPOPTS_H
3
4  #define LWIP_IPV4    1    // Habilitando suporte para IPv4
5
6  // Desativando IPv6 (a aplicação não vai possuir suporte para IPv6)
7  #define LWIP_IPV6    0
8
9  // A aplicação só vai possuir suporte a IGMP se estiver habilitado o IPv4
10 #define LWIP_IGMP    LWIP_IPV4
11
12 // A aplicação só vai possuir suporte a ICMP se estiver habilitado o IPv4
13 #define LWIP_ICMP    LWIP_IPV4
14
15 #define LWIP_TCP     1    // Habilitando suporte de TCP para a aplicação
16 #define LWIP_ARP     1    // Habilitando suporte ARP para a aplicação
```

Fonte: Autoria própria (2023).

Essa comunicação ocorre por meio da troca de mensagens individuais, em oposição a um fluxo contínuo de dados. As mensagens originadas pelo cliente são denominadas solicitações (*requests*) ou requisições, enquanto as mensagens provenientes do servidor como resposta são designadas como respostas (*responses*) (Mozilla Foundation, 2023). A Figura 9 representa visualmente essa comunicação:

Figura 9 – Diagrama em bloco da comunicação HTTP cliente-servidor



Fonte: Autoria própria (2023).

O modelo tradicional de páginas estáticas, inicialmente útil na WEB, tornou-se obsoleto com a ascensão de aplicações online. Atualmente, as aplicações WEB, como compras online, operam no navegador, interagindo com dados em servidores na internet. Esse paradigma desafiou o *software* de aplicação tradicional, exigindo que as páginas WEB se tornem dinâmicas para refletir informações em tempo real, geradas por programas no servidor ou navegador. Nesse contexto, a geração de conteúdo no lado do servidor exemplifica essa evolução, sendo a utilização de formulários um caso comum que demanda processamento no servidor. Quando um usuário preenche um formulário e o envia, uma solicitação é enviada para o servidor junto com os dados preenchidos no formulário, desencadeando um processo dinâmico que pode envolver processos como: Transações bancárias, atualizações de registros no banco de dados ou consultas em APIs. A página resultante é, assim, moldada pelo processamento realizado, tornando-se dinâmica e variável (Tanenbaum, 2003).

Assim, a evolução das aplicações WEB, impulsionada pela obsolescência do modelo tradicional de páginas estáticas, destaca a necessidade de tecnologias como *Server Side Includes* (SSI) e *Common Gateway Interface* (CGI). De um lado, o processamento no servidor, exemplificado pelo preenchimento de formulários, deu origem ao CGI. Do outro lado, para visualizar conteúdo dinâmico, surgiu o SSI como uma solução do lado cliente.

Servidor HTTP no lwIP

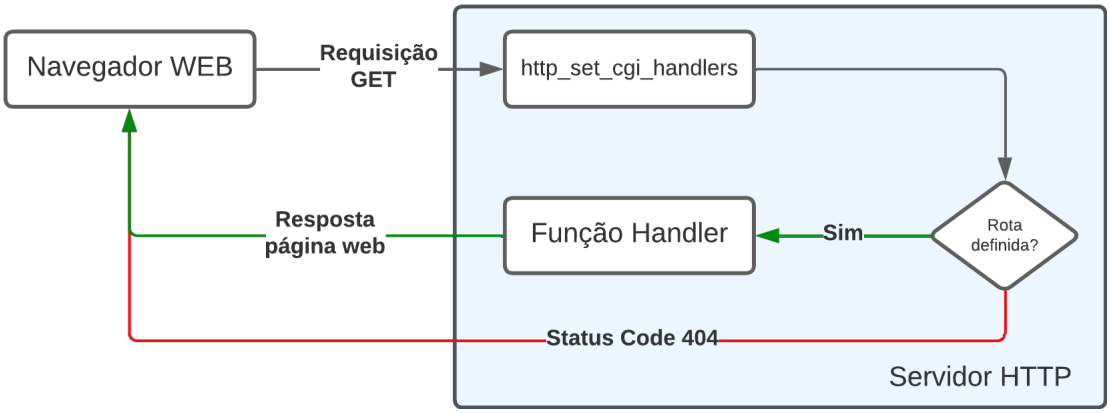
Uma das características do lwIP é sua facilidade em fornecer de maneira direta um servidor HTTP, facilitando o processo por meio de sua API. De acordo com a organização Dunkels (2002), a função central para esta operação é denominada: `httpd_init()`. Ao ser invocada, esta função cria um servidor configurado na porta 80, tornando-se acessível a qualquer endereço IP. Além disso, outras funções, como `http_set_cgi_handlers` e `http_set_ssi_handler`, desempenham um papel importante, responsável por manipular as interfaces CGI e SSI. Através desse conjunto de funções, o lwIP oferece uma abordagem abrangente para criar servidores WEB personalizáveis.

A função denominada `http_set_cgi_handlers`, conforme sua nomenclatura sugere, desempenha a função de configurar a interface CGI. Essa função recebe como parâmetro um ponteiro de função, designado como *handler* do tipo *tCGIHandler*, associado a uma rota específica. Dessa maneira, quando uma requisição CGI é recebida, a função `http_set_cgi_handlers` é invocada verificando a existência de um *handler*, configurado pelo desenvolvedor, correspondente à rota especificada na requisição. A função *tCGIHandler* recebe três parâmetros:

- **iIndex**: Fornece o índice da rota definida na função `http_set_cgi_handlers`;
- **pcParam e pcValue**: Proporcionam acesso aos parâmetros fornecidos pela *Uniform Resource Locator* (URL);
- **iNumParams**: Informa a quantidade de parâmetros enviados ao servidor HTTP por meio da requisição.

O retorno esperado dessa função consiste no caminho do arquivo de resposta, o qual será enviado ao navegador como resposta. A Figura 10 exemplifica o funcionamento do CGI:

Figura 10 – Diagrama em bloco CGI



Fonte: Autoria própria (2023).

Considerações sobre a utilização do CGI:

- O CGI suporta somente solicitações do método *GET*. Portando, os parâmetros devem ser enviados via URL, como por exemplo: `IP/rota?Parametro1=Valor1&Parametro2=Valor2& ... & ParametroN=ValorN`;
- Há um limite para o número de parâmetros, sendo no máximo 16 parâmetros suportados (N=16);
- A função encarregada de manipular o CGI não pode escrever diretamente na saída HTTP. Ao invés disso, deve retornar o nome de um arquivo. O servidor HTTP utilizará esse arquivo como resposta à solicitação CGI recebida, enviando-o posteriormente ao navegador.

Do mesmo modo, a função denominada `http_set_ssi_handler` desempenha o papel de configurar a interface SSI. O propósito desta função é especificar a função responsável por manipular o SSI quando convocada, além de determinar quais *tags* ela irá monitorar. Essas *tags* devem corresponder exatamente aos nomes presentes no arquivo WEB. Após serem validadas pelo manipulador, essas *tags* são substituídas por informações, conferindo dinamicidade à página.

A título de ilustração, caso haja uma *tag* `<!-#temperatura->` no arquivo WEB e ela for designada para monitoramento, cada solicitação que contenha essa etiqueta será encaminhada ao manipulador selecionado. Posteriormente, seu valor será modificado pela lógica da aplicação. A função do manipulador recebe três parâmetros:

- **ssi_tag_name:** Nome da *tag* SSI detectada no arquivo;
- **pcInsert:** Ponteiro para o *buffer* onde a *string* de inserção deve ser gravada;
- **iInsertLen:** Tamanho do *buffer* apontado por `pcInsert`.

A função retorna o número de caracteres gravados em `pcInsert`. Abaixo, algumas observações acerca da utilização das *tags* SSI:

- *Tags* que contenham caracteres como '-' ou espaços em branco não são suportadas;
- O comprimento máximo permitido para o nome das *tags* é determinado pela macro de configuração `LWIP_HTTPD_MAX_TAG_NAME_LEN`, sendo o valor padrão estabelecido em 8 caracteres;
- A *tag* empregada no arquivo WEB deve aderir ao formato padronizado: `<!-#tag->`, em que "*tag*" representa o seu nome.

3 MATERIAIS E MÉTODOS

Neste capítulo, será apresentado uma analise detalhada do desenvolvimento deste trabalho, fornecendo uma descrição dos métodos e as ferramentas que foram utilizadas para a realização do mesmo.

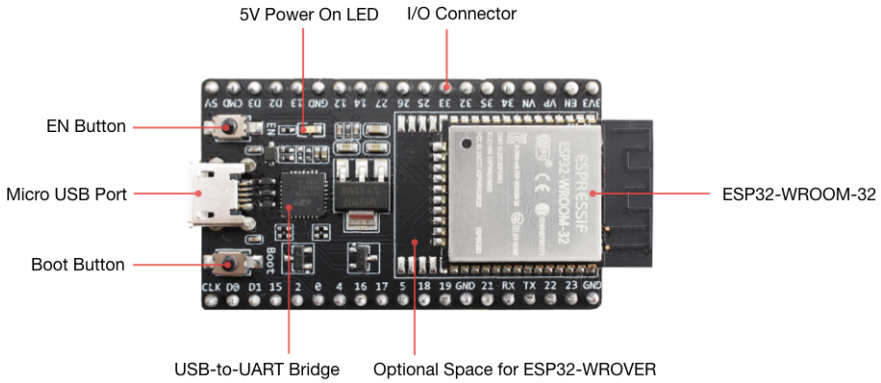
3.1 Materiais

Este capítulo apresenta informações detalhadas sobre os materiais utilizados no desenvolvimento do projeto, abrangendo tanto aspectos de *hardware* quanto de *software*.

3.1.1 Módulo ESP32 DevKitC V4 WROOM-32D

O ESP32-DevKitC é uma placa de desenvolvimento fornecida pela empresa *Espressif Systems*, projetadas para facilitar o desenvolvimento para os módulos ESP32. A ideia da placa é exportar todos os pinos do módulo, oferecer conexão via USB, botões de *boot* e *reset* (Espressif Systems, 2023d). Como mostra a Figura 11 abaixo:

Figura 11 – ESP32-DevKitC V4 com módulo ESP32-WROOM-32 soldado



Fonte: Retirado de Espressif Systems (2023c).

O módulo ESP32-WROOM-32E é soldado à placa ESP32-DevKitC e possui as seguintes configurações:

Quadro 1 – Especificação do módulo ESP32 DevKitC v4

Chip	ESP32-D0WD
Processador	Xtensa 32-Bit LX6 *Dual Core*
Clock	80-240 MHz
SRAM	520 Kb
FLASH	4 Mb (externa)
WiFi*	802.11 b/g/n
Protocolos de segurança	WPA / WPA2 / WPA2-Enterprise / WPS
Periféricos	UART, SPI, SDIO, I2C, etc

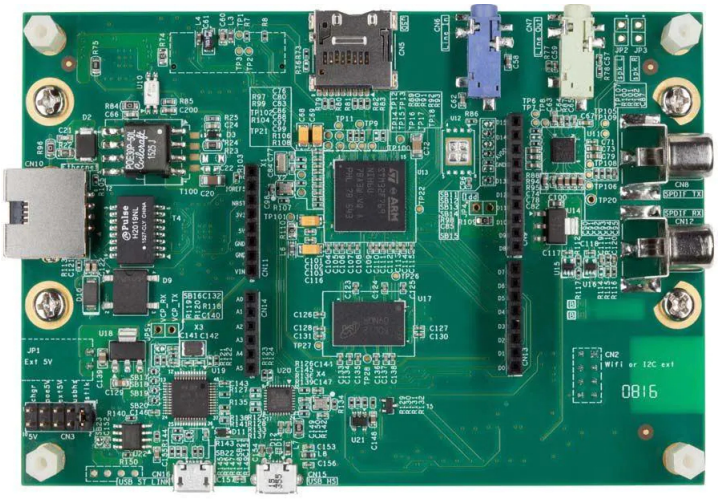
Fonte: Adaptado de Espressif Systems (2023d).

* O módulo WIFI suporta 4x interfaces virtuais e possui suporte simultâneo para os modos de: STA, AP e *Promiscuous* (modo de captura/monitoramento).

3.1.2 *Discovery kit* STM32F769NI MCU

O *kit* de desenvolvimento 32F769IDISCOVERY é um produto da *STMicroelectronics* que faz parte da série *Discovery* STM32 e desenvolvido para o microcontrolador STM32F769NI baseado no processador ARM Cortex M7. Essa placa de desenvolvimento é uma ferramenta que permite ao desenvolvedor criar, testar *software* e *hardware* e aproveitar uma ampla diversidade de recursos que são embarcados na placa, como, por exemplo, áudio, múltiplos sensores, tela gráfica, segurança, vídeos e conectividade de alta velocidade (STMicroelectronics, 2023a). A Figura 12 ilustra o *kit* de desenvolvimento.

Figura 12 – STM32F769I-DISCO



Fonte: Retirado de Mouser Electronics (2023).

A seguir, algumas das principais configurações do MCU:

Quadro 2 – Especificação do *Discovery kit* STM32F769I

Chip	STM32F769
Processador	ARM Cortex-M7
Clock	Até 216 MHz
RAM	532 KBytes
FLASH	2 Mbytes
Periféricos	UART, SPI, I2C, GPIO, USB, Ethernet, LCD, etc

Fonte: Adaptado de STMicroelectronics (2023a).

Foram empregados cabos *jumpers* macho-macho para interligar os módulos, e cabos USB Micro-B para realizar operações de depuração, fornecimento de energia e gravação dos *firmwares* nos microcontroladores.

3.1.3 *Softwares* e bibliotecas

- **ESP-IDF**@v5.2-dev-3775-gb4268c874a: *Espressif IoT Development Framework* (ESP-IDF) é uma estrutura de desenvolvimento projetada para MCU ou SoCs, oferecendo suporte para diversas plataformas (*Windows*, *macOS* e *Linux*). Em resumo, disponibiliza ao desenvolvedor um conjunto de ferramentas de desenvolvimento de *software* utilizadas para programar dispositivos da fabricante *Espressif* (Espressif Systems, 2023b);

- ***STM32CubeIDE***@V1.13.1: Plataforma de desenvolvimento dedicada aos microcontroladores STM32 da *STMicroelectronics*. Esta ferramenta simplifica significativamente o processo de desenvolvimento de *software*, facilitando tarefas como, por exemplo, a criação de códigos de inicialização e configuração de periféricos. Além disso, oferece recursos avançados de depuração, incluindo a visualização de registros da *Central Processing Unit* (CPU) e o monitoramento de variáveis em tempo real (STMicroelectronics, 2023b);

- **Git**@V2.34.1: Sistema de controle de versão gratuito e de código aberto, concebido por Linus Torvalds em 2005. Sua função é registrar e monitorar o histórico de edições em qualquer tipo de arquivo, permitindo, assim, os desenvolvedores a manter um controle preciso das modificações realizadas ao longo do tempo (Git, 2023);

- **PuTTY**@V0.78: Aplicação que, além de permitir conexões via *Secure Socket Shell* (SSH) e Telnet, também oferece a opção de visualizar portas seriais. Essa funcionalidade é útil para interagir com dispositivos que utilizam comunicação serial, facilitando a configuração e a depuração, como, por exemplo, microcontroladores (Putty, 2023);

- **Esp-Hosted-FG**@V0.0.5: Explicado na Seção 2.5;

- ***Free*RTOS**@V10.2.1: Explicado na Seção 2.6;

- **lwIP**@V2.2.0: Explicado na Seção 2.7;

- **Iperf**@V2.0.9: Ferramenta de linha de comando que mede o desempenho de redes, incluindo largura de banda e qualidade da conexão. Ele gera tráfego de rede entre dois computadores, permitindo medir a taxa de transferência de dados em testes unidirecionais e bidirecionais. Além disso, suporta diversos protocolos, como TCP e *User Datagram Protocol* (UDP), tornando-o uma ferramenta valiosa para análises científicas e pesquisas relacionadas ao desempenho de redes de computadores (Iperf, 2023).

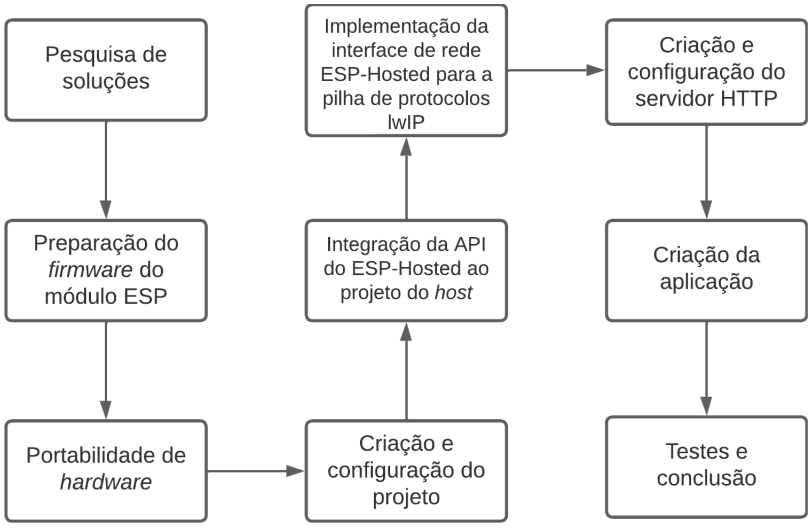
- **Protobuf-c**@V1.4.1: Essa ferramenta é uma dependência da solução ESP-Hosted. Ela é uma implementação em linguagem C do *Protocol Buffers*, uma tecnologia de serialização de dados que permite definir estruturas de dados com arquivos de definição do *Protocol Buffers* (.proto) e gerar código C correspondente para serializar e

desserializar dados de maneira eficiente. Em resumo, é uma ferramenta para aprimorar a comunicação e o processamento de informações estruturadas em sistemas de software (Protobuf-c, 2023).

3.2 Métodos

A Figura 13 apresenta uma representação visual dos estágios de desenvolvimento deste trabalho.

Figura 13 – Fluxograma Metodológico



Fonte: Autoria própria (2023).

3.2.1 Pesquisa de soluções existentes

Uma das etapas fundamentais do projeto foi a análise de soluções já existentes e sua subsequente adaptação no projeto. Essa abordagem se justifica, pois essas soluções oferecem vantagens significativas, proporcionando um adiantamento no processo de desenvolvimento e são continuamente atualizadas pela comunidade ou pela respectiva empresa que o desenvolve. Isso se mostrou particularmente atrativo, uma vez que as funcionalidades do projeto podem evoluir ao longo do tempo, além de lidar com correção de *bugs* e aprimoramentos na segurança. Dado que a ideia central do projeto é utilizar um módulo ESP32 como dispositivo de conexão WIFI, foram exploradas alternativas disponíveis para atender a essa necessidade. Durante a pesquisa que foi realizada no repositório oficial da empresa *Espressif Systems*, foi identificado dois projetos que se mostraram adequados para a integração no projeto. Esses projetos são:

- **ESP-AT**: A empresa *Espressif Systems* desenvolveu um conjunto de comandos AT com a finalidade de facilitar a integração de seus SoCs e módulos que possuem WIFI/BLE. Esses comandos AT permitem uma integração rápida, fornecendo uma pilha TCP/IP integrada, facilidade de integração com plataformas de recursos limitados e

protocolos de resposta de comando fáceis de analisar. Além disso, os desenvolvedores podem personalizar e definir comandos AT de acordo com suas necessidades, o que reduz custos de engenharia e simplifica a incorporação de recursos de conectividade sem fio em produtos novos e existentes (Espressif Systems, 2023a);

- **ESP-Hosted**: Explicado na Seção 2.5.

Uma das primeiras alternativas consideradas foi a utilização da biblioteca ESP-AT. Em resumo, essa biblioteca converte o módulo ESP32 em um dispositivo capaz de responder a comandos AT. No entanto, a interface de comunicação oficial mantida pela empresa é a UART e o SDIO, sendo que a comunicação via SPI foi descontinuada nas versões mais recentes, sendo substituída pelo SDIO. Isso representa um desafio para este projeto, uma vez que um dos requisitos fundamentais é a utilização da interface de comunicação SPI. Outro problema que pode surgir utilizando essa abordagem é a introdução de um *overhead* significativo decorrente da conversão de dados em formato de texto, bem como da interpretação desses comandos.

A segunda alternativa consistiu na utilização da solução ESP-Hosted, conforme detalhado na Seção 2.5. Essa abordagem conseguiu contornar todas as desvantagens associadas à solução ESP-AT. Portando, o próximo passo envolveu a análise detalhada dessa biblioteca e a sua integração com o escopo deste projeto.

3.2.2 Preparação e gravação do *firmware* no módulo ESP32

A gravação do *firmware* no módulo ESP32 possui duas alternativas distintas: a utilização dos binários pré-compilados fornecidos pela ESP-Hosted ou a compilação do *firmware* a partir do código-fonte. Neste trabalho, optou-se por compilar o código-fonte para criar um *firmware* personalizado, incorporando diferentes configurações.

Para efetuar a compilação do código-fonte, é necessário que o ambiente ESP-IDF esteja previamente instalado e configurado. O código-fonte encontra-se disponível no diretório: `esp_hosted_fg/esp/esp_driver/network_adapter` do arquivo ESP-Hosted, o qual foi baixado anteriormente.

Em um terminal, foram executados diversos comandos, seguindo a seguinte ordem:

1. `rm -rf sdkconfig build`: Responsável por apagar o arquivo "sdkconfig", caso ele exista, o qual contém as configurações que serão realizadas, bem como o diretório "*build*" que contém os arquivos necessários para a gravação do *firmware* no módulo ESP32;

2. `idf.py set-target esp32`: Estabelecendo a configuração no ambiente ESP-IDF para utilizar o ESP32 como dispositivo;

3. `idf.py menuconfig`: Comando empregado para acessar uma interface de texto que possibilita ao desenvolvedor realizar modificações nos parâmetros de compilação e configuração em projetos ESP-IDF;

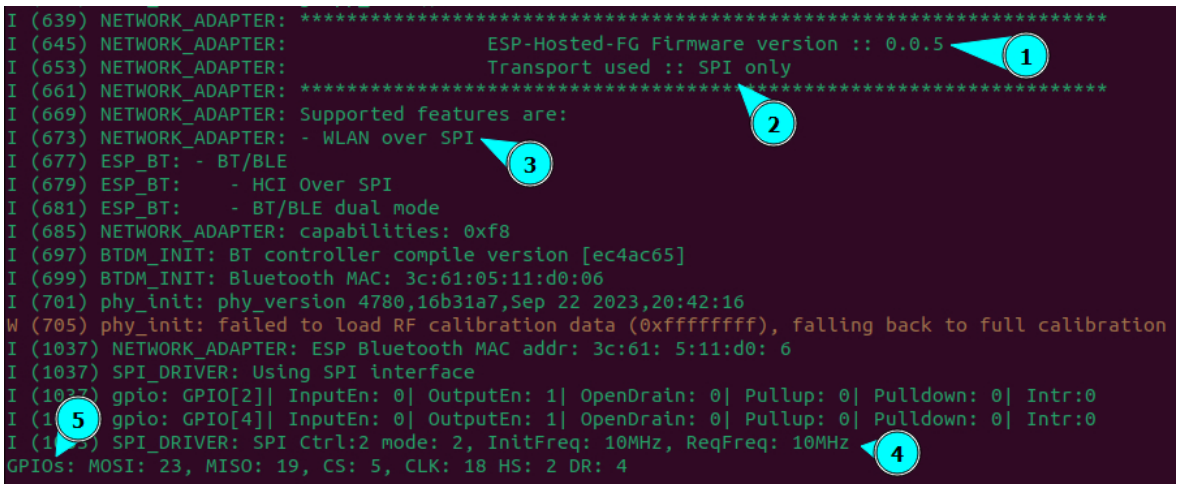
   Dentro desse menu, foi realizado as seguintes configurações:

   - Em: `Example Configuration -> Transport Layer -> SPI Interface -> Enter`, foi modificado a interface de comunicação de SDIO para SPI;
   - Em: `Example Configuration -> SPI Configuration` foi alterado o valor do `SPI controller to use` para 3. Ação executada com o objetivo de modificar o controlador SPI exclusivamente no módulo ESP32, como recomendado pela solução ESP-Hosted.

Depois de salvar as configurações e sair da interface de texto, um arquivo de configuração denominado "sdkconfig" foi gerado, contendo todas as configurações realizadas. Em seguida, o projeto foi compilado com o comando: `idf.py build`.

Após a conclusão bem-sucedida do processo de compilação, o procedimento de gravação e monitoramento do *log* no ESP32 foi realizado com o seguinte comando: `idf.py -p <serial_port> flash monitor`, em que `<serial_port>` representa a porta serial à qual o módulo ESP32 está conectado. A Figura 14 exibe o *log* gerado pelo módulo ESP32.

Figura 14 – *Log* de inicialização ESP32



Fonte: Autoria própria (2023).

Alguns aspectos necessitam de verificação no *log* de inicialização do módulo ESP32, a fim de avaliar o êxito dos procedimentos anteriores, são indicados na Figura 14 com marcadores para facilitar a identificação. São eles:

1. Versão do *firmware*;
2. Tipo de transporte utilizado;

3. Tipo de recurso suportado;
4. Frequência do SPI;
5. *General Purpose Input/Output* (GPIO)s utilizadas.

3.2.3 Portabilidade de *hardware* no dispositivo *host*

Esta seção tem como base a seleção e configuração dos pinos para estabelecer a conexão com o módulo ESP32. Foi realizado uma análise e escolha dos pinos que serão utilizados. Em resumo, essa etapa consistiu na adaptação dos pinos para um novo MCU (STM32F769I-DISCO). Ao consultar o *datasheet* do *kit Discovery*, foi identificado a seguinte tabela:

Tabela 7 – Especificação dos conectores externos - *Discovery kit* STM32F769I

Função	Pino STM32	Nome Pino
I2C1_SCL	PB8	D15
I2C1_SDA	PB9	D14
AVDD	-	AREF
GROUND	-	GND
SPI2_SCK	PA12	D13
SPI2_MISO	PB14	D12
TIM12_CH2, SPI2_MOSI	PB15	D11
TIM1_CH4, SPI2_NSS	PA11	D10
TIM12_CH1	PH6	D9
-	PJ4	D8
-	PJ3	D7
TIM11_CH1	PF7	D6
TIM3_CH3	PC8	D5
-	PJ0	D4
TIM10_CH1	PF6	D3
-	PJ1	D2
USART6_TX	PC6	D1
USART6_RX	PC7	D0

Fonte: Adaptado de STMicroelectronics (2023a).

A Tabela 7 apresenta os conectores disponíveis para o desenvolvedor. Observa-se que "SPI2" é a única interface SPI facilmente acessível para o desenvolvedor, portanto, foi escolhida para uso. Além dos pinos SPI, é necessário selecionar os pinos adicionais: HS, DS e o pino SS da interface SPI. As linhas destacadas na Tabela 7 indicam os pinos escolhidos.

Portanto, na Tabela 8, são apresentadas em detalhes as conexões físicas entre os dispositivos, proporcionando uma visão completa da configuração envolvendo o *hardware*. Os nomes dos pinos em ambos os dispositivos também estão inclusos nessa tabela.

Tabela 8 – Matriz de conexões entre dispositivo *host* e módulo ESP32

Pino STM32	Pino ESP32	Função
PB14 (D12)	IO19	SPI_MISO
PA12 (D13)	IO18	SPI_CLK
PB15 (D11)	IO23	SPI_MOSI
PA11 (D10)	IO5	SPI_CS
PJ1 (D2)	IO2	HANDSHAKE
PJ3 (D7)	IO4	DATA_READY
PJ0 (D4)	EN	RESET ESP32
GND	GND	GND

Fonte: Autoria própria (2023).

3.2.4 Criação e configuração do projeto para dispositivo *host*

A documentação oficial fornece um guia para configurar e preparar o ambiente de desenvolvimento para o dispositivo *host*. No entanto, todas as configurações são específicas para um MCU diferente. Portanto, foi necessário realizar todo o processo de preparação a partir do zero, adaptando as configurações para o novo MCU em questão.

Criação do projeto na CubeIDE

O primeiro passo envolveu a criação de um novo projeto na CubeIDE, seguindo as etapas a seguir:

- Acesse: `File -> New -> STM32 Project`;
- Na janela "MCU/PMU Selector", foi selecionado o MCU STM32F769NIH6;

Após a escolha do MCU, o ambiente de desenvolvimento CubeIDE abriu uma interface gráfica que permitiu configurar os pinos e periféricos de acordo com as necessidades do projeto.

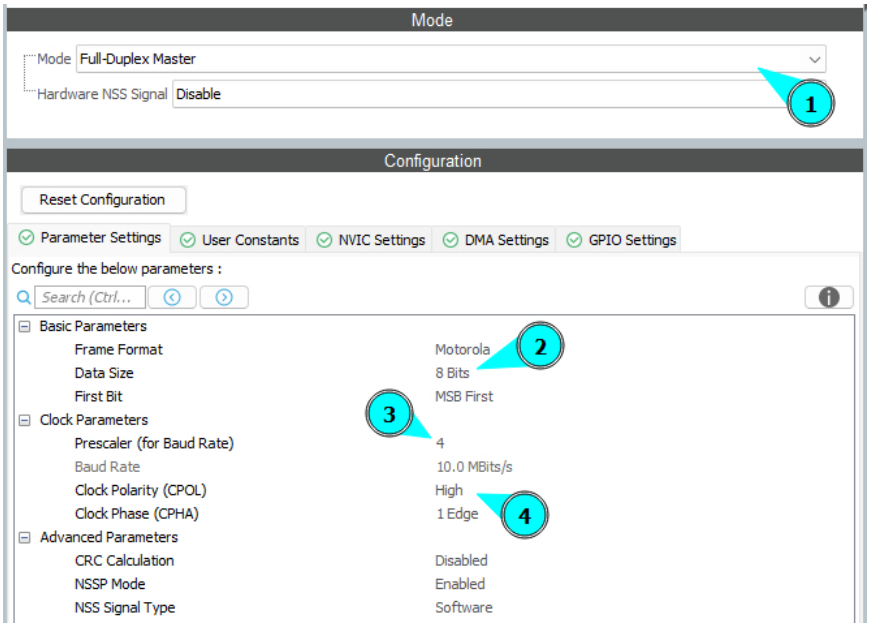
Configuração de *Clock*

Na configuração "RCC", o "Crystal/Ceramic Resonator" foi selecionado como fonte de *High Speed External clock* (HSE). Em seguida, na seção de "configuração do *clock*", foi escolhida a opção de utilizar a fonte de *clock* externa, especificamente a opção: "HSE". Posteriormente, a opção "PLLCLK" foi selecionada com o objetivo de obter um *clock* superior ao *clock* de origem. No campo de: "HCLK (MHz)" a frequência de 160MHz foi escolhida, embora seja importante observar que o MCU suporta uma frequência mais alta de 216MHz. A Figura 30, disponível no Apêndice F, ilustra os passos mencionados.

Configurações de periféricos

Na seção "Pinout & Configuration" do CubeIDE, o SPI2 foi selecionado no campo de conectividade, e o modo "FullDuplex Master" foi configurado, como exemplificado na Figura 15 a seguir:

Figura 15 – Configurações do periférico SPI2

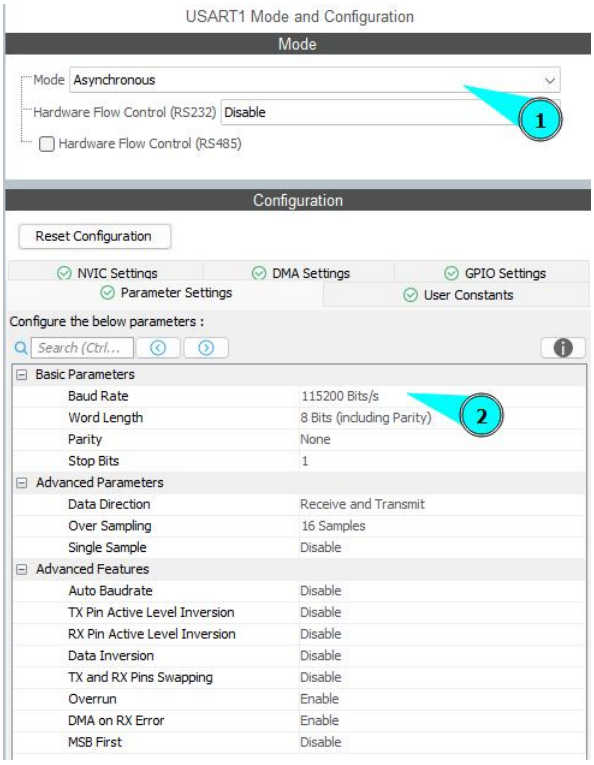


Fonte: Autoria própria (2023).

É importante ressaltar que a configuração efetuada na Seção 3.2.4 referente ao HCLK tem um impacto direto na velocidade do SPI. A taxa de transferência máxima recomendada, inicialmente, é de 10 MBits/s. Para atingir essa velocidade, foi aplicada uma divisão 4x no *clock*, realizada na opção "Prescaler", conforme mostrado na Figura 15.

Na mesma seção de conectividade, foi selecionado o USART1 com o objetivo de visualizar *logs* via comunicação serial, conforme ilustrado na Figura 16 abaixo:

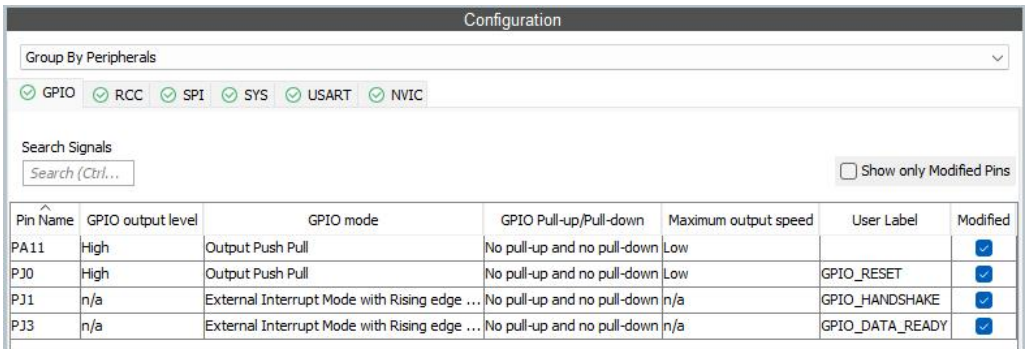
Figura 16 – Configurações do periférico UART1



Fonte: Autoria própria (2023).

Na seção "GPIO" dentro de "*System Core*", procederemos à configuração de alguns pinos conforme mencionado na Seção 3.2.3. As configurações são apresentadas na Figura 17 abaixo:

Figura 17 – Configurações GPIO do projeto



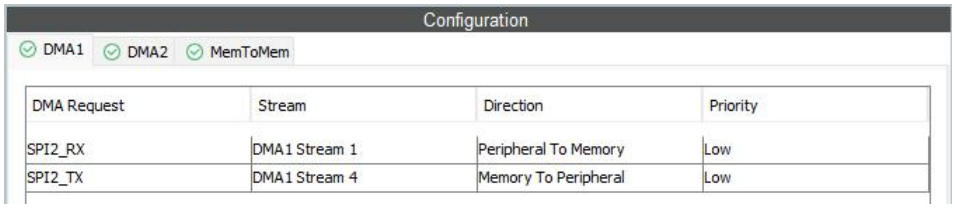
Fonte: Autoria própria (2023).

Resumidamente, foram adicionados e configurados os pinos de controle, e é imperativo que suas configurações sigam as exibidas na Figura 17. As *labels* devem ser nomeadas de acordo com as da imagem, uma vez que a solução ESP-Hosted faz uso destas nomenclaturas.

Configurações DMA

Na seção referente ao *Direct Memory Access* (DMA) no "System Core" do ambiente de desenvolvimento CubeIDE, o periférico SPI2 foi configurado para utilizar a funcionalidade de DMA. A Figura 18 apresentada a seguir ilustra a configuração específica desse processo.

Figura 18 – Configuração DMA

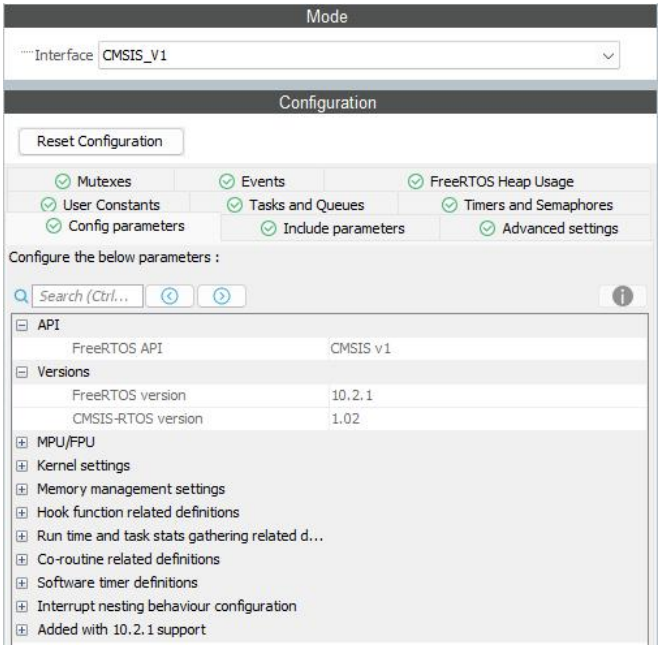


Fonte: Autoria própria (2023).

Configuração dos *middlewares*

Foi utilizado a funcionalidade de configuração fornecida pela ferramenta de desenvolvimento CubeIDE para a implementação do *Free*RTOS no projeto, bem como sua configuração. A configuração pode ser realizada por meio de uma interface gráfica, conforme exemplificado na Figura 19.

Figura 19 – Configuração gráfica do freeRTOS no CubeIDE



Fonte: Autoria própria (2023).

Na seção "*Middleware and Software Packs*" no CubeIDE, optou-se por selecionar o *Free*RTOS e escolher a interface "CMSIS_V1". Além disso, foram feitas as seguintes modificações:

- No campo "Config parameters", o tamanho da "HEAP" foi ajustada para 100000;
- Em "Advanced settings", o parâmetro "USE_NEWLIB_REENTRANT" foi habilitado, com o propósito de mitigar potenciais desafios de concorrência em ambientes multi-tarefa;

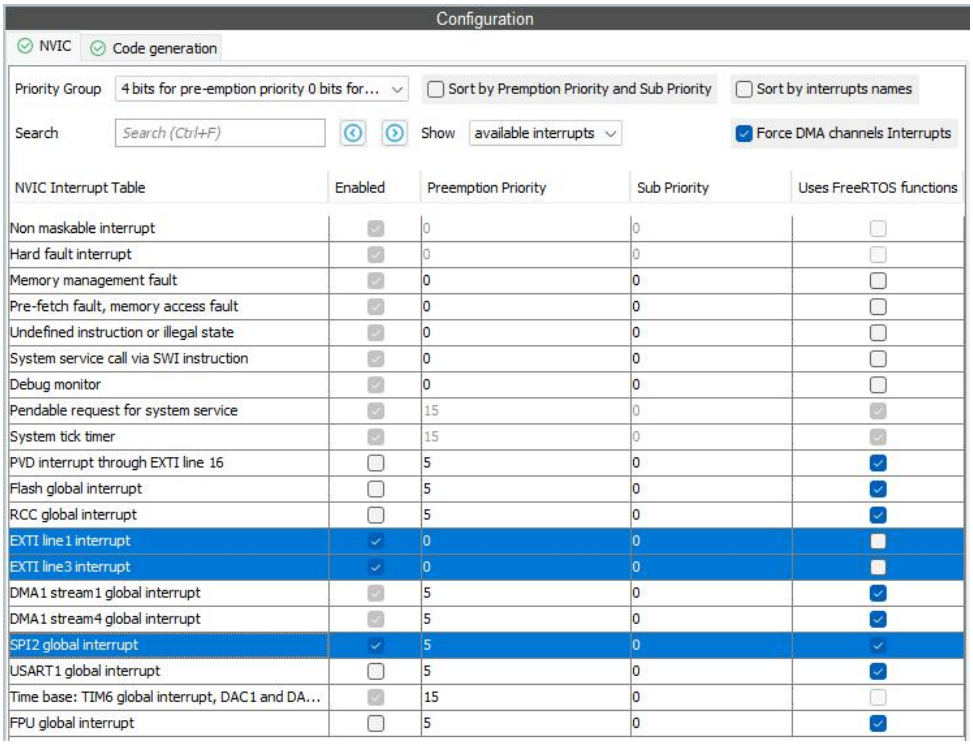
Configuração das interupções

A ativação das interrupções relacionadas ao SPI2 e aos pinos PJ1 e PJ3 por meio das opções disponíveis na opção "NVIC" no menu "*System Core*" são procedimentos indispensáveis. A seguir, são apresentadas as interrupções que necessitam ser habilitadas:

- EXTI *line* 1 *interrupt*;
- EXTI *line* 3 *interrupt*;
- SPI2 *global interrupt*.

A representação final deve ser exatamente como a ilustração exibida na Figura 20 a seguir:

Figura 20 – Configurações NVIC do projeto



Fonte: Autoria própria (2023).

Foram realizadas as configurações do projeto. Após a compilação do projeto, é fundamental garantir que a mensagem "*Build Finished*" seja exibida sem a ocorrência de erros ou avisos, conforme ilustrado no exemplo a seguir:

11:49:20 Build Finished. 0 errors, 0 warnings. (took 2s.869ms)

3.2.5 Integração da solução ESP-Hosted

Esta seção aborda a integração da solução ESP-Hosted no projeto. Importante notar que todos os *path* apresentados são relativos, ou seja, têm sua origem na raiz do projeto.

Um diretório denominado "esp_hosted" foi criado em: `Middlewares/Third_Party` e, dentro dele, foram incluídas duas subpastas, "common" e "host", que foram retiradas do arquivo baixado da solução ESP-Hosted.

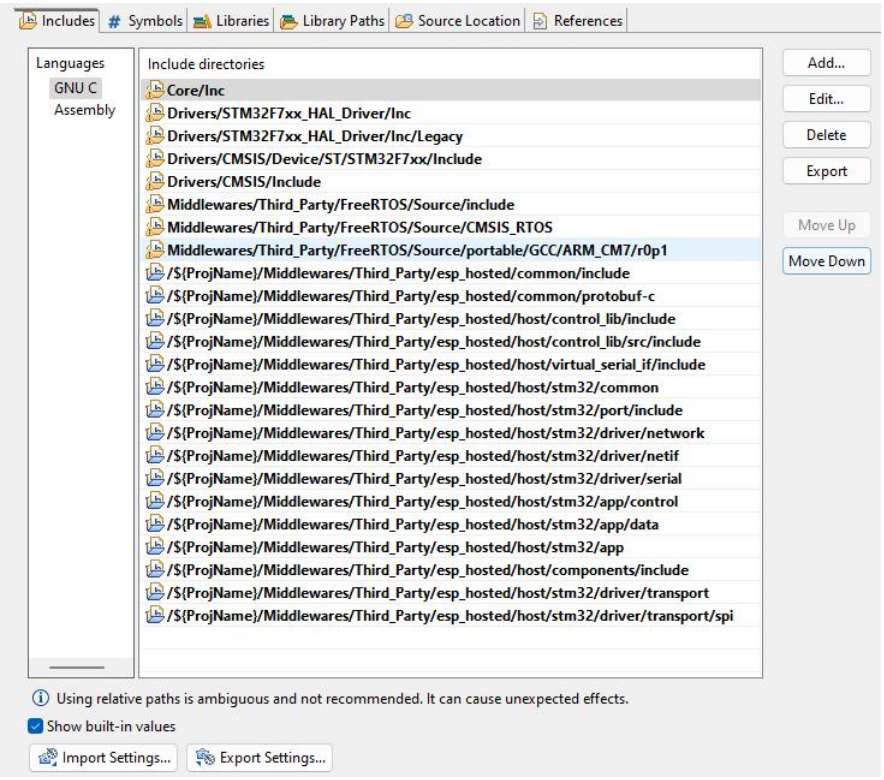
Dentro do diretório `Middlewares/Third_Party/esp_hosted/common/protobuf-c`, foi realizado o procedimento de extração do conteúdo que foi baixado previamente do protobuf-c.

No contexto, os diretórios a seguir foram removidos, uma vez que não se relacionavam com o escopo do projeto.

- `Middlewares/Third_Party/esp_hosted/host/linux`;
- `Middlewares/Third_Party/esp_hosted/host/stm32/proj`.

No ambiente do CubeIDE, foi necessário corrigir os caminhos dos arquivos incluídos, ou seja, fornecer ao CubeIDE os diretórios nos quais os arquivos de cabeçalho estão localizados. A Figura 21 a seguir exibe todos os caminhos que foram configurados.

Figura 21 – Configuração de *paths* no CubeIDE



Fonte: Autoria própria (2023).

Após a etapa de inclusão, torna-se necessário proceder com a exclusão de determinados diretórios do processo de *build*, chamado de: *Exclude from Build...* no CubeIDE. Os diretórios e arquivos foram:

- `Middlewares/Third_Party/esp_hosted/common/protobuf-c/t`;
- `Middlewares/Third_Party/esp_hosted/host/stm32/driver/transport/sdio`.

Solução de problemas da integração da solução ESP-Hosted

Foram identificados alguns problemas, sendo que alguns deles estavam relacionados à versão utilizada pela CubeIDE para a geração dos arquivos de configuração. Os problemas foram resolvidos da seguinte maneira:

- Foi incluído a diretiva `#include "stdint.h"` ao arquivo "trace.h" a fim de resolver questões relacionadas aos tipos de variáveis;
- No arquivo "main.h", foi realizado a inclusão do modificador *export* para as variáveis que contêm estruturas de configuração e utilização de periféricos, tais como SPI e UART, como ilustrado no trecho a seguir:

```
1 extern SPI_HandleTypeDef hspi2;
2 extern UART_HandleTypeDef huart1;
```

- No arquivo "spi_drv.h", as macros "USR_SPI_CS_GPIO_Port" e "USR_SPI_CS_Pin" foram atualizadas para refletir o pino correspondente ao SS da interface SPI, conforme demonstrado na Seção 3.2.3, o qual foi identificado como sendo o pino PA11. Portanto, as macros foram redefinidas da seguinte forma:

```
1 #define USR_SPI_CS_GPIO_Port     GPIOA
2 #define USR_SPI_CS_Pin           GPIO_PIN_11
```

- No arquivo "spi_drv.c", foram efetuadas as seguintes modificações:
  - Comentou-se as diretivas de inclusão dos arquivos "spi.h" e "gpio.h" e incluiu-se a diretiva `#include "main.h"`;
  - Renomeou-se a variável "hspi1" para "hspi2", que foi devidamente importada do arquivo "main.h";
  - Nesse arquivo, foi identificado a presença da função denominada "spi_transaction" possuindo duas versões distintas, "v1" e "v2". De acordo com a documentação oficial da solução ESP-Hosted, a variante "v1" é recomendada para

a utilização em módulos ESP32, enquanto "v2" é indicada para os demais módulos. Analisando as funções, a diferença entre elas reside na abordagem adotada em relação ao gerenciamento do pino SS da interface SPI. A variante "v2" adota uma abordagem na qual o controle desse pino é executado manualmente. Em outras palavras, em cada transação SPI, a responsabilidade de efetuar a alteração do estado lógico do pino SS recai sobre o código do programa. Diferente de uma abordagem "automática" em que o próprio periférico SPI detém a capacidade de gerenciar o estado do pino de SS em cada transação. Essa diferença é apresentada na Listagem 2.

Com base nessas análises efetuadas, constatou-se que a utilização da variante "v1" na qual o controle do pino de SS não é realizado de maneira manual, inviabiliza a execução de transações SPI, em contraste com a variante "v2" que viabiliza esse controle. Portando, foi realizado à modificação no código da solução ESP-Hosted, adotando a variante "v2" para as transações SPI. É de suma importância configurar o pino SS de acordo com as especificações mencionadas na Seção 3.2.4 e garantir que essas configurações sejam refletidas nas macros: "USR_SPI_CS_GPIO_Port" e "USR_SPI_CS_Pin", configuradas anteriormente.

Listagem 2 – Diferenças na função spi_transaction 'v1' e 'v2'

```
1  /* SPI transaction v1 */
2  retval = HAL_SPI_TransmitReceive(&hspi2, (uint8_t *)txbuff,
3      (uint8_t *)rxbuff, MAX_SPI_BUFFER_SIZE, HAL_MAX_DELAY);
4
5  /* SPI transaction v2 */
6  HAL_GPIO_WritePin(USR_SPI_CS_GPIO_Port, USR_SPI_CS_Pin, GPIO_PIN_RESET);
7
8  retval = HAL_SPI_TransmitReceive(&hspi2, (uint8_t *)txbuff,
9      (uint8_t *)rxbuff, MAX_SPI_BUFFER_SIZE, HAL_MAX_DELAY);
10
11 while (hspi2.State == HAL_SPI_STATE_BUSY);
12
13 HAL_GPIO_WritePin(USR_SPI_CS_GPIO_Port, USR_SPI_CS_Pin, GPIO_PIN_SET);
```

Fonte: Autoria própria (2023).

- No arquivo "app_main.c", foram efetuadas as seguintes modificações:
  - Comentou-se a diretiva de inclusão do arquivo "usart.h" e incluiu-se a diretiva `#include "main.h"`;
  - Comentou-se as funções `vApplicationGetIdleTaskMemory` e `vApplicationGetTimerTaskMemory` pois já foram implementadas no arquivo `freertos.c`;

– Renomeou-se a variável "huart3" para "huart1", que foi devidamente importada do arquivo "main.h".

3.2.6 Implementação da interface de rede do ESP-Hosted para a pilha lwIP

A ausência da implementação de uma interface de rede para a pilha de protocolos TCP/IP lwIP no projeto ESP-Hosted demandou a utilização da documentação da biblioteca lwIP para realizar o desenvolvimento da referida interface de rede. Essa integração do lwIP pode ser dividida em dois principais componentes: A primeira parte envolve a alocação dos arquivos da biblioteca (código fonte) do lwIP, enquanto a segunda parte envolve a criação da portabilidade e a configuração do lwIP.

Arquivos fonte lwIP

Para organizar os arquivos da biblioteca, foi criado um diretório chamado "lwip" em: `Middlewares/Third_Party`, e dentro dele, todas as pastas previamente baixadas do lwIP foram inseridas.

Devido à presença de exemplos e funcionalidades nos arquivos do lwIP que não serão aplicados no contexto deste projeto, procedeu-se à exclusão dos respectivos arquivos do diretório "lwip" do processo de *build*. Os arquivos excluídos são listados a seguir: `doc/`, `test/`, `src/apps/`, `src/core/ipv6/`, `contrib/addons/`, `contrib/apps/`, `contrib/Coverity/`, `contrib/examples/`, `contrib/ports/unix/`, `contrib/ports/win32/`.

Arquivos de configurações e portabilidade lwIP

No escopo da configuração e integração do lwIP, foi estabelecida uma estrutura organizacional mediante a criação de um diretório denominado "lwip" em: `Core`. Este diretório contém os arquivos necessários para realizar uma integração eficaz entre lwIP e o *hardware*.

Resumidamente, foram criados três arquivos cruciais para essa integração:

- **"lwipopts.h":** Arquivo de configuração da pilha lwIP, conforme discutido na Seção 2.7. No diretório localizado em: `Middlewares/Third_Party/lwip/contrib/examples/example/app`, contém uma configuração padrão que foi empregada como base neste projeto. Este arquivo de configuração é importante para ajustar o comportamento da pilha de acordo com os requisitos específicos do projeto;
- **"lwip_startup.h":** Arquivo *header*. Agrega as declarações das funções que serão chamadas e implementadas no arquivo ".c" correspondente;

- **"lwip_startup.c"**: Arquivo de implementação dedicado à portabilidade da pilha lwIP para a solução ESP-Hosted. Nesse contexto, são criadas funções específicas que realizam as inicializações e configurações necessárias para estabelecer uma comunicação coesa entre a pilha TCP/IP e o *hardware*. Destaca-se que, para realizar essa integração, foi obrigatório criar funções que inicializam a interface de rede e possibilitam o envio e recebimento de dados por meio dessa interface. Em resumo, este arquivo aborda não apenas a interconexão dos *buffers* das aplicações gerenciados pelo ESP-Hosted com a pilha lwIP, mas também a criação de funcionalidades específicas para a comunicação entre esses componentes. O arquivo em questão apresenta, portanto, três funções principais dedicadas a essas finalidades:
  - **sta_rx_callback**: Atua como um *callback* para processar todos os dados recebidos da interface STA. Para realizar isso, ela utiliza a função `network_read` para ler os dados da interface de rede no qual é implementada pela solução ESP-Hosted;
  - **myif_link_station_output**: Responsável por enviar todos os dados para a rede. Ela calcula o tamanho total dos dados a serem enviados, aloca em um *buffer* e envia os dados para a função `network_write`, implementada pela solução ESP-Hosted, responsável por efetuar o envio desses pacotes de dados para a rede;
  - **lwip_startup_station**: Função principal que abrange a inicialização da pilha lwIP, a vinculação do endereço MAC com a interface de rede, a inicialização dos endereços IP com *Dynamic Host Configuration Protocol* (DHCP) (aplicável apenas à interface STA), e configurações específicas da interface.

  Esse conjunto de funções integraram a pilha TCP/IP com a solução ESP-Hosted, utilizando sua API;

3.2.7 Servidor HTTP

O servidor HTTP foi desenvolvido com a finalidade de disponibilizar aos usuários um meio de configurar e visualizar os parâmetros da rede. Essa configuração engloba principalmente a alteração do *Service Set Identifier* (SSID) e da senha da interface AP, bem como o registro da rede na interface STA, à qual o módulo ESP32 se conectará para obter acesso à rede. Uma página WEB foi criada para oferecer ao usuário a capacidade de realizar essas modificações na rede, além de disponibilizar uma área de *status*, na qual o usuário pode visualizar os SSID das redes configuradas e verificar se o módulo ESP32 está conectado à rede.

Na etapa de implementação do servidor HTTP, foi empregada a aplicação HTTP oferecida pelo lwIP. Esta facilidade de implementação é uma das vantagens destacadas ao optar pelo uso do lwIP.

A aplicação WEB é composta por três páginas distintas:

- **index.shtml:** Ponto central da aplicação, incorporando todas as funcionalidades e recursos disponíveis;
- **success.html:** Criada com o propósito de oferecer um *feedback* ao usuário após a conclusão bem-sucedida de suas solicitações de alterações. O redirecionamento para esta página ocorre para informar que as mudanças foram recebidas e executadas com sucesso;
- **404.html:** Desenvolvida com o objetivo de redirecionar o usuário no caso de acessar uma rota inexistente.

Construção das páginas *web*

No processo de desenvolvimento das páginas, foi considerado o princípio de criar páginas simples e diretas, levando em consideração a limitação de recursos de memória do sistema embarcado. A página principal, localizada no arquivo "*index*", desempenha a responsabilidade central da aplicação, sendo dividida em duas áreas distintas:

- ***Status***: Parte responsável por receber os dados do dispositivo *host* e exibi-los na página WEB, informando ao usuário as configurações atuais. Isso foi implementado utilizando a interface SSI. Para isso, foram criadas quatro *tags* no arquivo *html*, que são responsáveis pela inclusão dinâmica do conteúdo. Essas *tags* são:

Tabela 9 – *Tags* utilizadas na interface SSI

Tags	**Descrição**
ssid_ap	Mostra o SSID da interface de rede AP (WIFI criado pelo módulo ESP32)
view_ap	Mostra o *status* de transmissão do SSID da interface de rede AP
ssid_sta	Mostra o SSID da interface de rede STA (WIFI ao qual o módulo ESP32 se conectará para obter à rede)
conn_sta	Mostra o *status* (Conectado/Desconectado) de conexão da interface de rede STA (WIFI ao qual o módulo ESP32 se conectará para obter à rede)

Fonte: Autoria própria (2023).

  No Apêndice A, é possível analisar como foi feito a inclusão dessas *tags* no arquivo HTML. Importante notar que as *tags* não devem possuir nomes extensos e devem ser incorporadas conforme exemplificado. O arquivo que contém essas *tags* devem ser nomeados com a extensão ".shtml", indicando a integração do SSI na página.

- **Configuração:** Esta seção é encarregada de enviar dados, ou seja, é a parte que transmite as novas configurações fornecidas pelo usuário para o dispositivo *host*, realizada pela interface CGI. Foram desenvolvidas duas rotas distintas: uma para a confi-

guração dos parâmetros da interface de rede AP e outra para STA. Os parâmetros são enviados por meio da URL e incluem:

Tabela 10 – Parâmetros e rotas utilizadas na interface CGI

Rotas	Parâmetros	Descrição
/**config_ap**	ssid_ap	Contém o SSID da interface de rede AP (WIFI criado pelo módulo ESP32)
	pwd_ap	Contém a senha do SSID da interface de rede AP (WIFI criado pelo módulo ESP32)
	show_ssid_ap	Determina se o SSID da interface de rede AP será transmitido (0 para Transmitir, 1 para Ocultar)
/**config_sta**	ssid_sta	Contém o SSID da interface de rede STA (WIFI ao qual o módulo ESP32 se conectará para obter à rede)
	pwd_sta	Contém a senha do SSID da interface de rede STA (WIFI ao qual o módulo ESP32 se conectará para obter à rede)

Fonte: Autoria própria (2023).

No Apêndice A, é apresentada a maneira como o procedimento para a configuração das rotas e seus respectivos parâmetros foi realizado no arquivo HTML.

Os demais arquivos *web*, "success.html", "404.html"e "styles.css", podem ser encontrados nos Apêndices B,C e D, respectivamente. Esses arquivos não envolvem configurações específicas do lado do dispositivo *host*.

Configuração dos diretórios http

O passo inicial para configurar o servidor HTTP fornecido pelo lwIP foi incorporar o diretório da aplicação HTTP no processo de compilação, cujo caminho é `Middlewares/Third\_Party/lwip/src/apps`. No entanto, nem todos os arquivos devem passar pelo processo de *build* e, obrigatoriamente, devem ser excluídos durante esse processo.

A pasta "http" contém todos os arquivos necessários para o lwIP estabelecer um servidor HTTP. A estrutura do diretório é essencialmente a seguinte: os arquivos relacionados à página HTML, como imagens e arquivos WEB, devem ser colocados na pasta "fs". A pasta "makefsdata" é uma aplicação responsável por transformar esses arquivos em arquivos do sistema de arquivos. Portanto, após incluir os arquivos na pasta "fs", é necessário usar o executável localizado na pasta "makefsdata" para realizar essa transformação. O resultado desse processo é armazenado no arquivo "fsdata.c", que é utilizado pelo lwIP.

Os arquivos que foram excluídos do processo de *build* são listados a seguir:

- **Diretório fs:** Não é utilizado diretamente; são os arquivos fonte WEB;
- **Diretório makefsdata:** Não faz parte da aplicação; é apenas um utilitário responsável por auxiliar na transformação dos dados HTML para os arquivos fs;

- **Arquivo fsdata.c:** Este arquivo é utilizado, mas não necessita passar pelo processo de *build*.

Configuração CGI e SSI

Após a configuração do servidor HTTP e a criação das páginas WEB, foi realizado o ajuste do CGI e SSI. O lwIP disponibiliza duas funções para os desenvolvedores, denominadas `http_set_cgi_handlers` e `http_set_ssi_handler`, conforme apresentado na Seção 2.7.2. Dessa forma, foi criado um arquivo no diretório `Core/Src` contendo as configurações dessas interfaces.

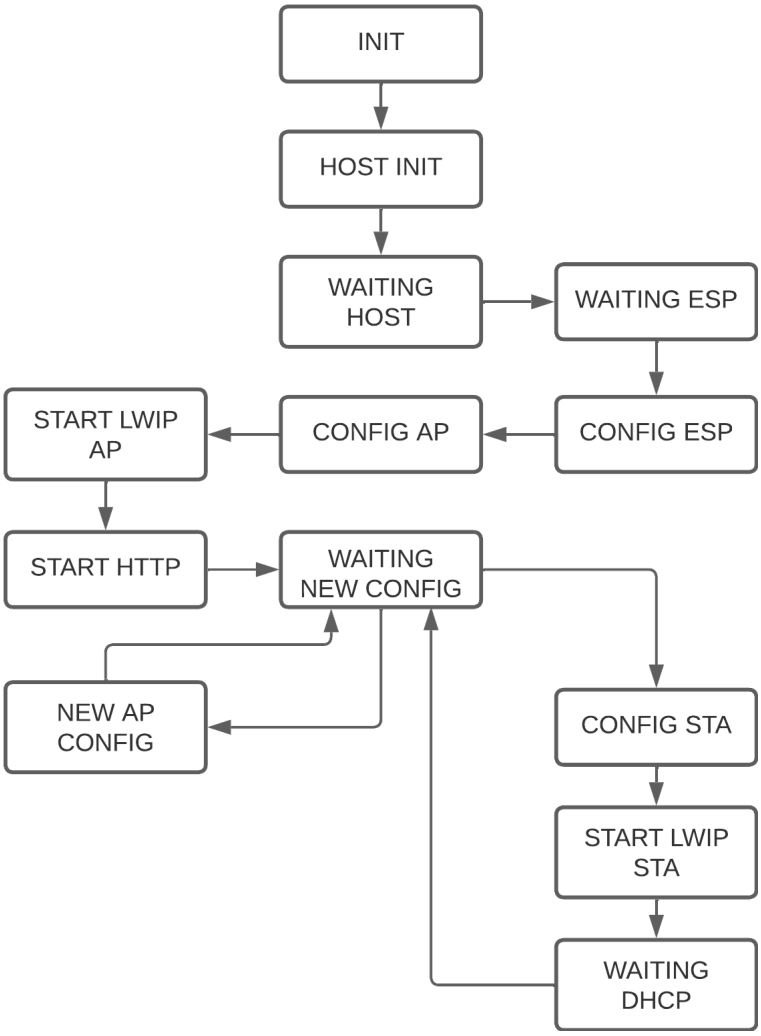
Para a interface SSI, foram declaradas todas as *tags* (Tabela 9) no vetor "SSI_TAGS". Além disso, foi criada a função principal da interface SSI chamada "ssi_handler". Esta função possui um bloco *switch-case* que, de acordo com o valor do parâmetro "iIndex- o qual identifica em ordem as *tags* criadas no vetor - organiza os dados a serem enviados para a aplicação *web*. Por fim, na função `http_set_ssi_handler`, foram definidos os parâmetros que foram criados anteriormente, fornecendo assim a função *handler* e o vetor de *tags*.

Para a configuração da interface CGI, foram declaradas as duas rotas (Tabela 10) em um vetor de estrutura chamado "FORM_CGI". Consequentemente, a função *handler* correspondente a cada rota foi associada sequencialmente. Neste caso, optou-se por utilizar o mesmo *handler* para ambas as rotas, denominado "CGIForm_Handler". A implementação desse *handler* foi a seguinte: O parâmetro "iIndex"informa qual rota recebeu a requisição e, com base nessa informação, a implementação é subdividida em duas partes. Uma parte lida com a rota X, enquanto a outra trata da rota Y. Os parâmetros passados pela requisição são comparados com os do *firmware*, e se houver correspondência, as novas configurações enviadas pela requisição *GET* são armazenadas. Em outras palavras, se o parâmetro enviado pela requisição for o parâmetro esperado pelo *firmware*, as novas configurações são salvas. O código-fonte correspondente a essa implementação pode ser encontrado no Apêndice E.

3.2.8 Criação e configuração da aplicação

Após a conclusão da portabilidade, um módulo foi desenvolvido para centralizar o gerenciamento de todas as operações. Esse módulo é responsável por inicializar as configurações do *hardware*, da solução ESP-Hosted, chamar as funções de portabilidade do lwIP e configurar o módulo ESP32 utilizando a API, conforme detalhado na Seção 2.5.5. A Figura 22 apresenta a máquina de estados desenvolvida para a aplicação.

Figura 22 – Diagrama de bloco do módulo de configuração da aplicação



Fonte: Autoria própria (2023).

Com o propósito de aprimorar a estrutura organizacional, os estados do diagrama em bloco do módulo de configuração foram explicados de forma individualizada, conforme apresentado a seguir:

- **INIT:** No contexto da máquina de estados, as variáveis de controle foram inicializadas neste estado, estabelecendo a estrutura de controle com um valor predefinido igual a zero;
- **HOST_INIT:** Estado encarregado de instaurar o contexto de configuração da solução ESP-Hosted. Este estado tem como propósito iniciar a comunicação SPI com o módulo ESP32, além de criar *mutexes*, filas, *threads* e *buffers*, elementos fundamentais para gerenciar a interação entre os dispositivos. As fases de criação mencionadas foram elaboradas pela solução ESP-Hosted e necessitam ser inicializadas antes de sua aplicação;

- **WAITING_HOST:** Aguarda-se até que todas as configurações mencionadas anteriormente estejam prontas para utilização. Para isso, uma *flag* de conclusão é ativada, possibilitando o prosseguimento da máquina de estados;

- **WAITING_ESP:** Após a inicialização dos parâmetros do lado do dispositivo *host*, este estado aguarda um *timeout* fixo de cinco segundos, permitindo que todas as configurações do lado do módulo ESP32 também entrem em vigor;

- **CONFIG_ESP:** A partir desta máquina de estados, estabeleceu-se uma conexão com o módulo ESP32, permitindo o início das configurações. Para tal, é empregada a API (conforme mencionado em 2.5.5) da solução ESP-Hosted. A primeira configuração efetuada consistiu na definição do modo de operação do módulo ESP32. Dado que o propósito do projeto é empregar o módulo ESP32 para obter conexão com a rede, optou-se pelo modo STA, que possibilita a conexão do módulo ESP32 a outros Pontos de Acesso. Contudo, a outra particularidade do projeto envolve a utilização de um servidor HTTP para configuração, implicando a necessidade de empregar também a interface de rede no modo AP. Ambas configurações (STA+AP) são suportadas pelo módulo ESP32. Para realizar essa configuração, a função `wifi_set_mode` da API foi empregada;

- **CONFIG_AP:** O próximo passo lógico para esta aplicação consistiu na configuração dos parâmetros da interface AP. Essa configuração abrange elementos como: SSID da rede, senha, canal de operação, modo de criptografia, número máximo de conexões permitidas simultaneamente, visibilidade do SSID e banda de operação. Para efetuar a configuração desses parâmetros, foi empregada a função `wifi_start_softap` da API de controle. Essa API retorna como parâmetro o endereço MAC da interface AP, que será utilizado no próximo estado;

- **START_LWIP_AP:** Após a configuração dos parâmetros da interface AP, foi invocada a função para criar o contexto do lwIP, como descrito na Seção 3.2.6. Em resumo, essa função tem como objetivo preparar e configurar toda a estrutura necessária para o funcionamento do lwIP, isso inclui: a inicialização da pilha lwIP com suporte ao RTOS, configuração de parâmetros de rede, adição da interface de rede e configuração do estado da interface. Essa função desempenha um papel fundamental ao estabelecer e preparar a interface AP para a comunicação;

- **START_HTTP:** Com o contexto lwIP definido e configurado, o próximo passo foi iniciar o servidor HTTP. Para isso, foi utilizada uma função específica do lwIP, como detalhado na Seção 2.7.2, para iniciar um servidor HTTP, denominada "`httpd_init`". Após a inicialização do servidor, foi chamada a função responsável por definir e gerenciar as interfaces SSI e CGI, criada na Seção 3.2.7. Portando, a aplicação agora

está pronta para receber novas configurações via HTTP e capacitada para lidar com páginas contendo conteúdo dinâmico;

- **WAITING_NEW_CONFIG:** Essa máquina de estados foi projetada para uma aplicação específica com características distintas, sendo uma delas a espera por configurações da interface STA após a configuração da interface AP. Portando, após a conclusão da configuração do modo AP, a aplicação tem como objetivo aguardar novas configurações realizadas pelo usuário via HTTP, e é exatamente isso que esse estado realiza.

  Após o servidor HTTP receber novas configurações, uma *flag* é definida para indicar a presença de atualizações, e essa *flag* também especifica se as atualizações são direcionadas à interface AP ou STA. Dessa forma, este estado monitora continuamente essa *flag* e, com base em seu estado, direciona a máquina de estados para o estado correspondente. Visto que o servidor HTTP proporciona opções para configurar os modos AP e STA, a máquina de estados está vinculada a dois estados. Portanto, se as novas configurações forem destinadas à interface AP, o módulo redirecionará a máquina de estados para o estado **NEW_AP_CONFIG**; caso sejam para a interface STA, o redirecionamento será para o estado **CONFIG_STA**;

- **NEW_AP_CONFIG:** Este estado é designado para estabelecer novos parâmetros configurados pelo usuário por meio da página *web*. Sua lógica assemelha-se à do estado **CONFIG_AP**; contudo, as modificações efetuadas concentram-se exclusivamente nos parâmetros de SSID, senha e visibilidade do SSID. Notavelmente, outras configurações não são sujeitas à personalização neste escopo de projeto, permanecendo fixas. Após a configuração, a máquina de estados retorna ao estado **WAITING_NEW_CONFIG**, no qual aguarda por novas modificações a serem realizadas nas interfaces;

- **CONFIG_STA:** Este estado foi concebido para configurar o módulo ESP32 como um AP. Se este ponto de acesso tiver conectividade com a *internet*, como por exemplo um roteador WIFI, a aplicação ganha acesso à rede. A configuração foi realizada por meio da função `wifi_connect_ap` da API de controle. Essa função possibilita ao desenvolvedor configurar parâmetros como: SSID da rede, senha, *Basic Service Set Identifier* (BSSID), e determinar se o ponto de acesso oferece suporte ao *Wi-Fi Protected Access 3* (WPA3). O retorno dessa função também fornece o endereço MAC da interface de rede STA, o qual será utilizado no próximo estado;

- **START_LWIP_STA:** Este estado compartilha o mesmo objetivo do estado **START_LWIP_AP**, contudo, o contexto que será criado para o lwIP é baseado na interface STA. Este estado também estabelece uma negociação com o servidor DHCP da rede para obter um endereço IP válido. A função responsável por realizar essa tarefa é a função `dhcp_start`, a qual está vinculada à funcionalidade do lwIP;

- **WAITING_DHCP:** Dado que a obtenção de um endereço IP válido via DHCP é um processo assíncrono, este estado monitora continuamente a variável de retorno do endereço IP até que seja atribuído um valor. Após a obtenção do endereço IP válido, a máquina de estados retorna ao estado **WAITING_NEW_CONFIG**, onde permanece à espera de novas requisições para efetuar novas modificações.

Esta seção conclui a descrição dos métodos aplicados durante o desenvolvimento do projeto. O repositório contendo o código fonte encontra-se disponível no GitHub, conforme Savi (2023). Na próxima seção, serão apresentadas as contribuições, testes e conclusões decorrentes da aplicação destes métodos.

4 RESULTADOS

Este capítulo apresenta os resultados obtidos durante a execução do presente estudo. Inicialmente, serão discutidos os testes relacionados aos protocolos de comunicação. Em seguida, abordaremos o funcionamento da aplicação na visão do usuário, para finalmente aprofundarmos a análise do seu uso, destacando suas vantagens e desvantagens.

4.1 Funcionalidades do projeto

A lista abaixo oferece uma descrição abrangente de todas as funcionalidades implementadas no projeto, detalhando, assim, as diversas operações que a aplicação pode oferecer.

- Suporte a duas interfaces de rede (STA e AP);
- Integração com *free*RTOS;
- Integração com a pilha TCP/IP utilizando a biblioteca lwIP;
- Integração com a solução ESP-Hosted;
- Servidor HTTP;
- Cliente DHCP na interface de rede STA;
- Suporte a *Internet Control Message Protocol* (ICMP) para interface de rede STA;
- Suporte a ICMP para interface de rede AP;
- Servidor iperf para interface de rede STA;
- Configuração de SSID e senha para interface de rede STA via HTTP;
- Configuração de SSID, senha e visibilidade SSID para interface de rede AP via HTTP;
- Monitoramento em tempo real do estado de conexão do modo Estação STA. Em caso de conexão, o servidor fornece, via HTTP, o nome do SSID da rede à qual o STA está atualmente conectado.

4.2 Configuração da interface de rede em modos AP ou STA

A aplicação desenvolvida apresenta uma variedade de parâmetros que são utilizados para configurações das interfaces de rede. É importante observar que esses parâmetros podem ser divididos em duas categorias distintas: ajustáveis via HTTP e fixos no código-fonte.

Os parâmetros ajustáveis via HTTP proporcionam uma flexibilidade ao usuário, permitindo modificações por meio da interface WEB. Por outro lado, alguns parâmetros são fixos, uma vez que suas configurações são diretamente incorporadas ao código-fonte da aplicação. Esses parâmetros, embora não sejam acessíveis para modificação através da interface HTTP, desempenham um papel crucial no funcionamento interno da aplicação.

Os valores fixos desses parâmetros, que também correspondem aos valores padrão, estão apresentados na lista abaixo.

- Modo de trabalho da interface de rede: STA+AP;
- Configurações do modo AP:
  - SSID: ESP_WIFI (configurável via HTTP);
  - Senha: 12345678 (configurável via HTTP);
  - Canal: 1;
  - Tipo de criptografia: WPA2_PSK;
  - Máximo de conexões simultâneas: 2;
  - Transmissão do SSID: Ativada (configurável via HTTP);
  - Banda: HT20;
  - Endereço IP do módulo ESP32: 192.168.2.1;
  - Mascara de sub-rede: 255.255.255.0 (/24).
- Configurações do modo STA:
  - SSID: Não possui valor padrão. Deve ser configurado a partir do servidor HTTP;
  - Senha: Não possui valor padrão. Deve ser configurado a partir do servidor HTTP;
  - Suporte a WPA3: Desativado.

4.3 Testes e exploração de configuração de rede

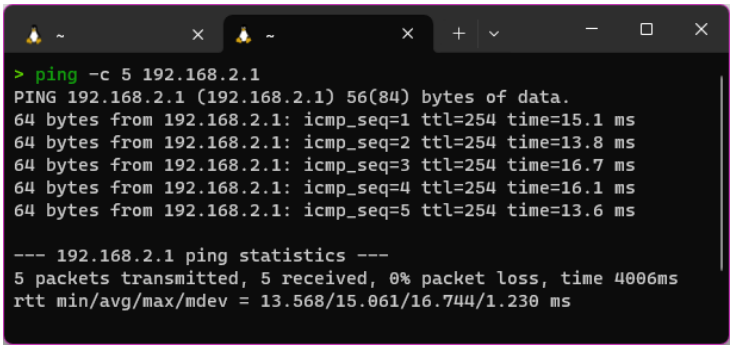
A integração da pilha TCP/IP utilizando a biblioteca lwIP e da solução ESP-Hosted na aplicação desencadeou uma série de testes para avaliar não apenas o desempenho do sistema, mas também a eficácia dessa integração. Nesta seção, exploramos os resultados de uma série de testes, incluindo operações com o protocolo ICMP, DHCP, HTTP e teste de banda. O objetivo desses testes é analisar a resposta da aplicação em diversos cenários de rede, proporcionando uma visão sobre como a integração se comporta em diferentes contextos.

4.3.1 Protocolo ICMP

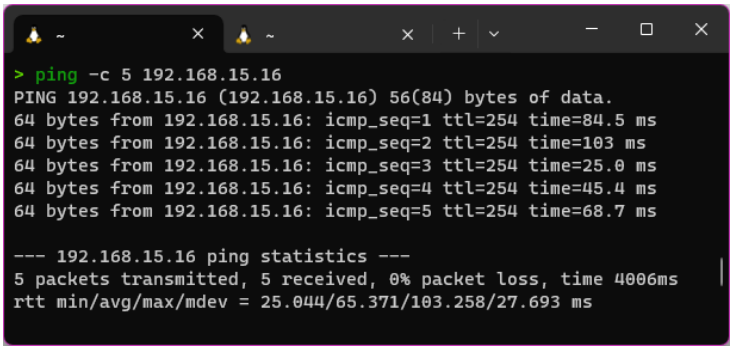
As figuras abaixo ilustram testes ICMP realizados nas interfaces de rede AP e STA, respectivamente.

Figura 23 – Teste do protocolo ICMP

(a) Interface AP



(b) Interface STA

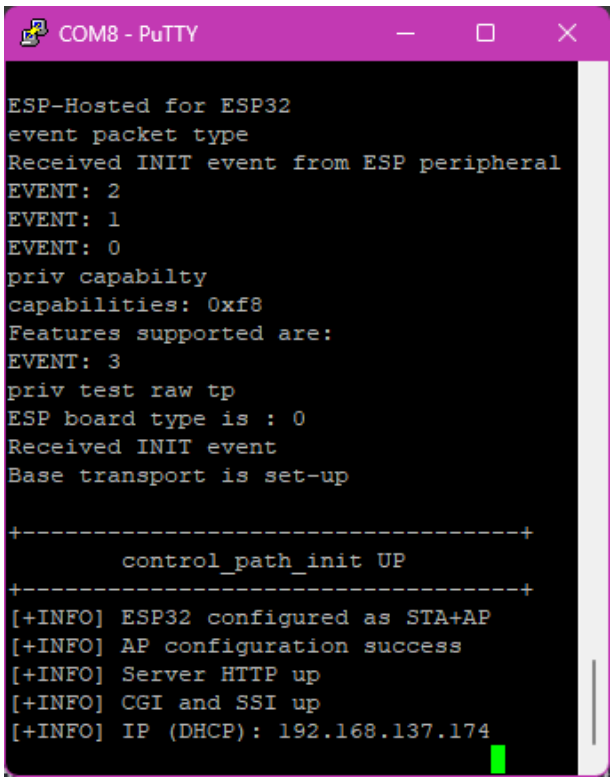


Fonte: Autoria própria (2023).

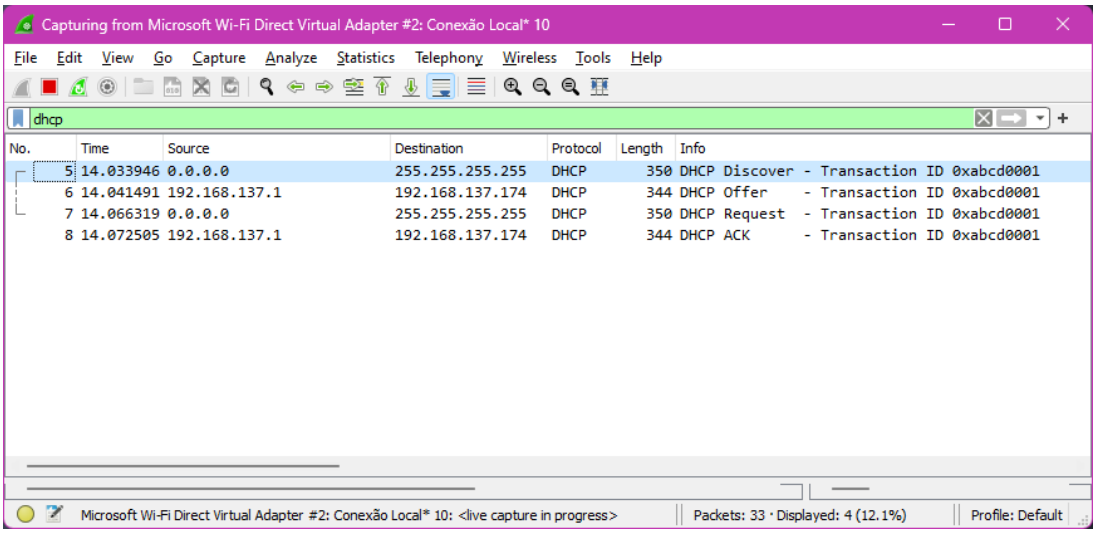
4.3.2 Protocolo DHCP

Esse teste aborda a negociação com o servidor DHCP para obtenção de um endereço IP válido. É relevante destacar que a interface STA possui um cliente DHCP, enquanto a interface AP não dispõe de um servidor DHCP, exigindo a configuração manual do endereço IP.

Figura 24 – Teste do protocolo DHCP
(a) *Log* via serial do módulo STM32



(b) Captura dos pacotes DHCP utilizando *wireshark*



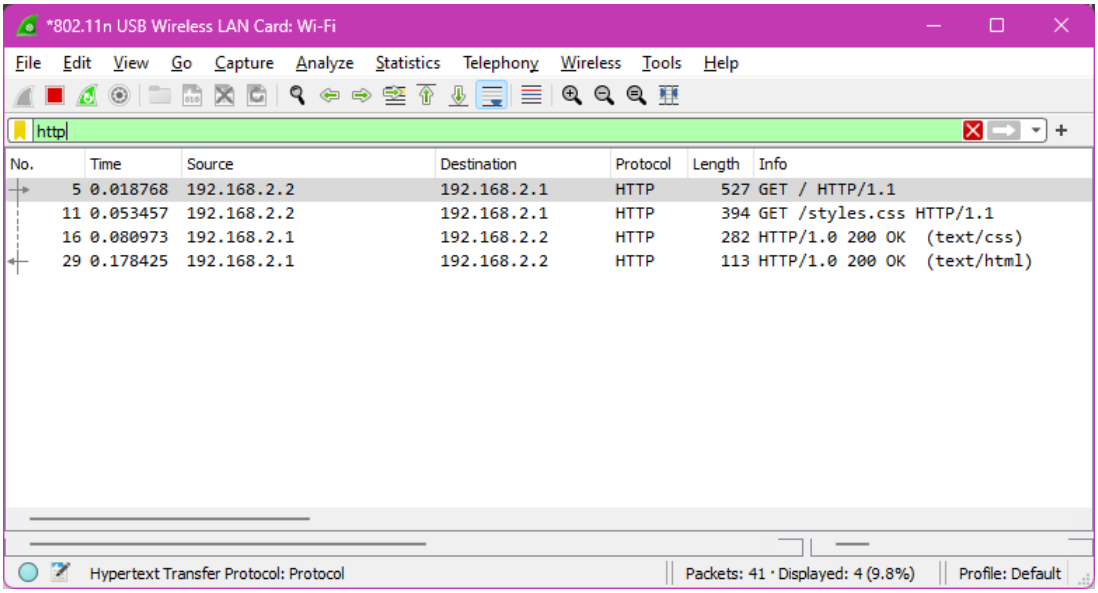
Fonte: Autoria própria (2023).

A Figura 24a representa o *log* do STM32, indicando a obtenção bem-sucedida de um endereço IP válido. A Figura 24b corrobora essa informação, exibindo detalhes da negociação desse endereço IP utilizando o *wireshark* para obtenção dos pacotes DHCP.

4.3.3 Protocolo HTTP

O teste a seguir consiste em uma requisição HTTP utilizando o servidor da aplicação por meio da interface de rede AP.

Figura 25 – Teste de requisição http



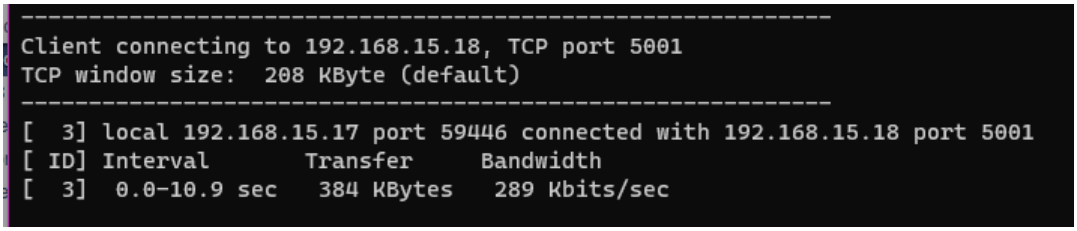
Fonte: Autoria própria (2023).

Na Figura 25, é possível observar o cliente (192.168.2.1) enviando uma requisição do tipo *GET* para o servidor HTTP (192.168.2.1) por meio da interface de rede AP. O servidor respondeu adequadamente à solicitação, destacando uma interação eficaz entre a aplicação cliente e o servidor HTTP.

4.3.4 Protocolo TCP

O teste empregando a ferramenta Iperf tem como propósito a medição da largura de banda e a avaliação da qualidade da conexão de rede entre dois pontos específicos. Esse procedimento possibilita uma análise do desempenho da rede ao identificar a taxa de transferência de dados entre o cliente e o servidor. A execução desse teste proporcionou uma investigação da capacidade de transferência de dados e do desempenho da rede. Os resultados obtidos são apresentados na Figura 26 abaixo.

Figura 26 – Teste de banda utilizando a ferramenta Iperf



Fonte: Autoria própria (2023).

Podemos observar que a quantidade de dados transferidos entre o cliente e o servidor (aplicação) durante a execução do teste foi de 384 bytes, enquanto a taxa de transferência foi de 289 Kbits por segundo.

No projeto da *free*PMU, a taxa média de transmissão de dados para o servidor é de 30 Kbps ao enviar dados sem harmônicas, e aumenta para 200 Kbps ao considerar dados com harmônicas.

4.4 Aplicação WEB

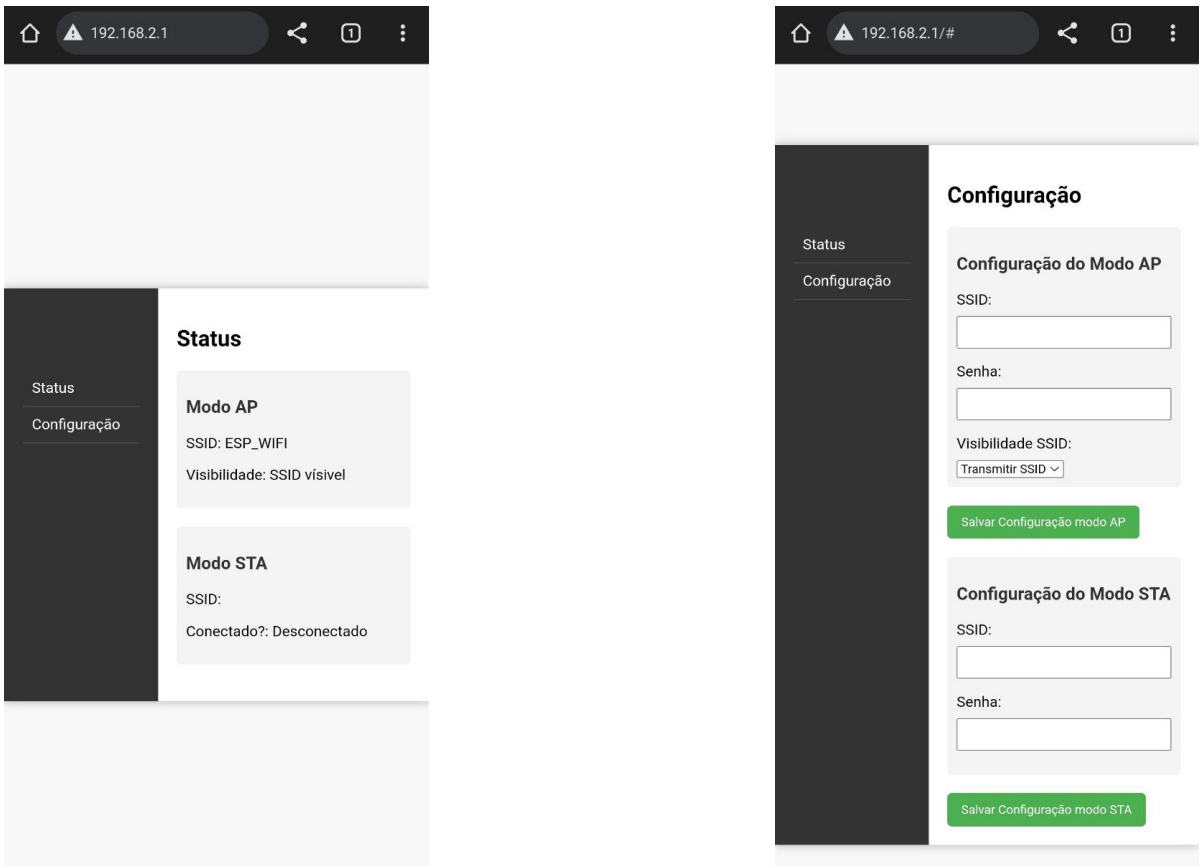
Esta seção apresenta os resultados relacionados ao servidor *http*, mais precisamente às páginas WEB criadas.

Após iniciar o *firmware*, é necessário conectar-se ao AP criado pelo módulo ESP32 para obter acesso ao servidor HTTP hospedado. Vale ressaltar que a interface AP não dispõe de um servidor DHCP, portanto, o endereço IP e a máscara de sub-rede devem ser configurados manualmente. Detalhes como nome da rede WIFI, senha e endereços IP foram abordados na Seção 4.2.

Uma vez inserido o endereço IP do servidor HTTP, será apresentada a página inicial da aplicação denominada "*status*", exibindo informações com os parâmetros atuais, conforme ilustrado na Figura 27a. A página de configurações está acessível no menu lateral denominado "Configuração", cuja aparência é demonstrada na Figura 27b.

Figura 27 – Página Inicial da aplicação WEB

(a) Menu *Status* **(b) Menu Configuração**

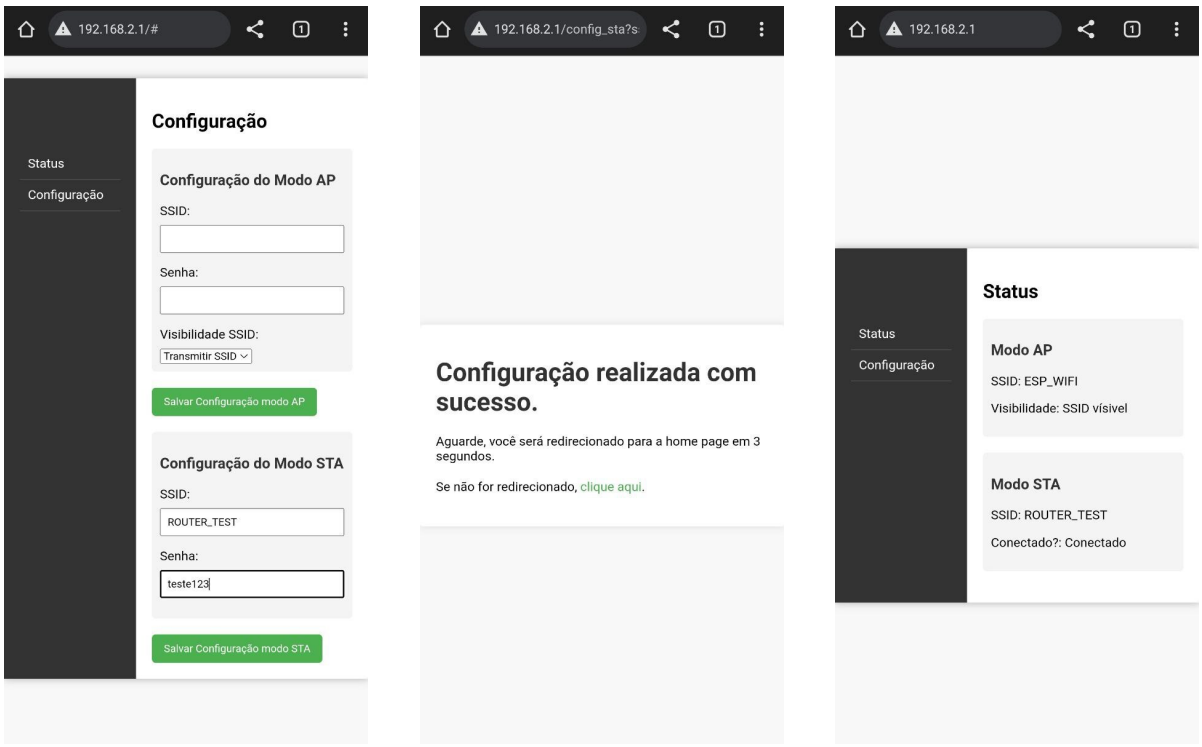


Fonte: Autoria própria (2023).

Observa-se que, ao ser iniciada pela primeira vez, a interface de rede STA não vem configurada, pois é esperado que o usuário forneça o SSID e a senha da rede. Nesse contexto, as figuras abaixo ilustram o processo de configuração da interface STA. Importante ressaltar que o procedimento e as páginas são semelhantes para a configuração do modo AP, divergindo apenas nas informações contidas na página, uma vez que são dinâmicas.

Figura 28 – Processo de configuração das interfaces

(a) Menu configuração **(b) Página de *feedback positivo*** **(c) Menu *status***



Fonte: Autoria própria (2023).

Na Figura 28a, são fornecidos um *ssid* e senha válidos de um AP (roteador) com acesso à internet. Após a submissão, a Figura 28b retorna com uma página confirmando que as configurações foram aceitas e realizadas com sucesso. Posteriormente, a página redireciona para o menu principal (*status*) indicando que a interface STA já está conectada, conforme exemplificado na Figura 28c. Em casos de uma rota inválida ou problemas na configuração, o usuário é redirecionado para a página 404, conforme ilustrado na Figura 29 abaixo.

Figura 29 – Página de feedback *negativo*

Fonte: Autoria própria (2023).

4.5 Gerenciamento de Recursos: Memória RAM e Flash

Um dos aspectos importantes a serem examinados, particularmente em sistemas embarcados, nos quais os recursos de memória são estritamente limitados, refere-se aos recursos de memória empregados por bibliotecas. Iniciando pela análise das páginas WEB, o consumo de memória FLASH associado a essas páginas é apresentado na Tabela 11 abaixo.

Tabela 11 – Consumo de memória FLASH das páginas WEB

Descrição	Memória FLASH
index.shtml	4,75 KB
success.html	1,1 KB
styles.css	776 B
404.html	684 B

Fonte: Autoria própria (2023).

Observa-se que, exclusivamente no contexto das páginas WEB, o consumo total de memória FLASH atingiu 7,31KB, podendo variar conforme as funcionalidades e recursos específicos oferecidos pelas páginas.

No que diz respeito ao consumo de recursos das bibliotecas empregadas, a Tabela 12 abaixo proporciona uma estimativa do total de memória ocupada pelo projeto no MCU.

Tabela 12 – Consumo de recursos de memória no *firmware*

Descrição	Memória RAM	Memória FLASH
Projeto, FreeRTOS lwIP e ESP-Hosted	116,97 KB	205,32 KB

Fonte: Autoria própria (2023).

A análise do consumo de memória pode se tornar complexa ao lidar com bibliotecas, uma vez que depende dos recursos específicos que você está utilizando. Nesse contexto, a avaliação do consumo de memória foi fundamentada na consideração global da solução implementada. Vale ressaltar que essa avaliação também está sujeita a variações de acordo com a configuração da "HEAP". No âmbito deste projeto, a macro `configTOTAL_HEAP_SIZE` foi definida como 10000 *bytes*, conforme modificado na Seção 3.2.4. Assim, levando em consideração o MCU empregado, que dispõe de 512 KB de memória RAM e 2 MB de memória FLASH, o projeto globalmente consumiu 22.85% da memória RAM e 10.03% da memória FLASH do MCU.

5 CONCLUSÃO

O desenvolvimento desse trabalho teve como principal finalidade viabilizar a integração de uma interface de rede sem fio, a um MCU que não possui essa tecnologia. Essa demanda surgiu no contexto do projeto *free*PMU, o qual requeria uma conexão sem fio para operar em locais remotos e/ou de difícil acesso à internet cabeada. Assim, o projeto foi concebido com foco especifico na solução desse desafio.

Ao longo do processo de desenvolvimento, a abordagem foi centrada exclusivamente na PMU, demandando pesquisas para identificar a solução mais adequada aos parâmetros do projeto. Inicialmente, considerou-se a solução ESP-AT, que converte o módulo ESP32 em um dispositivo capaz de responder a comandos AT. No entanto, sua utilização primária era pela interface de comunicação serial, não atendendo plenamente aos requisitos do projeto. Assim, a solução ESP-Hosted foi identificada como mais adequada, embora sua abordagem fosse diferente, transferindo a pilha de protocolos para o dispositivo *host* em vez do módulo ESP32.

Essa adaptação alterou a perspectiva do trabalho, transformando em uma solução versátil para qualquer projeto que enfrenta desafios relativos à conectividade sem fio. Tornou-se uma solução de fácil implementação em MCU, demandando apenas a portabilidade dos pinos de *hardware* para a comunicação SPI.

Foi considerado, inicialmente, a possibilidade de desenvolver uma solução própria. Contudo, a complexidade e a necessidade contínua de testes levaram à preferência por uma solução de código aberto, como a ESP-Hosted, mantida pela comunidade e pelos desenvolvedores da empresa.

O trabalho encontrou desafios significativos devido à falta e à precariedade da documentação fornecida pela solução ESP-Hosted, exigindo a análise detalhada do código-fonte para obter as respostas.

Uma das vantagens destacadas com a utilização da pilha de protocolos TCP/IP no dispositivo *host* foi a facilidade de integração de outros protocolos de comunicação, evidenciada pela facilidade de subir um servidor *host*. No entanto, as desvantagens associadas incluem o consumo de recursos de memória pela biblioteca, um aspecto que pode ser crítico para MCUs que possuem limitações significativas em termos de recursos de memória.

Embora o trabalho não tenha explorado todas as funcionalidades da integração ESP-Hosted e lwIP, concentrou-se na portabilidade dessas soluções. Um exemplo disso é a implementação simplificada de um servidor HTTP para configurar uma pequena gama de parâmetros fornecidos pela solução ESP-Hosted, que oferece amplas possibilidades nesse aspecto.

Durante a implementação do projeto, foram observados alguns desafios que ainda não foram completamente solucionados. Na máquina de estados desenvolvida para consumir a API da solução ESP-Hosted, não foram implementadas validações de erros específicas retornadas pela API de controle. A ausência dessas verificações pode resultar em falhas de tratamento, por exemplo, quando o usuário insere informações inválidas, como senhas ou SSIDs incorretos

no modo STA. Outra pendência identificada refere-se à falta de implementação para a definição manual do endereço IP.

Outro problema enfrentado é o fato do servidor *http* estar disponível tanto para a interface AP quanto para a interface STA. Esse comportamento é justificado pelo fato do *lwip* associar o servidor a um endereço IP e a uma porta em específica. Em tese, bastaria associar o servidor ao endereço IP da interface AP, no entanto, essa possível correção ainda não foi testada.

Adicionalmente, os resultados obtidos no teste de velocidade de banda, utilizando a ferramenta Iperf, foram satisfatórios em relação aos requisitos do projeto da *free*PMU, que demanda uma taxa de transferência em torno de 200 Kbps. O teste revelou uma taxa de 289 Kbps, superando as expectativas. Importante notar que essa velocidade está sujeita a variações de acordo com a frequência definida do *clock* do periférico SPI. Dentro do escopo desse trabalho, a frequência foi fixada em 10 MHz, contudo, é possível alcançar velocidades superiores ajustando esse parâmetro, resultando em um aumento correspondente na taxa de transferência. Vale ressaltar que, ao aumentar o *clock* do periférico SPI, podem ser instauradas instabilidades na comunicação, sendo necessário ponderar entre velocidade e estabilidade.

Concluindo, os objetivos do trabalho foram alcançados ao solucionar o problema de conectividade sem fio para MCUs que originalmente não possuíam essa capacidade, utilizando um módulo externo para essa finalidade. O processo de desenvolvimento, apesar dos obstáculos citados, contribuiu para a ampliação das possibilidades de aplicação da solução, tornando-a não apenas específica para o projeto *free*PMU, mas também aplicável a uma variedade mais ampla de contextos que demandem conectividade sem fio em microcontroladores.

REFERÊNCIAS

AGNIHOTRI, N. **AT Commands, GSM AT command set**. 2021. Disponível em: https://www.engineersgarage.com/at-commands-gsm-at-command-set/. Acesso em: 13.10.2021.

ANDRADE, S. R. C. Sistemas de medição fasorial sincronizada: aplicações para melhoria da operação de sistemas elétricos de potência. Universidade Federal de Minas Gerais, 2008.

DENARDIN, G. W.; BARRIQUELLO, C. H. **Sistemas operacionais de tempo real e sua aplicaçao em sistemas embarcados**. [*S.l.*]: Editora Blucher, 2019.

DHAKER, P. Introduction to spi interface. **Analog Dialogue**, v. 52, n. 3, p. 49–53, 2018.

DUNKELS, A. **lwIP - A Lightweight TCP/IP stack**. 2002. Acessado em 14 outubro de 2023. Disponível em: https://savannah.nongnu.org/projects/lwip/.

ESP-HOSTED. **ESP-Hosted-FG - Docs**. 2023. Acessado em 11 de outubro de 2023. Disponível em: https://github.com/espressif/esp-hosted/tree/master/esp_hosted_fg.

Espressif Systems. **ESP-AT - Overview**. 2023. https://github.com/espressif/esp-at. Acessado em 22 de outubro de 2023.

Espressif Systems. **ESP-IDF - Get Started**. 2023. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. Acessado em 26 de outubro de 2023.

Espressif Systems. **ESP32-DevKitC V4 Getting Started Guide**. 2023. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html. Acessado em 18 de outubro de 2023.

Espressif Systems. **ESP32 Series - Datasheet**. 2023. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acessado em 18 de outubro de 2023.

FALCAO, D. M. *et al.* Integração de tecnologias para viabilização da smart grid. **III Simpósio Brasileiro de Sistemas Elétricos**, p. 1–5, 2010.

FREEPMU. **FreePMU**. 2018. Disponível em: https://github.com/gustavowd/FreePMU. Acesso em: 13.10.2021.

FREERTOS. **RTOS - Free professionally developed and robust real-time operating system for small embedded systems development**. 2019. Disponível em: https://www.freertos.org/RTOS.html.

GIT. **GIT Documentation**. 2023. https://git-scm.com/doc. Acessado em 18 de outubro de 2023.

GRANDO, F. L. **Arquitetura para o desenvolvimento de Unidades de Medições Fasorial Sincronizada no monitoramento a nível de distribuição**. 2016. 162 p. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal Do Paraná, Pato Branco, 2016.

IBM. **Application Programming Interface (API)**. 2020. Disponível em: https://www.ibm.com/cloud/learn/api. Acesso em: 3.11.2021.

IPERF. **iperf: A TCP, UDP, and SCTP network bandwidth measurement tool**. 2023. https://github.com/esnet/iperf. Acessado em 18 de outubro de 2023.

JAMIL, E.; RIHAN, M.; ANEES, M. Towards optimal placement of phasor measurement units for smart distribution systems. *In*: . [*S.l.*: *s.n.*], 2014. p. 1–6.

JOSHI, P. M.; VERMA, H. Synchrophasor measurement applications and optimal pmu placement: A review. **Electric Power Systems Research**, v. 199, p. 107428, 2021. ISSN 0378-7796. Disponível em: https://www.sciencedirect.com/science/article/pii/S0378779621004090.

LEENS, F. An introduction to i²c and spi protocols. **IEEE Instrumentation Measurement Magazine**, v. 12, n. 1, p. 8–13, 2009.

LWIP. **LwIP Application Developers Manual**. 2023. https://lwip.fandom.com/wiki/LwIP_Application_Developers_Manual. Acessado em 17 de outubro de 2023.

Mouser Electronics. **STM32F769I-DISCO**. 2023. https://br.mouser.com/ProductDetail/STMicroelectronics/STM32F769I-DISCO?qs=7UaJ5Mrpeu0U%2FOoa82WAYg%3D%3D. Acessado em 18 de outubro de 2023.

Mozilla Foundation. **An overview of HTTP**. 2023. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. Acessado em 10 de novembro de 2023.

PARZIALE, L. *et al.* Tcp/ip tutorial and technical overview. IBM Redbooks, 2006.

PRADO, S. **Sistemas de Tempo Real – Parte 1**. 2010. Disponível em: https://sergioprado.org/sistemas-de-tempo-real-part-1/.

PROTOBUF-C. **Protobuf-c - Overview**. 2023. https://github.com/protobuf-c/protobuf-c. Acessado em 16 de outubro de 2023.

PUHLMANN, H. F. W. Sistemas operacionais de tempo real-introdução. **disponıvel na internet em http://www. embarcados. com. br/sistemas- operacionais- de- tempo- real- rtos/. Acessado em**, v. 16, 2014.

PUTTY. **Putty**. 2023. https://www.putty.org/. Acessado em 18 de outubro de 2023.

REDHAT. **What is an API?** 2017. Site RedHat. Disponível em: https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces.

SACCO, F. **Comunicação SPI – Parte 2**. 2014. Site Embarcados. Disponível em: https://www.embarcados.com.br/comunicacao-spi-parte-2/.

SAVI, A. **Projeto de interface de rede sem fio utilizando ESP32 como coprocessador**. 2023. Acessado em 22 novembro de 2023. Disponível em: https://github.com/Alfredosavi/espHosted.

STMICROELECTRONICS. **Discovery kit with STM32F769NI MCU**. 2023. https://www.st.com/en/evaluation-tools/32f769idiscovery.html. Acessado em 18 de outubro de 2023.

STMICROELECTRONICS. **Integrated Development Environment for STM32**. 2023. https://www.st.com/en/development-tools/stm32cubeide.html. Acessado em 18 de outubro de 2023.

TANENBAUM, A. S. Redes de computadores: ed. **Campus-Tradução da Terceira Edição, Rio de Janeiro**, 2003.

WOHLFAHRT, A. R. **Simulação de uma Unidade de Medição Fasorial utilizando Typhoon virtual HIL**. 2019. 47 p. Dissertação (Mestrado) — Faculdade de Engenharia Elétrica, Universidade Federal de Santa Maria, Cachoeira do Sul, 2019.

ZANETTA, L. C. J. **Fundamentos de sistemas elétricos de potência**. [*S.l.*]: Editora Livraria da Física, 2006.

APÊNDICES

APÊNDICE A – Código fonte desenvolvido para a página WEB index.shtml

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Home</title>
    <link rel="stylesheet" href="styles.css" />
    <style>
      aside {
        background-color: #333;
        color: white;
        padding: 20px;
        flex: 1;
      }
      p {
        margin-bottom: unset;
      }
      a {
        color: unset;
        text-decoration: unset;
      }
      a:hover {
        text-decoration: unset;
      }
      ul {
        list-style-type: none;
        padding: 0;
        margin: 0;
      }
      li a {
        display: block;
        padding: 10px;
        text-decoration: none;
        color: white;
        border-bottom: 1px solid #555;
        transition: background-color 0.3s;
      }
      li a:hover {
        background-color: #555;
      }
      form {
        max-width: 100%;
        margin: 0 auto;
        margin-top: auto;
      }
      label {
        display: block;
        margin-bottom: 8px;
      }
```

```
50        input {
51          width: 100%;
52          padding: 8px;
53          margin-bottom: 16px;
54          box-sizing: border-box;
55        }
56        button {
57          background-color: #4caf50;
58          color: white;
59          padding: 10px 15px;
60          border: none;
61          border-radius: 5px;
62          cursor: pointer;
63          transition: background-color 0.3s;
64        }
65        button:hover {
66          background-color: #458447;
67        }
68        .info-section {
69          background-color: #f4f4f4;
70          padding: 10px;
71          margin-top: 20px;
72          margin-bottom: 20px;
73          border-radius: 5px;
74        }
75        .info-section h3 {
76          color: #333;
77        }
78      </style>
79    </head>
80    <body>
81      <div class="container">
82        <aside>
83          <h1>Menu</h1>
84          <ul>
85            <li><a href="#" onclick="showSection('status')">Status</a></li>
86            <li>
87              <a href="#" onclick="showSection('configuracao')">Configuração</a>
88            </li>
89          </ul>
90        </aside>
91
92        <main>
93          <section id="status">
94            <h2>Status</h2>
95            <div class="info-section" id="modoAP">
96              <h3>Modo AP</h3>
97              <p>
98                SSID:
```

```
99              <!--#ssid_ap-->
100           </p>
101           <p>
102             Visibilidade:
103             <!--#view_ap-->
104           </p>
105         </div>
106
107         <div class="info-section" id="modoSTA">
108           <h3>Modo STA</h3>
109           <p>
110             SSID:
111             <!--#ssid_sta-->
112           </p>
113           <p>
114             Conectado?:
115             <!--#conn_sta-->
116           </p>
117         </div>
118       </section>
119
120       <section id="configuracao" style="display: none">
121         <h2>Configuração</h2>
122
123         <form method="get" action="/config_ap">
124           <div class="info-section" id="configModoAP">
125             <h3>Configuração do Modo AP</h3>
126             <label for="ssid_ap">SSID:</label>
127             <input
128               type="text"
129               id="ssid_ap"
130               name="ssid_ap"
131               pattern=".{0,32}"
132               title="O campo não deve ultrapassar 32 caracteres."
133             />
134             <label for="pwd_ap">Senha:</label>
135             <input
136               type="text"
137               id="pwd_ap"
138               name="pwd_ap"
139               pattern=".{8,63}"
140               title="A senha deve ter entre 8 e 63 caracteres."
141             />
142             <label for="show_ssid_ap">Visibilidade SSID:</label>
143             <select id="show_ssid_ap" name="show_ssid_ap">
144               <option value="0">Transmitir SSID</option>
145               <option value="1">Ocultar SSID</option>
146             </select>
147           </div>
```

```
148             <button type="submit">Salvar Configuração modo AP</button>
149           </form>
150
151           <form method="get" action="/config_sta">
152             <div class="info-section" id="configModoSTA">
153               <h3>Configuração do Modo STA</h3>
154               <label for="ssid_sta">SSID:</label>
155               <input
156                 type="text"
157                 id="ssid_sta"
158                 name="ssid_sta"
159                 pattern=".{0,32}"
160                 title="O campo não deve ultrapassar 32 caracteres."
161               />
162               <label for="pwd_sta">Senha:</label>
163               <input
164                 type="text"
165                 id="pwd_sta"
166                 name="pwd_sta"
167                 pattern=".{8,63}"
168                 title="A senha deve ter entre 8 e 63 caracteres."
169               />
170             </div>
171             <button type="submit">Salvar Configuração modo STA</button>
172           </form>
173         </section>
174       </main>
175     </div>
176     <script>
177       function showSection(sectionId) {
178         // Esconde todas as seções
179         document.getElementById("status").style.display = "none";
180         document.getElementById("configuracao").style.display = "none";
181         // Mostra apenas a seção desejada
182         document.getElementById(sectionId).style.display = "block";
183       }
184     </script>
185   </body>
186 </html>
```

APÊNDICE B – Código fonte desenvolvido para a página WEB success.html

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="refresh" content="5;url=/" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Sucesso</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <div class="container">
      <main>
        <h1>Configuração realizada com sucesso.</h1>
        <p>
          Aguarde, você será redirecionado para a home page em
          <span id="countdown">5</span> segundos.
        </p>
        <p>Se não for redirecionado, <a href="/">clique aqui</a>.</p>
      </main>
    </div>
    <script>
      function updateCountdown(seconds) {
        document.getElementById("countdown").textContent = seconds;
        if (seconds > 0) {
          setTimeout(function () {
            updateCountdown(seconds - 1);
          }, 1000);
        }
      }
      window.onload = function () {
        updateCountdown(5);
      };
    </script>
  </body>
</html>
```

APÊNDICE C – Código fonte desenvolvido para a página WEB 404.html

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Erro 404 - Página Não Encontrada</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <div class="container">
      <main>
        <h1>Erro 404 - Página Não Encontrada</h1>
        <p>
          A página que você está procurando não existe. Retorne à
          <a href="/">página inicial</a>.
        </p>
      </main>
    </div>
  </body>
</html>
```

APÊNDICE D – Código fonte desenvolvido para a estilização da página WEB

```
body {
  font-family: Arial, sans-serif;
  display: flex;
  min-height: 100vh;
  margin: 0;
  align-items: center;
  justify-content: center;
  background-color: #f8f8f8;
}

.container {
  display: flex;
  max-width: 800px; /* Defina o tamanho máximo desejado */
  width: 100%;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
}

main {
  flex: 2;
  padding: 20px;
  background-color: white;
  border-radius: 5px;
}

h1 {
  color: #333;
}

p {
  margin-bottom: 20px;
}

a {
  color: #4caf50;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}
```

APÊNDICE E – Código fonte desenvolvido para configurar as interfaces CGI e SSI do servidor HTTP

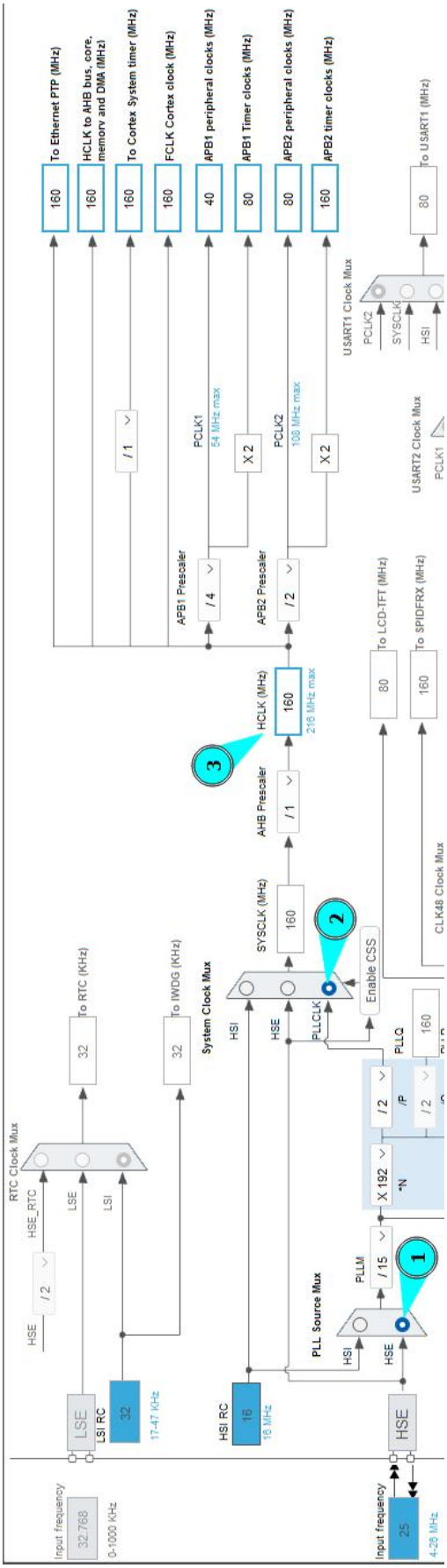
```c
/*
 * http_server.c
 *
 *  Created on: Out 11, 2023
 *      Author: Alfredo Savi
 */

#include "http_server.h"

#include "stdint.h"
#include "string.h"
#include "stdio.h"

#include "main.h"

#include "mod_interface_wifi.h"
#include "mod_persist.h"

#include "httpd.h"
#include "lwip/def.h"

#define SSI_TAGS_NUM   (uint8_t)4


hs_ssi_parameters_t ssi_params = {0};
persist_payload_t cgi_params = {0};
const char* SSI_TAGS[] = {"ssid_ap", "view_ap", "ssid_sta", "conn_sta"};


uint16_t ssi_handler(int iIndex, char *pcInsert, int iInsertLen)
{
  get_data_ssi(&ssi_params);

  switch(iIndex){
    case 0: // ssid_ap
      strcpy(pcInsert, (char *)ssi_params.ssid_ap);
      return strlen(pcInsert);
    break;

    case 1: // view_ap (Se está visivel ou não)
      if(ssi_params.ap_ssid_is_hidden == SSID_BE_BROADCASTED)
        strcpy(pcInsert, "SSID vísivel");
      else
        strcpy(pcInsert, "SSID oculto");

      return strlen(pcInsert);
    break;

    case 2: // ssid_sta
```

```
50        strcpy(pcInsert, (char *)ssi_params.ssid_sta);
51        return strlen(pcInsert);
52      break;
53
54      case 3: // conn_sta (Se está conectado ou não)
55        if(get_status_conn_ap())
56          strcpy(pcInsert, "Conectado");
57        else
58          strcpy(pcInsert, "Desconectado");
59
60        return strlen(pcInsert);
61      break;
62
63      default:
64        return strlen(pcInsert);
65      break;
66    }
67
68    return 0;
69  }
70
71  // ============================ CGI HANDLER ==================================
72  const char *CGIForm_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);
73
74  const tCGI FORM_CGI[] = {
75    {
76      "/config_ap",
77      CGIForm_Handler
78    },
79    {
80      "/config_sta",
81      CGIForm_Handler
82    }
83  };
84
85  const char *CGIForm_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[])
86  {
87    if(iIndex == 0){ // CONFIG AP
88      for(int i=0; i<iNumParams; i++){
89        if(!strcmp(pcParam[i], "ssid_ap")){
90          strcpy((char *)cgi_params.ap_conf.ssid, pcValue[i]);
91
92        }else if(!strcmp(pcParam[i], "pwd_ap")){
93          strcpy((char *)cgi_params.ap_conf.pwd, pcValue[i]);
94
95        }else if(!strcmp(pcParam[i], "show_ssid_ap")){
96          if(*pcValue[i] == '0')
97            cgi_params.ap_conf.ssid_hidden = SSID_BE_BROADCASTED;
98          else
```

```
99              cgi_params.ap_conf.ssid_hidden = SSID_NOT_BE_BROADCASTED;
100           }
101         }
102         set_data_cgi(&cgi_params, UPDATE_AP);
103
104       }else if(iIndex == 1){ // CONFIG STA
105         for(int i=0; i<iNumParams; i++){
106           if(!strcmp(pcParam[i], "ssid_sta")){
107             strcpy((char *)cgi_params.st_conf.ssid, pcValue[i]);
108
109           }else if(!strcmp(pcParam[i], "pwd_sta")){
110             strcpy((char *)cgi_params.st_conf.pwd, pcValue[i]);
111           }
112         }
113         set_data_cgi(&cgi_params, UPDATE_STA);
114
115       }else{
116         return "/404.html";
117       }
118
119       return "/success.html";
120     }
121
122
123     void http_server_init(void)
124     {
125       http_set_ssi_handler(ssi_handler, SSI_TAGS, SSI_TAGS_NUM);
126       http_set_cgi_handlers(FORM_CGI, LWIP_ARRAYSIZE(FORM_CGI));
127     }
```

APÊNDICE F – Árvore de *Clock* da aplicação

Figura 30 – Árvore de *Clock* do projeto



Fonte: Autoria própria (2023).