UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

HERACTO MYCHAJLÓ CHRUSCINSKI VOIGT

USO DO MICROCONTROLADOR ESP32 PARA MONITORAMENTO EOPERAÇÃO DE UM INVERSOR DE FREQUÊNCIA VIA REDE WI-FI.

MEDIANEIRA

2022

HERACTO MYCHAJLÓ CHRUSCINSKI VOIGT

USO DO MICROCONTROLADOR ESP32 PARA MONITORAMENTO EOPERAÇÃO DE UM INVERSOR DE FREQUÊNCIA VIA REDE WI-FI.

Using the Esp32 microcontroller for monitoring andoperating a frequency inverter via a Wi-Fi Network.

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Leandro Antonio Pasa.

MEDIANEIRA

2022

HERACTO MYCHAJLÓ CHRUSCINSKI VOIGT

USO DO MICROCONTROLADOR ESP32 PARA MONITORAMENTO EOPERAÇÃO DE UM INVERSOR DE FREQUÊNCIA VIA REDE WI-FI.

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

Data de aprovação: 25/novembro/2022

Leandro Antonio Pasa
Doutorado
Universidade Tecnológica Federal do Paraná

Alex Lemes Guedes
Mestrado
Universidade Tecnológica Federal do Paraná

Amauri Massochin
Mestrado
Universidade Tecnológica Federal do Paraná

MEDIANEIRA

2022

Dedico esse trabalho a Universidade por ter disponibilizado todo o conhecimento que jamais teria alcançado se não tivesse passado por ela.

AGRADECIMENTOS

A Deus por tudo, a minha Mãe e ao meu Pai por toda a ajuda disponibilizada, ao professor Leandro pela excelente orientação, ao professor Alex pela ajuda com o trabalho, ao Victor pela disponibilização dos laboratórios nos finais de semana, e principalmente ao meu grande amigo e colega de universidade André, pelo auxílio em todo o processo de elaboração do trabalho.

Se eu vi mais longe, foi por estar sobre ombros de gigantes. (NEWTON, 1676).

RESUMO

Com o avanço da tecnologia e a necessidade de comunicação cada vez mais rápida entre máquinas, foi desenvolvido o conceito de "Internet das Coisas" (IoT), sendo uma das finalidades, refere-se ao ser humano poder acompanhar o processo sem ter a necessidade de estar próximo ao local para saber o que está acontecendo. Sendo assim foi criada a ideia de sistemas inteligentes para a troca de informações entre dispositivos que estejam conectados em rede. Na indústria, há vários equipamentos em que a troca de informações dos dispositivos acontecem através da rede RS-485, tendo a supervisão e operação do processo limitada a presença humana direta no local, para que as informações sejam repassadas à frente. Com o intuito de apresentar mais flexibilidade e agilidade ao processo de supervisão e operação, o seguinte trabalho desenvolverá um protótipo para a aquisição de dados e operação de dispositivos comumente utilizados na indústria, sendo o dispositivo em questão, um inversor de frequência, através do uso do microcontrolador ESP32 e o desenvolvimento de páginas web através de linguagens de programação como HTML, C e JavaScript para a aquisição de dados e operação do dispositivo por intermédio de uma Interface Homem Máquina (IHM) que esteja conectada na rede Wi-fi. Com a utilização do protocolo de comunicação Modbus na troca de informações entre dipositivos mestre e escravo, além da apresentação dos resultados obtidos na prática durante o seu funcionamento.

Palavras-chave: microontroladores; controle remoto; monitorizão elétrica no ambiente de trabalho; internet das coisas.

ABSTRACT

With the advancement of technology and the need for increasingly faster communication between machines, the concept of "Internet of Things"(IoT) was developed, being one of the purposes, it refers to the human being being able to follow the process without having the need from being close to the place to know what's going on. Therefore, the idea of intelligent systems for the exchange of information between devices that are connected in a network was created. In the industry, there are several pieces of equipment in which the exchange of information from the devices takes place through the RS-485 network, with supervision and operation of the process limited to direct human presence on site, so that the information is passed on to the front. In order to present more flexibility and agility to the supervision and operation process, the following work will develop a prototype for the acquisition of data and operation of devices commonly used in the industry, being the device in question, a frequency inverter, through the use of the ESP32 microcontroller and the development of web pages through programming languages such as HTML, C and JavaScript for the acquisition of data and operation of the device through a Human Machine Interface (HMI) that is connected to the Wi-Fi network. With the use of the Modbus communication protocol in the exchange of information between master and slave devices, in addition to the presentation of the results obtained in practice during its operation.

Keywords: microcontrollers; remote control; electronic monitoring in the work environment; internet of things.

LISTA DE ILUSTACÕES

Figura 1 – Comunicação utilizando o modelo de transmissão diferencial 20
Figura 2 – Planta do projeto 24
Figura 3 – Diagrama de força do Inversor de frequência 27
Figura 4 – Características da Porta RJ45 do Inversor de frequência ATV312 27
Figura 5 – Dados de placa do motor 28
Figura 6 – Microcontrolador EPS32 38 pinos 29
Figura 7 – Esquema para aquisição dos dados do inversor de frequência através da rede Recommendad Standart-485 (RS-485) 30
Figura 8 – Ilustração do adaptador conector RJ45 vista de cima 31
Figura 9 – Planta do projeto para a comunicação entre inversor de frequência e microcontrolador 32
Figura 10 – Página Web elaborada para ler o status e enviar comandos para o inversor de frequência 41
Figura 11 – Página Web com as três opções de ação para comando 42
Figura 12 – URL's dos comandos enviados para o inversor de frequência 42
Figura 13 – Página web interagindo com o inversor de frequência durante a operação do motor 43
Figura 14 – Planta do projeto com o periférico Realtime Clock (RTC) acoplado 44
Figura 15 – Gráfico de consumo da corrente do motor em relação ao tempo 47
Figura 16 – Gráfico de consumo em kVAh do motor 47
Figura 17 – Dados extraídos em relação a data e hora 48
Figura 18 – Protótipo 49
Figura 19 – Exemplo de uso do conversor online 56
Figura 20 – Exemplo Deslocamento de bits 56
Figura 21 – Menu de configuração 58
Figura 22 – Submenu de configuração Example Configuration 58
Figura 23 – Página 10 do manual 83
Figura 24 – Funções Modbus inversor de frequência 85
Figura 25 – Dados das Funções Modbus inversor de frequência 85
Figura 26 – Eta Status Inversor de frequência 87
Figura 27 – Registrador de Status inversor de Frequência 87

LISTA DE QUADROS

Quadro 1 – Formato da mensagem em Modbus RTU.....22
Quadro 2 – Ajustes dos parâmetros do inversor para comunicação Modbus.....33
Quadro 3 – Formato da mensagem Modbus RTU.....35
Quadro 4 – Frame para STATUS.....36
Quadro 5 – Frame para comando.....36
Quadro 6 – Frame para Shutdown.....37
Quadro 7 – Frame para Switch On.....37
Quadro 8 – Frame Enable Operation.....37
Quadro 9 – Frame para leitura de corrente do inversor de frequência.....38
Quadro 10 – Frame para leitura da frequência do inversor de frequência.....38
Quadro 11 – Frame para escrita da frequência do inversor de frequência.....38
Quadro 12 – Frame referente a resposta da mensagem Modbus do inversor de frequência.....38

LISTA DE ABREVIATURAS E SIGLAS

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IHM	Interface Homem Máquina
IoT	"Internet das Coisas"
IP	Internet Protocol
PEAP	Protected Extensible Authentication Protocol
ROM	Read-Only Memory
RS-485	Recommendad Standart-485
RTC	Realtime Clock
RTU	Remote Terminal Unit
SoC	System On Chip
SRAM	Static Random Acess Memory
TC’s	Transformadores de Corrente
TTL	Transistor-Transistor Logic
UART	Universa Asynchronous Receiver/Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
VCS	Sistema de Contrele de Versão
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
WPS	Wi-Fi Protected Setup

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Objetivos 13
1.1.1 Objetivo geral 13
1.1.2 Objetivos específicos 13
1.2 Justificativa 15
2 REFERENCIAL TEÓRICO 16
2.1 Aquisição de dados 16
2.2 Microcontrolador ESP 32 16
2.3 Ambientes de desenvolvimento 17
2.3.1 Linguagens de programação 18
2.3.2 Rede RS-485 19
2.3.3 Protocolo Modbus RTU 20
3 MATERIAIS E MÉTODOS 23
3.1 Tipo de pesquisa 23
3.2 Planta do projeto 23
3.3 Comunicação 23
3.4 Microcontrolador 24
3.5 Conversor de dados 24
3.6 Página *web* 24
3.7 Armazeamento de dados 24
3.8 Execução do trabalho 25
4 RESULTADOS 26
4.1 Componentes 26
4.1.1 Inversor de frequência 26
4.1.2 Motor 27
4.1.3 Microcontrolador 28
4.1.4 Periféricos 28
4.2 Montagem da planta 29
4.3 Ajuste dos parâmentros do inversor 32
4.4 Comunicação 33

4.4.1 Mensagem Modbus 35
4.5 Conexão com a internet 39
4.5.1 Página Web dados de leitura e escrita 40
4.5.2 Página Web Gráficos 43
5 CONCLUSÕES E TRABALHOS FUTUROS 50
REFERÊNCIAS 52
APÊNDICE A - EXEMPLO ELABORADOS PELO AUTOR 55
APÊNDICE B - CONFIGURAÇÃO PARA SE CONECTAR NA REDE WI-FI DA UNIVERSIDADE 57
APÊNDICE C - CÓDIGO 59
ANEXO A - FLUXOGRAM PARA ACIONAMENTO DO INVERSOR DE FREQUÊNCIA 82
ANEXO B - FUNÇÕES MODBUS INVERSOR DE FREQUÊNCIA 84
B.1 Supervisão e controle em modo de linha 85
ANEXO C - STATUS INVERSOR DE FREQUÊNCIA 86
C.1 Status Word 87

1 INTRODUÇÃO

Com o avanço da tecnologia e a possibilidade de comunicação em tempo real com os dispositivos tecnológicos, as industriais entraram em uma nova etapa, a indústria 4.0, pois, a operação e supervisão remota de processos vem ganhando cada vez mais espaço no mercado de trabalho pela praticidade e flexibilidade que oferecem. No qual, também, abrange um conjunto de tecnologias de ponta ligadas à Internet com o objetivo de tornar os sistemas de produção mais flexíveis e colaborativos (SANTOS *et al.*, 2018).

O termo IoT é um modelo recente na história da tecnologia, porém, o mesmo está sendo levado muito em consideração no decorrer dos últimos anos. A IoT pretende tornar os sistemas mais inteligentes para automação e troca de informações entre dispositivos conectados em rede, ou também, para auxiliar um ser humano a fazer escolhas mais bem informadas com ajuda dos dados disponibilizados pelo próprio dispositivo (RIBEIRO, 2019).

Diante da necessidade de acompanhar os avanços tecnológicos ao redor do mundo, o seguinte trabalho propõe fazer um protótipo para a aquisição de dados e atuação de dispositivos comumente usados em uma indústria, sendo o objeto de interesse do trabalho um inversor de frequência, através de uma rede Wi-fi.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver um protótipo para um sistema de monitoramento e atuação de dispositivos industriais em painéis elétricos via Wi-fi.

1.1.2 Objetivos específicos

Os objetivos específicos deste trabalho consistem nos seguintes tópicos listados abaixo:

- Utilizar o microcontrolador ESP32 para a comunicação wi-fi com o inversor de frequência;
- Realizar a aquisição de dados do inversor de frequência a partir da rede

RS-485 através do protocolo de comunicação modbus *Remote Terminal Unit* (RTU).

- Implementar um sistema de banco de dados para armazenar as informações de operação do inversor de frequência;
- Criar páginas web a fim de disponibilizar os dados para monitoramento e ser possívela operação do inversor de frequência.

1.2 Justificativa

Ter a possibilidade de monitorar e operar os equipamentos industriais de maneira descentralizada sem a necessidade de estar na sala de comando ou em frente ao quadro elétrico.

2 REFERENCIAL TEÓRICO

2.1 Aquisição de dados

A obtenção de dados, de maneira geral, é feita através de periféricos que tem como objetivo medir a grandeza física para qual foram projetados, podendo citar alguns exemplos como Transformadores de Corrente (TC's) que são responsáveis por ficar apresentando os valores de corrente no condutor em tempo real (KONZEN, 2019), ou o termostato, que é responsável pelo monitorando da temperatura (SILVA, 2000). Assim também, um RTC no qual é responsável exclusivamente por gerar dados de tempo, como segundos, minutos, horas, dia, mês e ano (INTEGRATED, 2015).

A coleta e armazenamento de dados apresentados pelos sensores podem ser feitas através de um sistema de aquisição de dados. Comercialmente existem modelos de baixo custo para a aquisição de dados, como Raspberry Pi, Arduino e placas da família ESP. Entre as principais aplicações em que estas plataformas podem ser utilizadas estão aquelas voltadas para controle, IoT e sistemas de monitoramento (PONTES, 2021).

2.2 Microcontrolador ESP 32

O ESP32 é um microcontrolador, com uma plataforma de protipagem com n pinos, o número de pinos irá depender do modelo. O microcontrolador tem disposição de comunicação à distância, como Bluetooth e Wi-fi. Possui baixo consumo de energia elétrica e permite desenvolver aplicações de acesso remoto com a disponibilidade de armazenamento em nuvens, banco de dados, monitoramento e atuação em tempo real de dispositivos (MIRANDA, 2019).

O Microcontrolador é bem maleável, interativo e tem a disponibilidade de ser programado diretamente em qualquer *Integrated Development Environment* (IDE) de preferência do usuário, com a opção de utilizar diversos *Shields* e sensores comumente usados na eletrônica (PONTES, 2021).

O Módulo ESP32 WROOM 32 DEV KIT já possui um regulador de tensão de 3,3V, botões com a opção de reset e load, além de possuir também um conector micro-*Universal Serial Bus* (USB) e um *chip* de interface Serial-USB. A quantidade

de pinos disponíveis para uso são de 38 pinos, no qual cada pino tem uma função específica (MURTA, 2018a).

No geral, os novos modelos de microcontroladores da família ESP32 na versão 4.0 possuem 448 KB de *Read-Only Memory* (ROM) para inicialização e funções principais, 520 KB de *Static Random Acess Memory* (SRAM) no chip, para dados e instruções, além de possuir um RTC interno com disponibilidade de 16 kB de SRAM na memória (ESPRESSIF, 2020).

2.3 Ambientes de desenvolvimento

A IDE é um ambiente de desenvolvimento integrado (SOUZA, 2021), possui vários recursos para deixar a codificação mais simplificada, pois englobam várias conjuntos de ferramentas de desenvolvimento de *software*, como, editor de código, compilador, depurador e vários outros recursos, tudo em um único ambiente de desenvolvimento (LIMA, 2022).

O Visual Studio Code é uma IDE criada pela Microsoft especialmente dedicada para as linguagens de programação C (CARDENAS; JURASK, 2019), suporta a criação de *softwares*, testes, análise da performance de códigos, *debug* entre várias outras funções, além de fazer uso do Bower, o que permite o controle de *frameworks* e bibliotecas relacionados ao JavaScript (PELLIZZONI, 2016).

O ESP-IDF é um *Framework* de desenvolvimento de IoT, ou seja, um conjunto que engloba biblioteas, programas, compiladores e todos os recursos necesssários para o desenvolvimento de uma aplicação. A IDF foi elaborada pela ESPRESSIF que é a estrutura oficial de desenvolvimento para os *System On Chip* (SoC)s das séries ESP32, ESP32-S e ESP32-C (ESPRESSIF, 2022). O *Framework* se destaca por ser totalmente gratuíto e com a plataforma *open source*, tendo melhorias de mananeira contínua e aumentada, abrange vários exemplos de aplicações e aproveita todos os recursos de *software* e *hardware* da família ESP32, sendo a linguagem de uso para a programação, a linguagem C. A grande vantagem é que no GitHub elaborado para essa IDF possível encontrar suportes para problemas relacionados com o *softaware* ou *hardware* (MURTA, 2018a).

O GitHub é uma ferramenta muito popular designada para engenheiros de *softwares*, acomodando mais de 25 milhões de usuários. No geral o GitHub é um serviço com base em nuvem que hospeda um Sistema de Contrele de Versão (VCS)

denominado Git. O que permite que os desenvolvedores colaborem e façam as mudanças em projetos que estejam compartilhados, enquanto assim, todo o registro desse processo se mantêm detalhado durante o seu progresso (ANDREI, 2022).

Existem vários exemplos prontos de códigos com funcionalidades específicas disponibilizados no GitHub, permitindo que a comunidade entusiasta e técnica possa colaborar no seu desenvolvimento, com o objetivo de facilitar a criação de novas extensões e novas funcionalidades (EDSON, 2016).

O uso do ESP32 na inserção dos dispositivos no contexto de IoT foi apresentado em autores como (PONTES, 2021) e (MIRANDA, 2019). E o uso de exemplos retirados diretamente do GitHub para ser utilizado na programação do microcontrolador ESP 32, foi apresentado em trabalhos como (PELLIZZONI, 2016).

2.3.1 Linguagens de programação

A linguagem de programação C é utilizada por diversos tipos de sistemas operacionais e não possui uma área de desenvolvimento que seja totalmente específica para ela, entretanto, a sua difusão aconteceu devido a dar fácil acesso a parte física das máquinas, trabalhando muito bem em conjunto com os *hardwares*, o que possibilita ter acessibilidade a direta programação em microprocessadores e durante a transferência de um código de uma máquina para outra, na maioria dos casos, o código não irá precisar sofrer alterações (CARDENAS; JURASK, 2019).

A linguagem de programação Java se caracteriza por ser uma linguagem de programação de alto nível com a vantagem de ser portável para diversos sistemas operacionais, sendo também, ótima para a aplicação e desenvolvimento de aplicativos que estejam relacionados com a *internet* devido a facilidade em trabalhar com protocolos de uso remoto como o *HyperText Transfer Protocol* (HTTP) (MARAFON, 2006). Ela tem facilidade para trabalhar com programações que necessitem ser orientadas à objetos, além de ser uma multiplataforma, o que faz com que linguagem represente muito fielmente o que acontece na vida real (CARDENAS; JURASK, 2019).

A linguagem *HyperText Markup Language* (HTML) é usada para descrever documentos de uso na *web*, com a característica de usar Tags para destinguir elementos de textos, *hiper-linkslinks*, multimídia, entre outros. Quando o usuário abre uma página ao acessar a *internet*, ele tem disponibilidade das informações

HTML, pois o navegador processa as informações contidas no documento e mostra o resultado final na tela (SILVA, 2011).

A linguagem HTML é bastante usada em conjunto com o javascript, afim de deixar a página HTML mais dinâmica, pois o javascript tem maior rapidez em executar as informações e criar mais *tags* HTML sem ter a necessidade do navegador ser atualizado (SILVA, 2011).

2.3.2 Rede RS-485

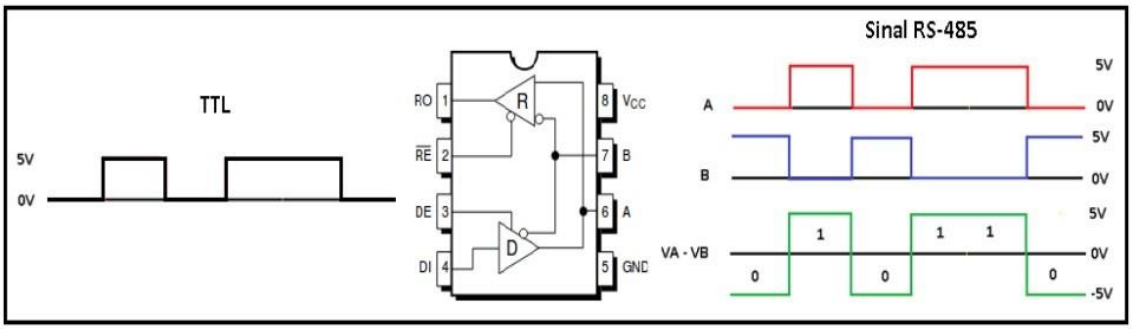
A RS-485 é um padrão da camada física de comunicação, e a principal característica da rede RS-485 é a sua capacidade de atenuação de ruídos (GADELHA, 2019).

O ambiente industrial é bem severo em relação a interferências eletromagnéticas, por isso, a rede RS-485 é amplamente utilizada para a transferência de dados nesses locais (SOUZA, 2005).

Outra vantagem está na capacidade de conectar vários equipamentos em apenas um barramento através das conexões multipontos. Adotar essa estratégia de ligação faz com que a informação fique centralizada em um único dispositivo, o que possibilita a comunicação entre diversos equipamentos conectados na rede. A sua limitação é de até 32 dispositivos conectados na rede, podendo ser um dispositivo mestre e 31 dispositivos escravos (MIRANDA, 2019).

A comunicação entre o microcontrolador ESP32 com a rede RS-485 necessita de um periférico para fazer a conversão dos níveis lógicos de tensão adequados de ambos os dispositivos, para que haja a troca de informações. A figura 1 apresenta a conversão dos protocolos TTL para a rede RS-485 através do chip MAX 485.

Figura 1 – Comunicação utilizando o modelo de transmissão diferencial



Fonte: Gadelha (2019)

O chip MAX 485 é um componente existente dentro do CDT11, que é um conversor bidirecional de dados TTL para o protocolo RS485. O seu modo de funcionamento durante a transmissão de dados acontece através de dois fios denominados de A e B, e por meio de suas polaridades eles determinam qual será o nível lógico da rede. Sendo o dispositivo intermediário no processo de aquisição de dados.

O padrão RS-485 não recomenda a utilização de protocolos desenvolvidos em *softwares*, entretanto, o meio industrial utiliza o protocolo Modbus RTU para a comunicação entre os dispositivos.

2.3.3 Protocolo Modbus RTU

Os protocolos são sistemas desenvolvidos por algortimos com instruções bem definidas para executar uma tarefa (RIOS, 2012). O protocolo Modbus RTU utiliza um arranjo mestre/escravo no processo de comunicação com os dispositivos conectados na rede.

O protocolo tem uma comunicação serial assíncrona, sendo possível ter um único dispositivo mestre, geralmente um computador, e um único dispositivo escravo para que a troca de informações e a conexão da rede seja estabelecida.

O equipamento designado como mestre irá iniciar e finalizar as solicitação de comunicação com o escravo. Que, por sua vez, a limitação máxima de dispositivos RS-485 no barramento depende das unidades de cargas de cada dispositivo, o que varia entre 32 e 256, se desconsiderar o aumento de carga causado pelos resistores de BIAS.

O mestre também responsável por obter as informações dos dispositivos escravos, além da capacidade de gravar nos seus registros.

Os dispositivos que possuem o padrão de meio físico RS-485 normalmente podem fazer uso do padrão de comunicação RTU.

A maneira que se inicia a transmissão irá definir a codificação da informação trasmitida. Ao usar o modo de transmissão RTU em uma linha de transmissão serial MODBUS, os dispositivos se comunicam através de mensagens formadas por *bytes*. Cada *byte* em uma mensagem possui dois caracteres hexadeximais e o seu formato é designado de *frame*, sendo composto por 8 *bits* de dados, 1 start *bit*, 1 *bit* de paridade e 1 stop *bit*, ou 2 stop *bits* quando não há *bit* de paridade (PONTES, 2021). Para checar se a mensagem que foi recebida é a mesma que foi enviada é utilizado o CRC.

O CRC é uma mensagem binária que verifica se realmente o recepetor recebeu a informação correta. Caso aconteça alguma falha, a mensagem recebida não será a mesma da enviada.

O CRC-16 é um código atualizado para verificação de erros em transmissão de dados (SALERNO *et al.*, 2005). Esse código oferece uma taxa de precisão de 99,997% e o seu funcionamento se baseia em cálculos ao deslocar os *bits* em OU-Exclusivo de uma determinada palavra de código binário. Quando a mensagem é transmitida pelo mestre ou pelo escravo, o CRC é concatenado à mensagem, ao chegar no dispositivo receptor, novamente é submetido a cálculos para a verificação. Caso esteja incorreta uma mensagem de erro é enviada. Esse processo garante uma comunicação de confiança entre o *software* e o *hardware*

Os dispositivos Modbus utilizam na sua comunicação quatro tabelas de dados para facilitar a comunicação entre si.

O Modo RTU é caracterizado por ordenar uma faixa de tempo para o começo e término da mensagem, em vez de usar caracteres especiais para essa finalidade. O monitoramento dos escravos consiste em estar esperando por um tempo que pode variar de 3 a 5 *bytes* transmissores de silêncio (T1-T2-T3-T4) (CUNHA, 2000) logo para o início e término da mensagem, ao decorrer desse intervalo, o dispositivo consegue identificar que os próximos dados serão de endereço do destinatário acompanhado pelo restante da mensagem, até a repetição do intervalo ocorrer novamente. A Tabela 1 mostra um exemplo do formato da

mensagem do Modbus em RTU.

Quadro 1 – Formato da mensagem em Modbus RTU

INÍCIO	ENDEREÇO	FUNÇÃO	DADOS	CRC	FIM
T1,T2,T3,T4	8 BITS	8 BITS	N X 8 BITS	16 BITS	T1,T2,T3,T4

Fonte: Adaptado de Cunha (2000)

Em resumo, para manter uma comunicação eficaz entre mestre e escravo, é necessário que os paramêtros estejam bem definidos, como o formato dos caracteres, taxa de transmissão e principalmente o ID do escravo. Se houver uma incompatibilidade dentros desses parâmetros, ocorrerá falhas no processo de comunicação.

.

3 MATERIAIS E MÉTODOS

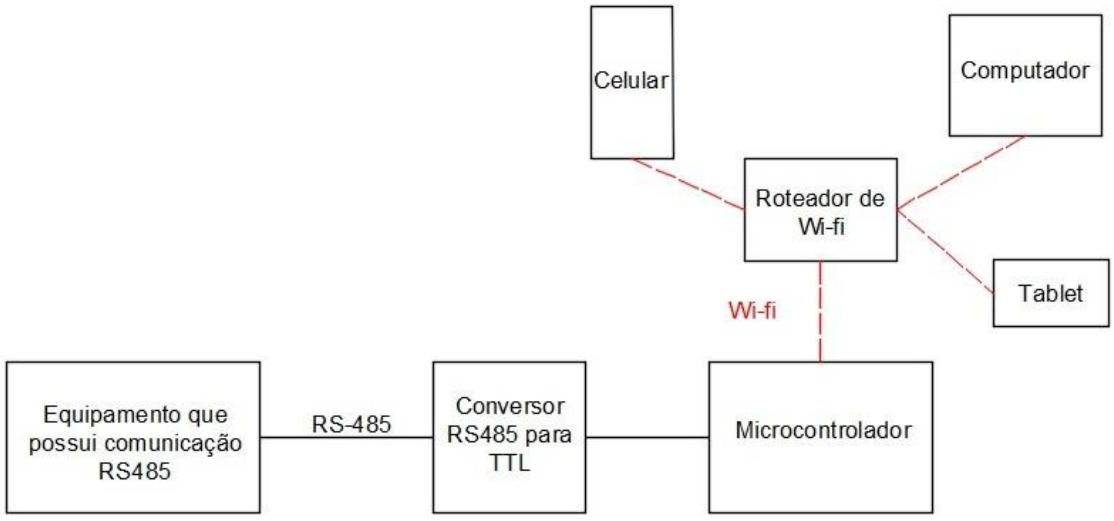
3.1 Tipo de pesquisa

O tipo de pesquisa nesse trabalho se constituirá da pesquisa aplicada, pois o objetivo é a aplicação imediata do conhecimento na resolução prática de um desafio envolvendo processos de acionamento de um motor e aquisição de dados referente a seu funcionamento de maneira remota.

3.2 Planta do projeto

A ideia básica da planta é poder extrair informações e mandar comandos para um inversor de frequência que possua comunicação RS-485, afim de controlar e obter dados de um motor trifásico, via Wi-fi. Através de uma IHM por intermédio do microcontrolador ESP32, como observa-se na Figura 2. O inversor de frequência que será utilizado é o Altivar 312, cujo códigodo produto é ATV312H075M2.

Figura 2 – Planta do projeto



Fonte: Autoria própria (2022)

3.3 Comunicação

A comunição utilizará o protocolo Modbus RTU e os comandos de leitura e escrita serão enviados por intermédio de uma página *web*. O meio físico de comunicação com o inversor de frequência se dará através da rede RS-485 e

dependerá de um conversor de sinal bidirecional para poder enviar e receber as mensagens. Convertendo o sinal da rede RS-485 para a rede *Transistor-Transistor Logic* (TTL) e de TTL para RS-485.

O microcontrolador ESP32 não consegue implementar a rede RS-485 diretamente sema ajuda do periférico de conversão.

3.4 Microcontrolador

O microcontrolador que será utilizado no projeto de pesquisa é o ESP32. A programação do componente será feita utilizando a lingagem de programação C. Ele se tornará um servidor e fornecerá uma página *web* ao obter um endereço IP do roteador que ele se conectar, com isso qualquer dispositivo com acesso ao Wi-fi da rede local poderá ter acesso aos dados fornecidos pelo inversor de frequência, desde que possua o endereço da página para acessá-la.

3.5 Conversor de dados

O conversor que será utilizado é o CDT11, pois possui a propriedade de conversão bidirecional. Sendo um componente relativamente barato e de fácil acesso.

O CDT11 faz a conversão dos níveis de tensão do microcontrolador para níveis adequados de tensão da rede RS-485. Os pinos de entrada/saída que serão utilizados para se comunicar com o microcontrolador são: RXD;TXD;GND e VCC e os pinos de saída/entrada que serão utilizados para se comunicar com o elemento escravo são: A;B e GND.

3.6 Página *web*

A página *web* será disponibilizada através do próprio microcontrolador, ao gerar um endereço IP quando utilizar o roteador como meio de conexão Wi-fi.

Os dispositivos que estiverem conectados na rede Wi-fi poderão acessar os dados e enviar comandos a partir da *Uniform Resource Locator* (URL) definida pelo programador.

A página *web* será desenvolvida principalmente com a linguagem HTML e

JavaScript, e disponibilizará três URL's para acesso, sendo uma com a possibilidade de ligar o motor no sentido direto de giro, assim como no sentido reverso, desligar o motor e fornecer dados de corrente, frequência de operação e *status* de funcionamento (ON/OFF). A segunda página *web* fornecerá dados da maior corrente utilizada em relação ao tempo e a terceira fornecerá dados de consumo do motor em KVAh.

3.7 Armazeamento de dados

Os dados poderão ser verificados através dos gráficos e estarão no histórico de execução do terminal do Visual Code Studio conforme o programa estiver sendo executado. A cada leitura que o inversor de frequência fizer, os dados estarão aparecendo em formato de texto e separados por ";", afim de auxiliar na identificação dos mesmos. Referente aos dados de tempo, o microcontrolador estará obtendo eles através do seu RTC interno, após ter sido configurado com o auxílio de um RTC externo pela primeira vez, cuja a principal finalidade do RTC externo é contabilizar dados de tempo, no trabalho será contabilizado dados de segundos, minutos, horas, dia, mês e ano.

3.8 Execução do trabalho

A primeira parte do trabalho mostrará todo o processo para tornar possível a interação do microcontrolador com o inversor de frequência através de uma página *web* ao utilizar o protocolo de comunicação modbus para ler e escrever as informações de interesse.

A segunda parte e final do trabalho mostrará a criação de gráficos em relação ao tempo com o auxílio de um RTC externo e o RTC interno do microcontrolador.

4 RESULTADOS

4.1 Componentes

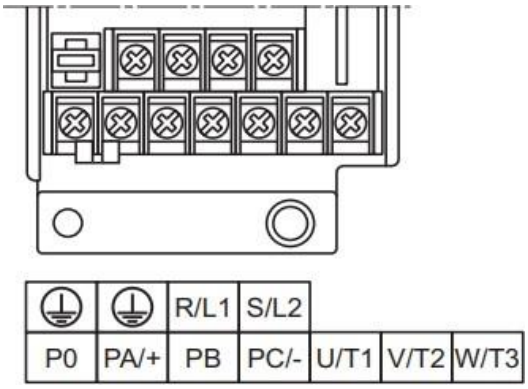
4.1.1 Inversor de frequência

O inversor de frequência utilizado é o Altivar 312, os modelos são projetados para motores trifásicos com potência entre 0.18 a 15kW, para esse projeto, o modelo em uso é o ATV312H075M2 no qual é indicado para motores com potência de saída de placa de até 1CV ou 0,75kW.

A alimentação do inversor de frequência para o seu funcionamento adequeado está entre as faixas de tensões de 200V-240V. A energização do componente em uma rede 110/127V se dá através de duas fases e a saída do inversor devolve para o motor uma alimentação trifásica em 220V, possibilitando o funcionamento de motores trifásicos.

O diagrama para alimentação do inversor de frequência pode ser observado figura 3.

Figura 3 – Diagrama de força do Inversor de frequência

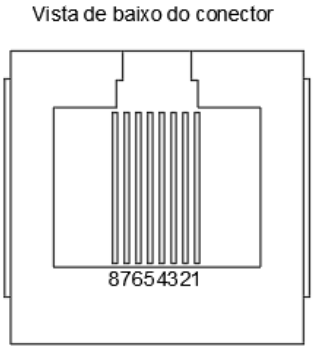


Fonte: Schneider (2016)

A energização do inversor de frequência é através dos bornes R/L1 e S/L2, e a saída atavés dos bornes U/T1, V/T2 e W/T3. Os demais bornes de força não serão utilizados no desenvolvimento do projeto.

O inversor de frequência também tem a disponibilidade de uma porta RS-485, como observa-se na figura 4.

Figura 4 – Características da Porta RJ45 do Inversor de frequência ATV312



PIN	SIGNAL
1	CANopen signal reserved
2	CANopen signal reserved
3	CANopen signal reserved
4	D1 Modbus signal
5	D0 Modbus signal
6	Not connected
7	VP (1)
8	Common Modbus signal

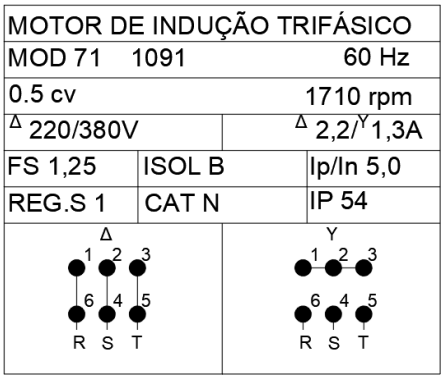
Fonte: Adaptado de Schneider (2009b)

Por meio dessa porta que há a possibilidade de estabelecer a comunicação entre o inversor de frequência e o microcontrolador ESP32 por intermédio de um conector RJ45. Todos os protocolos de comunicação serão enviados pela rede RS-485 e devolvidos para o microcontrolador pela mesma, devido a característica da rede RS-485 ter comunicação bidirecional.

4.1.2 Motor

O motor de uso neste trabalho é um motor trifásico de 0.5CV ou 0.373kW de potência, a sua alimentação será em delta 220V e os dados de placa podem ser observados na figura 5.

Figura 5 – Dados de placa do motor

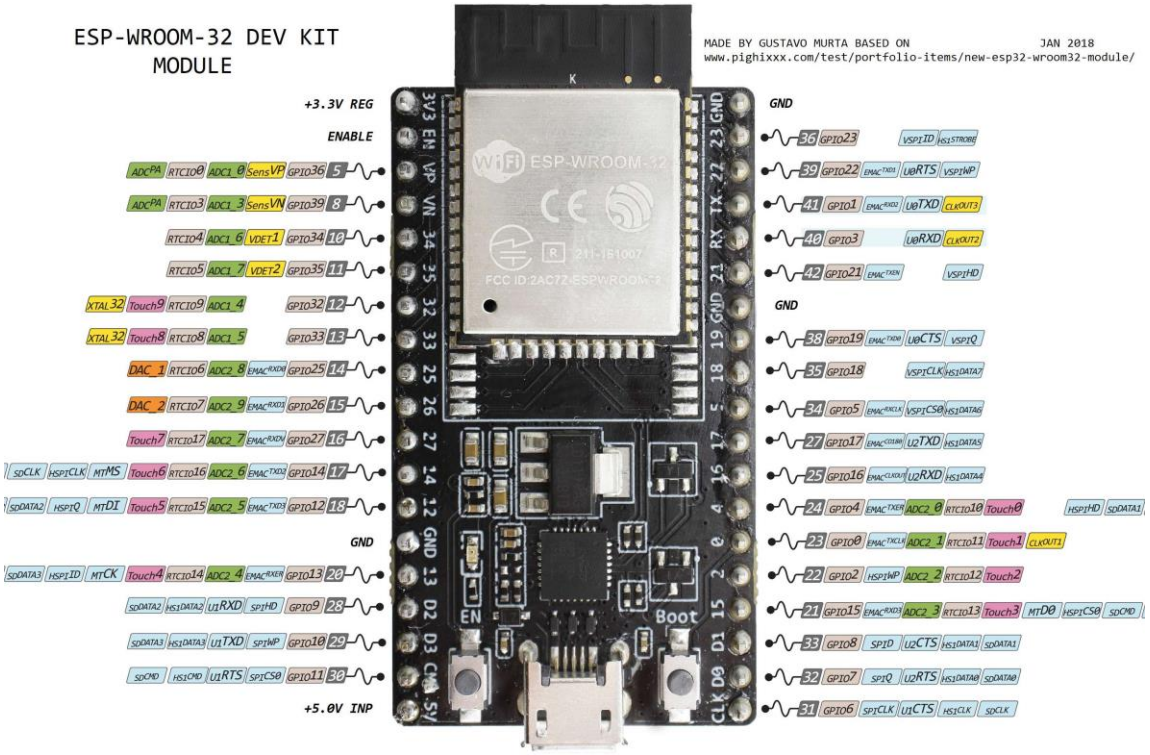


Fonte: Autoria própria (2022)

4.1.3 Microcontrolador

O microcontrolador que será utilizado é o ESP32 WROOM 32 DEV KIT, o modelo para o projeto em si conta com 38 pinos e o propósito de cada pino pode ser observado na figura 6.

Figura 6 – Microcontrolador EPS32 38 pinos



Fonte: Murta (2018b)

A sua escolha foi planejada devido a possibilidade da comunicação Wi-fi com outros dispositivos, além também, de ser possível a interação com periféricos comumentes utilizados na eletrônica.

4.1.4 Periféricos

DS3231 é um relógio de tempo real extremamente preciso, no qual ele dispõe de uma bateria que ajuda a manter a contagem do tempo mesmo depois que a fonte de alimentação tenha sido interrompida. Ele mantém as informações de segundos, minutos, horas, dia, data, mês e ano guardadas em sua memória, após esses parêmetros terem sidos ajustados pela primeira vez pelo programador.

A sua utilização na planta é exclusivamente para configurar uma única vez o RTC interno do microcontrolador.

CDT11 é um conversor de dados da rede TTL para a rede RS-485 e vice-versa, devido ao utilizar o chip MAX485ESA em sua composição. Esse é o conversor que possibilita a comunicação entre o inversor de frequência e o microcontrolador.

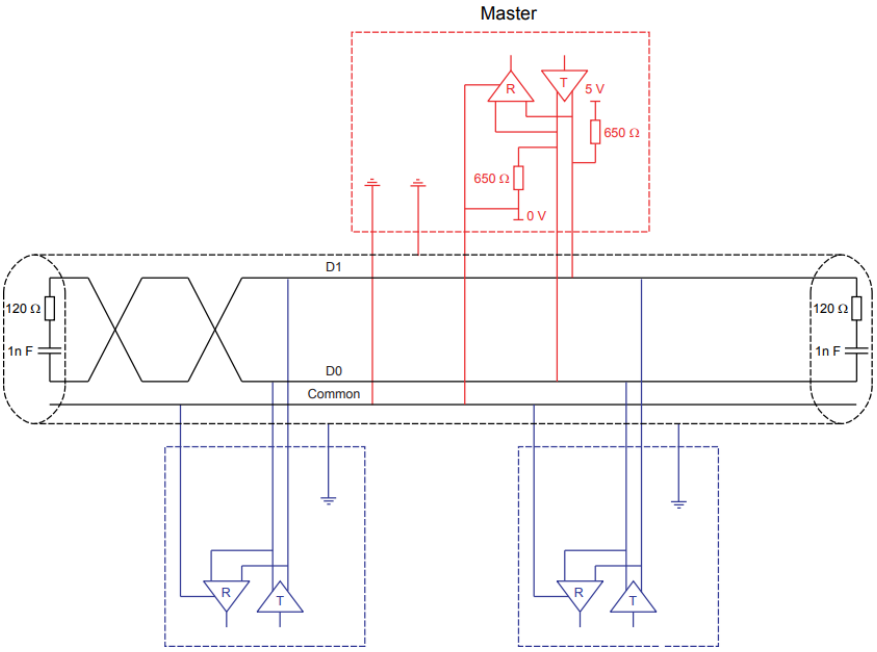
4.2 Montagem da planta

Antes de integrar os componentes da planta do projeto é necessário tomar nota dos pinos do microcontrolador e para que eles serão utilizados em conjunto com os outros componentes, afim de tornar possível todas as interações para garantir uma comunicação correta e eficaz.

O microcontrolador utiliza a comunicação serial, para isso é necessário fazer uso das portas do microcontrolador que possuam a comunicação UART. Para esse projeto foram utilizadas as portas 16 (RX) e 17 (TX), pois para esse tipo de comunicação é necessário apenas dois fios para a transmissão de dados entre duas UARTS, no caso, os dados fluirão do pino TX da UART transmissora para o pino RX da UART receptora.

Ao começar pela aquisição de dados do inversor de frequência é importante levar em consideração como funciona a obtenção desses dados. Inicialmente tudo ocorre através da rede RS-485 pelo conector RJ45 conforme observado na figura 4. O Esquema para a obtenção dos dados do inversor de frequência pode ser visto na figura 7.

Figura 7 – Esquema para aquisição dos dados do inversor de frequência através da rede RS-485

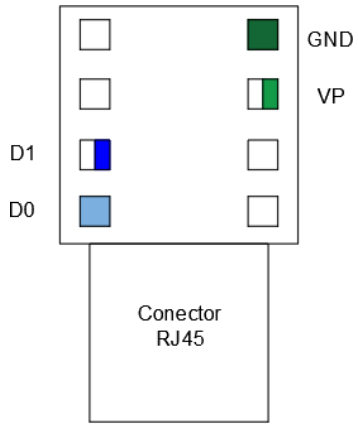


Fonte: Schneider (2009b)

A comunicação entre o microcontrolador e o inversor de frequência utilizará o protocolo de comunicação Modbus RTU, para isso é necessário utilizar dois fios como mostrado na figura 7 o D1 e D0 para a troca simultânea de informações.

Para facilitar a montagem do protótipo foi adquirido um adaptador que possibilita utilizar os fios do conector RJ45 sem precisar modificar o cabo, em outras palavras, não seria necessário cortar o mesmo para separar apenas os fios necessários para a comunicação, mantendo o cabo apto para ser utilizado em outra aplicação a qual não fosse apenas para a execução do projeto. O adaptador faz com que os fios do cabo possam ser utilizados de maneira livre, os pinos de interesse para serem usados como foi mostrado na figura 4, são os pinos 4,5,7,8 respectivamente se referem aos sinais D1,D0,VP e comum ou gnd. Na figura 8 pode-se observar quais sinais dos pinos vão estar em cada borne.

Figura 8 – Ilustração do adaptador conector RJ45 vista de cima



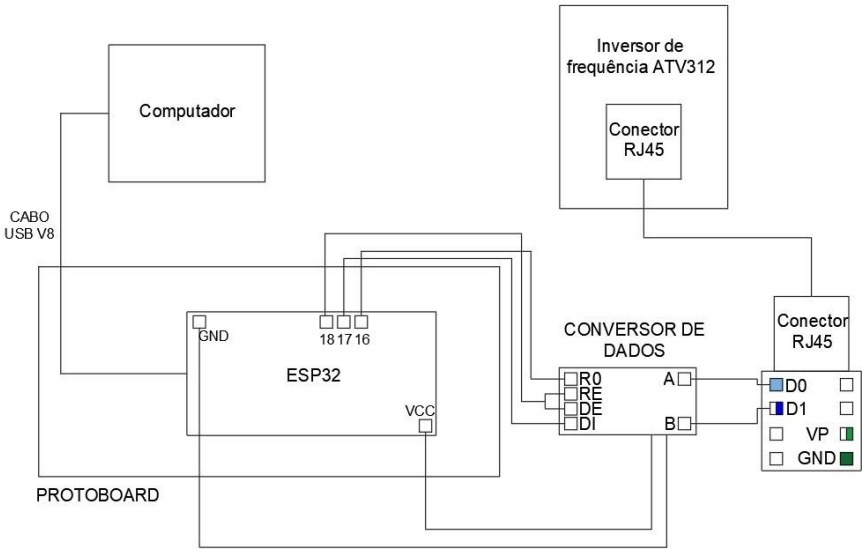
Fonte: Autoria própria (2022)

Tendo em posse os fios D1 e D0 da rede RS-485 é obrigatório que a informação dos mesmos sejam convertidos para a rede TTL, afim de realizar a comunicação serial corretamente. Para isso foi utilizado o conversor de dados com posse do chip MAX485ESA, no qual desempenha esse papel.

Com isso, a comunicação entre o microcontrolador e o inversor de frequência já se torna possível, a montagem da planta ilustrada até esse processo pode ser observada na figura 9.O pino 18 também está em uso, por fazer o controle de fluxo de dados, pois durante a comunicação, dois dispositivos não podem enviar mensagens simultâneamente, o microcontrolador através desse pino alterna o envio e recebimento de dados de maneira automática.

A comunicação com o inversor de frequência utilizará o protocolo de comunicação Modbus RTU, pois o mesmo já incorpora em si esse protocolo, porém antes de programar o microcontrolador para que ele faça comandos de leitura e escrita no inversor de frequência é necessário ajustar os parâmetros do inversor para que ele aceite esse tipo de comunicação.

Figura 9 – Planta do projeto para a comunicação entre inversor de frequência e microcontrolador



Fonte: Autoria própria (2022)

4.3 Ajuste dos parâmentros do inversor

Os parâmentros do inversor podem ser ajustados manualmente através do seu botão de pressão rotativo localizado no centro do dispositivo. Para acessar as configurações basta darum clique e girar para a opção que você deseja configurar.

A tabela 2 mostra o passo a passo para ajustar o inversor de frequência para ser possível a interação com o microocontrolador através do protocolo de comunicação Modbus.

Quadro 2 – Ajustes dos parâmetros do inversor para comunicação Modbus

Parâmetro	Menu	Opção	Ajuste
Modbus Boud Rate	CON	Add	19.2
Modbus Address	CON	tbr	1
Modbus Format	CON	tfo	8E1
Modbus TimeOut	CON	tto	10s
Ref 1. Channel	Ctl	Fr1	Mdb
Ref 2. Channel	Ctl	Fr2	Mdb
Acess Level	Ctl	LAC	L3
Cmd Channel 1	Ctl	Cd1	mdb
Cmd Channel 2	Ctl	Cd2	mdb
HMI Comand	Ctl	LCC	YES
Summing Ref 2	Fun	SA2	mdb
Summing Ref 3	Fun	SA3	mdb

Fonte: Autoria própria (2022)

Alguns parâmetros poderiam estar ajustados para não ser através da comunicação Modbus, entretando, como a meta é interagir com o inversor de frequência de maneira remota, todos os parâmetros foram ajustados para tal. Há alguns pontos importantes, como a taxa de transmissão de dados (*Boud Rate*), segundo o manual do dispositivo, para comandos provenientes de interfaces remotas, é o obrigatório que esteja a uma taxa de transmissão de 19.200kbps. O formato da mensagem Modbus escolhido foi o de oito (8) dados de *bits* com paridade par e um *bit* de stop, pois também, esse é o formato suportado para comandos provenientes de interfaces remotas.

4.4 Comunicação

A programação do microcontrolador foi feita no Visual Studio Code, mas ela pode ser executada em qualquer IDE de preferência do usuário. Vale ressaltar que na plataforma oficial de desenvolvimento da estrutura do microcontrolador ESP32 (ESP-IDF) existem bibliotecas e ferramentas disponíveis para *download* que auxiliam na programação, o *framework* da ESP-IDF foi utilizada no projeto em conjunto com exemplos disponíveis no GitHub e adaptou para as necessidades do trabalho. O acesso aos exemplos no GitHub estão disponíveis a partir do seguinte domínio:

- **Link: GitHub.** JACK0C (2022)

Primeiramente é essencial instalar o *framework* da Espressif contendo o conjunto de ferramentas e bibliotecas do ESP-IDF afim de conseguir trabalhar com os exemplos disponibilizados, para fazer isso, no terminal de desenvolvimento Visual Studio Code, basta pesquisar no guia extensões "ESP IDF"e clicar para iniciar a instalação.

Para inicializar a comunicação *Universal Asynchronous Receiver/Transmitter* (UART) e ter o controle de fluxo de dados foi utilizado o exemplo disponível no GitHub através do seguinte domínio:

- **Link: GitHub-UART.** KAYA (2022)

Através desse exemplo é possível configurar quais portas do microcontrolador serão utilizadas para realizar a comunicação UART com os periféricos, e também vai garantir que a comunicação entre o computador e o microcontrolador ocorra corretamete.

O modo de transmissão de dados escolhido foi o RS485 Half-Duplex, afim de garantir que o fluxo de dados seja apenas para um lado, ou seja, quando o mestre enviar um comando os escravos entram em modo de recebimento de mensagem, e quando os escravos decidirem enviar uma mensagem, o mestre entra em modo de recepção, fazendo assim, com que não aconteça a colisão de dados.

A parte da programação que possibilita esse meio de transmissão de dados pode ser verificada no Apêndice C na linha 135 do código.

A taxa de transmissão de dados e o formato da mensagem Modbus devem estar configuradas de acordo com os parâmetros definidos no inversor de frequência conforme mostrado na tabela 2, a taxa de transmissão de dados pode ser definida através da linha 129 do código disponível no Apêndice C.

A quantidade de *bits* de dados, a paridade e o *bit* de parada então definidas entre as linhas 193 e 195 do código disponível no Apêndice C.

Com o controle de fluxo definido e o formato da mensagem Modbus configurada, tanto no inversor de frequência quanto no microcontrolador, já é possível iniciar a comunicação entre os dispositivos, para isso é necessário saber

como e quais *frames* serão enviados na mensagem Modbus.

Os *frames* necessitam estarem sendo enviados de maneira periódica para que o inversor de frequência não apresente perda de comunicação durante o seu funcionamento.

4.4.1 Mensagem Modbus

O formato de transmissão da mensagem é através do modo RTU, a tabela 3 informa a maneira que os dados devem ser enviados.

Quadro 3 – Formato da mensagem Modbus RTU

ENDEREÇO DO ESCRAVO	CÓDIGO DE SOLICITAÇÃO	DADOS	CRC16

Fonte: Adaptado de Schneider (2009b)

Após tomar nota do *frame*, é necessário saber quais comandos devem ser enviados para realizar as operações de leitura e escrita em registradores, as funções do Modbus referente ao inversor de frequência se encontram no Anexo B

Assim serão utilizados o código 3 (0x03) para a leitura e o código 6 (0x06) para a escrita. Ao saber dos endereços e comandos necessários, necessitasse dos endereços dos registradores, no qual podem ser vereficados no Anexo B.1.

Com isso já é possível saber quais valores de comandos que é preciso ser escrito noregistrador W8501 (0x2135). O valor que deve ser enviado para habilitar a operação é o 0x0007. Para conseguir ler o registrador de *STATUS* do inversor de frequência, a imagem referente ao registrador *STATUS* se encontra no Anexo C, precisa-se ler o registrador de endereço 0x0C81, ao tomar conhecimento do conteúdo desses registradores pode-se saber qual comando deve ser enviado para o registrador de comando, afim de avanças as etapas no fluxograma disponível no Anexo A. Para ler o *STATUS* do inversor, deve-se enviar o *FRAME* apresentado na tabela 4.

Quadro 4 – *Frame* para *STATUS*

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC16
0x01	0x03	0x0C81	0x0001	0xD772
1 byte	1 byte	2 bytes	2 bytes	2 bytes

Fonte: Autoria própria (2022)

Em que 0x01 é o endereço do escravo, 0x03 é o comando para leitura, 0x0C81 é o endereço do registrador *ETA STATUS*, 0x0001 fornece o número 1 que é a quantidade de registradores que devem ser lidos, por fim, o CRC16 que é calculado.

Um comentário importante quanto ao CRC16 utilizado no formato de mensagem Modbus que foi apresentado na Tabela 4 e que serão apresentados nas próximas tabelas. O formato da mensagem CRC16 é em *little endian*, ou seja, é enviado primeiramente o byte menos significativo na mensagem. E para encontrar o valor correto do CRC16 foi utilizado um conversor *online* disponível no seguinte domínio:

- **Link: Conversor Online.** CRCCALC (2022)

Exemplo de como pode-se estar utilizando o conversor *online* vai estar disponível no Apêndice A.

Pra escrever comandos que alteram o estado do fluxograma disponível no Anexo A, deve-se escrever os *frames* conforme a tabela 5.

Quadro 5 – *Frame* para comando

0x01	0x06	WORD	CRC16
1 byte	1 byte	2 bytes	2 bytes

Fonte: Autoria própria (2022)

Em que 0x01 é o endereço do escravo, 0x06 é o comando para escrita, 0x2135 é o endereço do registrador de comandos, WORD é o conteúdo que deve ser enviado para o registradore CRC16 é o valor do CRC em *little endian*.

Considerando os comandos que devem ser enviados, supõe-se que se inicia no momento em que o inversor de frequência recebe a alimentação da rede. Para

que seja enviado algum comando para o inversor de frequência, é necessário que o inversor esteja no estágio 3 do fluxograma. Afim de confirmar isso, é necessário ler o registrador ETA Status e caso o mesmo possua a resposta 0x0040 após a máscara 0x006F ser aplicada, significa que é possível transitar de etapa. Para isso, é preciso enviar o *Frame* para *STATUS* e analisar a sua reposta, caso esteja na etapa 3, deve-se enviar o comando 0x0006 (*Shutdown*) para que o inversor de frequência passe para a etapa 4. A mensagem Modbus para o comando *Shutdown* pode ser verificada conforme a tabela 6.

Quadro 6 – *Frame* para *Shutdown*

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC-16
0x01	0x06	0x2135	0x0006	0x13FA
1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Autoria própria (2022)

Após enviar o comando *Shutdown*, o inversor transitará para a etapa 4. Para cada etapaé necessário enviar o *frame* para *STATUS*, portanto assume-se que após enviar um *Frame* para comando, será sempre confirmado se o inversor transitou para a nova etapa. Após o inversorse estabelecer na etapa 4, deve ser enviado o comando *Switch On*, de acordo com a tabela 7.

Quadro 7 – *Frame* para *Switch On*

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC-16
0x01	0x06	0x2135	0x0007	0xD23A
1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Autoria própria (2022)

Com isso, será alcançada a etapa 5, portanto deve ser enviado o *frame Enable Operation* de acordo com a tabela 8.

Quadro 8 – *Frame Enable Operation*

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC-16
0x01	0x06	0x2135	0x000F ou 0x0008	0xD3FC ou 0xD43C
1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Autoria própria (2022)

Os bytes podem mudar conforme o sentido de giro desejado, um formato para o sentido direto e outro formato para o sentido inverso, por isso os dados transmitidos mudam e o CRC precisa ser recalculado.

Com o inversor pronto para entrar em operação, foram definidos comandos de leitura para corrente e frequência e um comando de escrita para a frequência nominal de rotação do motor para que ele operasse a 60Hz. Os comandos através da mensagem Modbus podem ser verificados nas tabelas 9, 10 e 11.

Quadro 9 – *Frame* para leitura de corrente do inversor de frequência

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC-16
0x01	0x03	0x0C84	0x0001	0xC773
1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Autoria própria (2022)

Quadro 10 – *Frame* para leitura da frequência do inversor de frequência

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC-16
0x01	0x03	0x2136	0x0001	0x6E38
1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Autoria própria (2022)

Quadro 11 – *Frame* para escrita da frequência do inversor de frequência

Endereço do Escravo	Código de solicitação	Dados	Dados	CRC-16
0x01	0x06	0x2136	0x0258	0x6362
1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Autoria própria (2022)

No inversor de frequência, o formato da mensagem Modbus tem dois formatos, os formatos que foram apresentados nas tabelas até agora neste trabalho, são referente a comunicação Modbus para solicitação, quando a resposta é enviada pelo escravo ela tem o formato apresentado na tabela 12.

Quadro 12 – *Frame* referente a resposta da mensagem Modbus do inversor de frequência

Endereço do Escravo	Código de solicitação	Números de bytes lidos	Primeira Word	Última Word	CRC-16
1 byte	1 byte	1 byte	2 bytes	2 bytes	2bytes

Fonte: Adaptado de Schneider (2009b)

A tabela 12 vai ajudar a esclarecer como foi implementada na programação do códigono microcontrolador ESP32 para retirar informações específicas que cada

bit representa.

Das tabelas para solicitação que foram apresentadas, a maneira que pode estar criando as mensagens Modbus afim de se comunicar com o inversor de frequência por meio da programação, pode ser verificada entre as linhas 246 e 388 do código disponível no Apêndice C.

Para extrair os dados dos *frames* de respostas enviadas pelo escravo foi criada uma função que recebe como parâmetro, qual é o registrador que foi lido, isso possíbilita extrair informações específicas que cada *bit* da mensagem Modbus carrega. Entre as linhas 390 e 440 do código disponível no Apêndice C, vão estar lendo as repostas dos registradores relacionados a corrente, status e frequência.

Ao usar como exemplo a parte do código entre as linhas 425 e 428 disponível no Apêndice C para ler os registradores de status, ele é composto por 16 *bits* de dados e cada *bit* carrega uma informação específica, essas informações podem ser verificadas no Anexo C.1. O *bit* de interesse para verificar se o motor está em operação ou não é o *bit* 2: *Operation Enable*, e também, é o *bit* que mostra se o motor está no sentido de giro direto, pois quando o motor entra em operação, ele já começa nesse sentido de giro. Para fazer a verificação foi feita o deslocamento de *bits*, justamente para pegar essa informação específica, detalhes do que o código está executando pode ser verificado no Apêndice A, figura 20.

Verificado a eficácia da comunicação Modbus, o trabalho seguiu para a elaboração da comunicação através da rede Wi-fi.

4.5 Conexão com a internet

O ESP32 é um servidor HTTP para os aparelhos conectados na rede Wi-fi local ou externa, para esse projeto no caso, não foi desenvolvido a possibilidade de um usuário externo à rede Wi-fi local estar interagindo com o inversor de frequência.

Para habilitar a comunicação através da rede Wi-fi foram utilizadas duas maneiras de se conectar na rede, na rede comum foi utilizado o protocolo de segurança *Wi-Fi Protected Access* (WPA)/*Wi-Fi Protected Setup* (WPS) e na rede da universidade foi utilizado o protocolo *Wi-Fi Protected Access 2* (WPA2) Enterprise, ambos os exemplos se encontram no GitHub através dos respectivos domínios:

- **Link: GitHub Wi-fi.** YDESP (2022)
- **Link: GitHub Wi-fi Enterprise.** HFUDEV (2022)

Para se conectar na rede comum é necessário informar para o microcontrolador o nome da rede e a senha da mesma na qual deseja-se conectar, o que pode ser verificado no Apêndice C entre as linhas 95 e 96 do código.

Para verificar se a conexão com a rede Wi-fi foi estabelicida existem funções *handler* que ficam administrando o processo afim de conseguir realizar operações de acordo com o que acontece com o microcontrolador, no caso será possível realizar uma ação de mostrar que o microcontrolar está conectado no Wi-fi, detalhes desse processo pode ser verificado no Apêndice C entre as linhas 986 e 1018 do código.

Para se conectar na rede Wi-fi da universidade precisa utilizar o protocolo WPA2 En- terprise e informar o nome da rede, nome do usuário e senha, detalhes do processo pode ser verificado no Apêndice B. Após a configuração para acessar a rede Wi-fi, já é possível elaborar uma página *web* no microcontrolador para poder interagir com o inversor de frequência de maneira remota.

4.5.1 Página Web dados de leitura e escrita

As páginas Webs foram feitas utilizando programação HTML e JavaScript. A primeira página *web* que envia comandos e lê informações do inversor de frequência foi feita utilizando como base um exemplo público disponível em um compilador *online*, e ajustada conforme as necessidades do projeto. o compilador *online* com o respectivo exemplo se encontra no seguinte domínio:

- **Link: Compilador Online.** W3SCHOOLS (2022)

A segunda e terceira página *web* que são responsáveis por mostrar os gráficos do projeto foram feitas utilizando como base um exemplo público e ajustado conforme as necessidades do projeto. O exemplo se encontra no seguinte domínio:

- **Link: Base para gráficos.** USEFULANGLE (2016)

Inicialmente é preciso registrar as páginas *webs* com seus respectivos endereços no microcontrolador para ser possível acessar elas. A parte do código que mostra esse processo está disponível no Apêndice C entre as linhas 919 e 944. A página designada como "hello"é a página que permite enviar os comandos para o inversor de frequência, a página "dados"é a responsável por mostrar os dados da maior corrente no motor em relação ao tempo e a página "consumo"é a responsável por mostrar o consumo do motor em kVAh.

A página "hello", possui um combo box com três opções, com isso é possível selecionaro comando que será enviado para o inversor de frequência. As opções disponíveis são as de ligar no sentido direto, reversor e desligar o motor, além de possuir a combo box com as opções, a página "hello"mostra o status de funcionamento do motor (ON/OFF) e apresenta dados de corrente e frequência, como pode observar nas figuras 10 e 11.

Figura 10 – Página Web elaborada para ler o status e enviar comandos para o inversor de frequência

Fonte: Autoria própria (2022)

Figura 11 – Página Web com as três opções de ação para comando

Fonte: Autoria própria (2022)

Os dados são atualizados dentro de uma faixa de tempo, mas para visualizá-los é necessário ficar atualizando a página *web*, pois não foi implementada uma solução neste trabalho na qual a página web se atualizasse sozinha.

Ao selecionar o comando e clicar no botão de enviar as informações, o cliente será remetido a uma página com uma URL específica, o qual depende da opção que foi selecionada. Para isso foi criada uma função específica para obter a URL ao qual o cliente foi remetido, em outras palavras, a URL mostrará qual opção o cliente selecionou, como observa-se na figura 12. Essa função vai estar entre as linhas 201 e 235 do código disponível no Apêndice C. Com isso, depois que houve a identificação do comando enviado através da página *web*, o microcontrolador vai enviar a respectiva mensagem em formato Modbus para o inversor de frequência.

Figura 12 – URL's dos comandos enviados para o inversor de frequência

Fonte: Autoria própria (2022)

A função que configura a página *web* "hello"está entre as linhas 841 e 917 do código,disponível no Apêndice C.

Para acessar a página *web* é necessário utilizar o *Internet Protocol* (IP) que

o microcontrolador obteu quando entrou na rede Wi-fi após ter inicializado o servidor, juntamente com uma "/"e o nome da página que foi definida na programação, o endereço IP é apresentado diretamente na tela de execução do terminal do Visual Code Studio. No caso para acessar a página "hello", deve-se digitar no navegador o IP obtido pelo microcontrolador seguido de "/hello". Para acessar as outras páginas basta utilizar o mesmo IP seguido do seu respectivo nome após a "/". A figura 13 apresenta a página *web* recebendo os dados do inversor de frequência durante o funcionamento do motor.

Figura 13 – Página *web* interagindo com o inversor de frequência durante a operação do motor

Fonte: Autoria própria (2022)

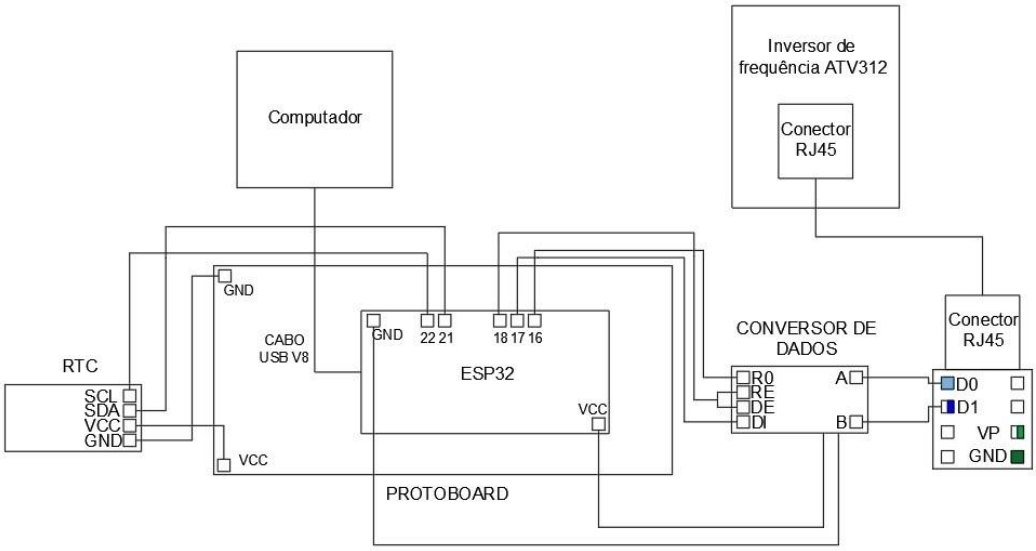
O trabalho feito até essa parte mostra todo o processo para conseguir interagir com o inversor de frequência para fazer comandos de leitura e escrita através do protocolo de comunicação Modbus RTU por intermédio de uma rede Wi-fi. Para ter a possibilidade de fazer um gráfico em relação ao tempo foi utilizido um RTC externo o DS3231 e o RTC interno do microcontrolador. No caso, o periférico externo é usado para ajustar a data e a hora do relógio interno do microcontrolador, após o ajuste, o microcontrolador vai contabilizar o tempo internamente de maneira automática.

4.5.2 Página Web Gráficos

Para tornar possível a criação dos gráficos é necessário vincular os valores de corrente e consumo do motor com o o horário que foi realizado a medida, esse horário é retirado do módulo RTC interno do ESP32 através de variáveis, as varíaveis foram definidas conforme a linha 91 do código e a contagem do tempo foi definida através das linhas 1218 e 1236 do código e podem ser verificadas no Apêndice C.

Para ser possível trabalhar com o periférico externo RTC DS3231 é necessário habilitar o periférico da I2C do ESP32, a I2C é um protocolo de comunicação que possibilita o envio dos bytes do mesmo, esse processo de habilitação se encontra entre as linhas 174 e 188 do código disponível no Apêndice C. Após ter habilitado o periférico é preciso que estajam conectados o pino SCL do DS3231 no pino 22 do microcontrolador que tem a função de clock e o pino SDA no pino 21 do microcontrolador que tem a função de dados. E também é necessário declarar no código do microcontrolador ambos os pinos que vão estar sendo utilizados. A figura 14 mostra a planta final do projeto.

Figura 14 – Planta do projeto com o periférico RTC acoplado



Fonte: Autoria própria (2022)

Para começar a configurar o RTC DS3231 pela primeira vez é preciso escrever os dados de horas, minutos, segundos, dia, mês e ano nos respectivos registrador em valores hexadecimais. Como precisa estar definindo o horário e a data apenas uma única vez, é necessário deixar essa parte comentada no código

na próxima inicialização do programa para que o horário e a data não sejam desconfigurados. A configuração desses registradores estão disponíveis entre as linhas 1389 e 1396 do código disponível no Apêndice C

O controle do horário no microcontrolador vai estar armazenado dentro de variáveis, e para implementar a contagem do horário é preciso iniciar um periférico interno de *timer* que fica operando periodicamente e executa uma função a cada um segundo. Dessa maneira é possível incrementar valores nas variáveis de tempo. A inicialização do periférico *timer*, a criação e inicialização do mesmo pode ser verificada entre as linhas 1381 e 1383 do código, disponível no Apêndice C.

Para não perder os dados lidos pelo inversor de frequência é necessário estar habilitando o armazenamento dos mesmos, para isso é necessário criar uma partição para armazenamento de arquivos na memória *flash* do microcontrolador, através da biblioteca esp_spiffs.h. O arquivo é salvo em formato de texto.

Para implementar esse processo no código foi utilizado um exemplo disponível no GitHub e adaptado as necessidades do projeto, o exemplo se encontra através do seguinte domínio:

- **Link: Armazenamento.** SONGRUO (2022)

Para criar, editar e excluir os arquivos de texto presentes na partição é necessário utilizar funções nativas da linguagem C para a criação de arquivos. Elementos diponíveis na partição estão presentes em um arquivo .CSV de nome "partitions" disponível no exemplo utilizado do GitHub, esse arquivo deve ser adicionado no mesmo diretório em que o programa está sendo elaborado, e após executar o programa o compilador vai traduzir todos os arquivos do diretório para linguagem de máquina e enviar para o microcontrolador.

Das funções utilizadas pode-se citar a "fopen" que é utilizada para criar/adicinar e abrir arquivos de texto. A "fprintf" que é para imprimir o conteudo no arquivo de interesse e o "fclose" que é para fechar o arquivo após ter sido escrito nele.

No código foi implementado a função "fopen" para abir ou ler o arquivo e para isso é necessário informar em qual local da memória ele se encontra. Esse endereçamento no programa está na linha 1164 do código disponível no Apêndice

C. A função "fprint"foi utilizada para escrever os dados no arquivo de texto e a "fclose"foi utilizada para fechar a coleta de dados. Esse processo vai estar se repetindo e armazenando dados de maneira periódica.

Os dados de interesse lidos devem ser separados por ";". No caso, os dados que estão sendo lidos são os dados de corrente, consumo do motor em kVAh, segundos, minutos, horas, ia, mês e ano, pois no nomento em que precisar extrair as informações do arquivo, vai ser possível distinguir cada dado de si, referente aos dados de kVAh eles foram obtidos através da relação direta da corrente e da tensão. Esses mesmos dados vão estar sendo extraídos e armazenados a cada 5 segundos com o intuíto de armazenar as informações para a elaboração dos gráficos em tempo real.

A função principal que é responsável pela coleta e armazenamento de dados se encontra entre as páginas 1161 e 1260 do código disponível no Apêndice C.

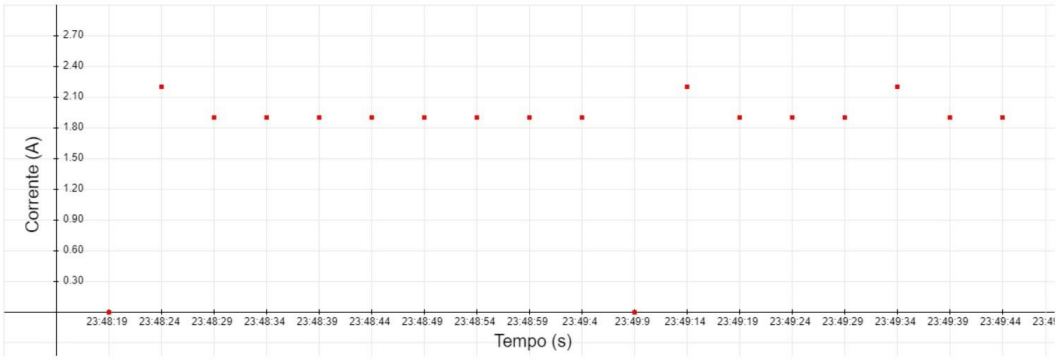
Essas informações vão estar armazenadas dentro da memória *flash* do microcontrolador e estarão sendo mostradas na tela do terminal de visualização do Visual Code Studio.

Com todo esse processo já é possível estar elaborando as duas páginas *webs* que contém os gráficos de corrente e consumo do motor em kVAh. As linhas da programação para a elaboração dos dois gráficos então entre as linhas 476 e 917 do código disponível no Apêndice C.

Para a elaboração do primeiro gráfico de corrente em relação ao tempo e do gráfico de consumo em kVAh do motor, se iniciou o motor com uma partida direta e manteve ele em operação, a próxima ação foi o seu desligamento durante cinco segundos, após isso o comando de rotação de sentido inverso foi enviado e continuou em operação, seguido de um deligamento e outro comando de partida direta do motor, como pode observar nas figuras 15 e 16.

Figura 15 – Gráfico de consumo da corrente do motor em relação ao tempo

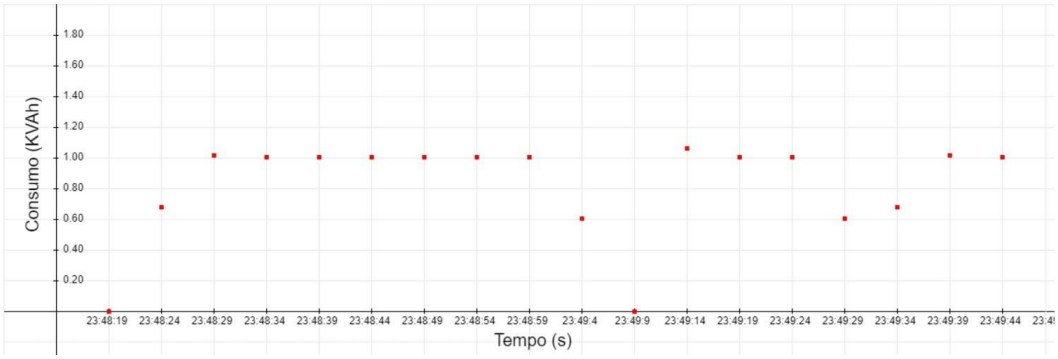
Gráfico:



Fonte: Autoria própria (2022)

Figura 16 – Gráfico de consumo em kVAh do motor

Gráfico:



Fonte: Autoria própria (2022)

É esperado que durante a partida do motor ele atinja picos de correntes maiores do que a nominal, isso foi possível ser verificado nos tempos 23:48:19;23:49:14;23:49:34 da figura 15, entretando, quando houve uma parada seguida de uma ligação, o valor da corrente mostrada no gráfico não é nulo. Isso ocorre devido ao fato que o microcontrolador está extraindo dados a cada cinco segundos do inversor de frequência, e se o valor de corrente de partida acontecer entre esse intervalo o microcontrolador não vai conseguir registar ele, nesse caso, vai registar o valor decorrido após os cinco segundos completos.

A escolha de extração de dados a cada cinco segundos foi feita com o intuito de elaborar um gráfico de maneira rápida para comprovar a eficácia da página *web*, mas o período de tempo pode ser configurado conforme o interesse da pessoa em

questão que deseja analisar o gráfico. O intervalo de tempo configurado está entre as linhas 1237 e 1256 do código disponível no apêndice C.

Os dados no gráfico de consumo do motor em kVAh apresentam valores médios do consumo no decorrer do tempo, e são contabilizados conforme a operação do motor. Entre os tempos 23:49:29 e 23:49:34 do gráfico de consumo, pode-se observar que houve valores menores mesmo tendo um valor de corrente maior em relação as outras medidas naquele mesmo instante, isso deve ao fato que durante esse processo o motor foi desligado e religado rapidamente, ou seja, ficou em inoperação por um período de tempo, logo, esperasse que o seu consumo seja menor.

Referente ao armazenamento de dados, eles podem ser observados na figura 27 no qual a cada cinco segundos vai estar armazenando os dados de operação do inversor de frequênciae sendo mostrados na tela do terminal.

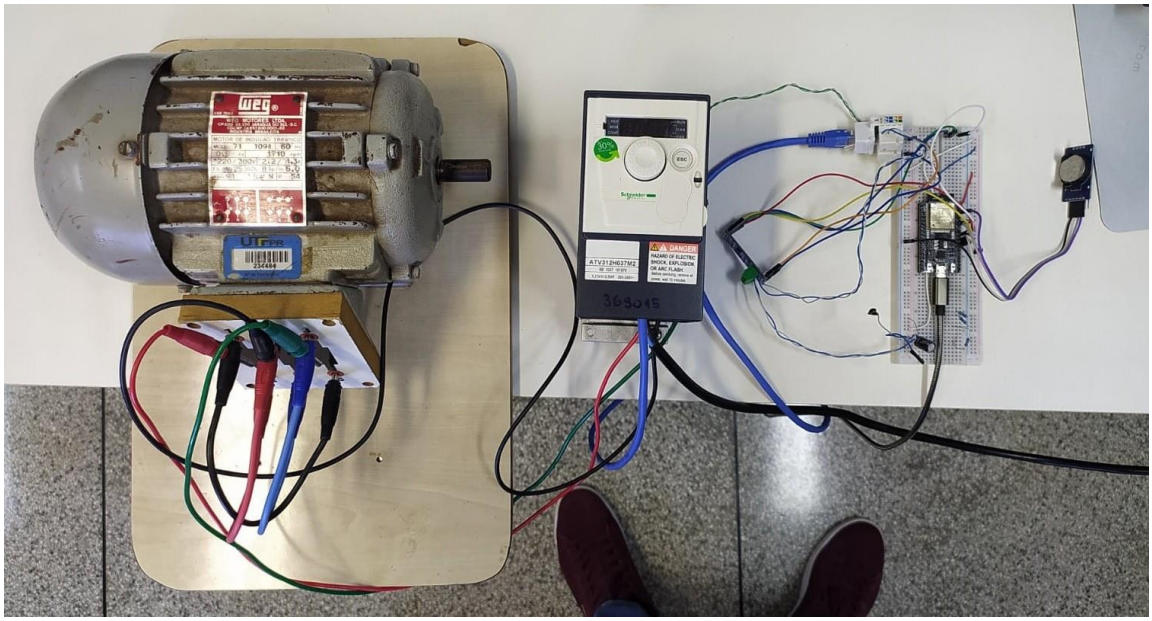
Figura 17 – Dados extraídos em relação a data e hora

```
24/10/22;23:48:19;0.000000;0.000000;
24/10/22;23:48:24;2.200000;0.676622;
24/10/22;23:48:29;1.900000;1.014933;
24/10/22;23:48:34;1.900000;1.004361;
24/10/22;23:48:39;1.900000;1.004361;
24/10/22;23:48:44;1.900000;1.004361;
24/10/22;23:48:49;1.900000;1.004361;
24/10/22;23:48:54;1.900000;1.004361;
24/10/22;23:48:59;1.900000;1.004361;
24/10/22;23:49:4;1.900000;0.602617;
24/10/22;23:49:9;2.000000;0.000000;
24/10/22;23:49:14;2.200000;1.057222;
24/10/22;23:49:19;1.900000;1.004361;
24/10/22;23:49:24;1.900000;1.004361;
24/10/22;23:49:29;1.900000;0.602617;
24/10/22;23:49:34;2.200000;0.676622;
24/10/22;23:49:39;1.900000;1.014933;
24/10/22;23:49:44;1.900000;1.004361;
24/10/22;23:49:49;1.900000;1.004361;
```

Fonte: Autoria própria (2022)

A figura 18 apresenta a planta final do protótipo montado com todos os componentesdurante os testes realizados em laboratório.

Figura 18 – Protótipo



Fonte: Autoria própria (2022)

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho concluiu todos os objetivos propostos e elaborou a sequência de como foi executada a planta do protótipo na prática. É importante ressaltar que os frames enviados para o microcontrolador representam a parte mais importante do projeto, pois uma vez que a mensagem Modbus esteja errada a comunicação não vai ser estabelecida. A mensagem Modbus enviada através da rede RS-485 não apresentou nenhuma falha, o que comprova a eficácia dos componentes utilizados para essa finalidade.

O uso do microcontrolador para a comunicação com o inversor de frequência através da rede Wi-fi da universidade e da rede comum apresentaram ótimos resultados. Durante os testes, os comandos de ações enviados para o inversor de frequência eram executados praticamente de maneira instantânea, mesmo sendo enviados de maneira remota. O que permitia operar o motor no momento em que o click na combo box da página *web* era realizado.

Durante os testes efetuados dentro da universidade, houve problemas do microcontrolador conseguir um endereço IP durante os horários em que estavam acontecendo as aulas, provavelmente devido a quantidade de usuários que estavam conectados através do roteador no qual o microcontrolador estava tentando extrair o endereço IP. Nos finais de semana quando não tinha fluxo de pessoas na universidade, a obtenção do endereço IP não enfrentou nenhum problema e os testes puderam ser realizados sem interrupções.

Devido a isso, foi criado um acesso a rede comum WPA, para conseguir reproduzir o trabalho fora do ambiente universitário. O qual é recomendado para não estar enfrentando problemas com a conexão em dias que a rede esteja sobrecarregada. Apenas para fins de testes. Durante a operação do microcontrolador em uma faixa grande de tempo, ele reinicializava o processo, o que fazia com que a comunicação com o inversor de frequência e com a rede Wi-fi fosse perdida por alguns instântes. A causa provável seria em relação ao estouro da memória de armazenamento, nesse caso, o inversor constatava perda de comunicação e a operação do motor era interrompida, sendo necessário o envio de um novo comando para que ele entrasse em operação novamente.

O sistema de banco de dados apresentaram valores correspondentes aos

mostrados nos gráficos elaborados e ficam disponíveis no histórico da janela do terminal, entretanto, durante um período de tempo, os últimos dados são limpados da memória, só tendo a possibilidade de verificar os dados mais recentes da operação.

As páginas *webs* elaboradas conseguiram com êxito representar o que estava acontecendo durante o precesso de operação, os dados de leitura e operação condiziam com o funcionamento do motor, mas para os gráficos serem atualizados com os novos dados, é necessário estar atualizando a página *web* manualmente, pois o presente trabalho não elaborou uma maneira da página se atualizar automaticamente. Uma problemática encontrada no projeto acontecia quando o microcontrolador reinicializava o processo da comunicação, devido a ficar por um grande período de tempo executando o programa. Com isso, foi constatado que os dados gerados nos gráficos apresentados nas páginas *webs* eram perdidos, iniciando do zero a sua elaboração assim que fosse atualizada a página. Se o microcontrolador não reiniciar o processo, as páginas *webs* podem ser atualizadas a todo instânte que os dados obtidos estaram presentes para serem analisados.

Caso seja de interesse do leitor reproduzir o trabalho, o mesmo está disponibilizando tudo o que é necessário para ser possível a sua reprodução.

Referente as otimizações que o trabalho poderá estar sendo submetido, elaborar uma solução para que a operação do motor não seja interrompida de maneira abrupta quando o microcontrolador apresentar falhas de comunicação.

Elaborar um projeto de supervisão e operação no qual a página web atualize os dados em tempo real sem que seja necessário o próprio operador estar atualizando a página de maneira manual.

Criar a possibilidade de supervisionar e operar mais de um dispositivo através da rede RS-485 por meio do protocolo de comunicação modbus, aprofundando a interação entre mestre e escravos durante a comunicação.

Criar um sistema de armazenamento de dados externo ao microcontrolador para ser possível estar analisando o histórico completo da extração dos dados, pois o microcontrolador tem uma memória interna considerada pequena para grandes quantidades de dados.

REFERÊNCIAS

ANDREI, L. **O Que é GitHub e Como Usá-lo**. 2022. HOSTINGER TUTORIAIS. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-github>.

CARDENAS, R.; JURASK, G. **UM ESTUDO SOBRE O USO DE LINGUAGENS DE PROGRAMAÇÃO E SOFTWARES UTILIZADOS NA INDÚSTRIA DE JOINVILLE E REGIÃO E SUA RELAÇÃO COM O PERFIL DO EGRESSO DO BACHARELADO EM CIÊNCIA E TECNOLOGIA DA UFSC CAMPUS JOINVILLE1.** Santa Catarina: Universidade Federal de Santa Catarina, 2019. 27 p.

CRCCALC. **Consistent Overhead Byte Stuffing (COBS) Encoder/Decoder**. 2022. Crccalc. Disponível em: <https://crccalc.com/>.

CUNHA, J. **PROTÓTIPO DE REDE INSDUTRIAL UTILIZANDO O PADRÃO SERIAL RS485 E PROTOCOLO MODBUS.** Santa Catarina: Universidade Regional de Blumenal, 2000. 104 p.

EDSON. **Introdução ao Visual Studio Code**. 2016. .Net magazine 127. Disponível em: <https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>.

ESPRESSIF. **ESP32 Series Datasheet**. 2020. ESPRESSIF. Disponível em: <https: //www.espressif.com/en/support/documents/technical-documents>.

ESPRESSIF. **ESP-IDF Programming Guide**. 2022. GitHub. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html>.

GADELHA, E. **DESENVOLVIMENTO DE UM PROTÓTIPO PARA AQUISIÇÃO DE DADOS DE PRODUÇÃO INDUSTRIAL UTILIZANDO PROTOCOLO MODBUS RTU EM UMA REDE RS-485 E INTERFACE GRÁFICA INTEGRADA AO GERENCIADOR DE BANCO DE DADOS MYSQL.** Ceará: Universidade Federal do Ceará, 2019. 53 p.

HFUDEV. **docs: changes docs supported targets tables**. 2022. GitHub. Disponível em: <https://github.com/espressif/esp-idf/tree/master/examples/wifi/wifi_enterprise>.

INTEGRATED, M. **DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal**. 2015. MAXIM INTEGRATED. Disponível em: < https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>.

JACK0C. **Merge branch 'feature/esp32c2_optimize_ble_init' into 'master'**. 2022. GitHub. Disponível em: <https://github.com/espressif/esp-idf>.

KAYA, L. **uart: support examples and tests on esp32c6**. 2022. GitHub. Disponível em: <https://github.com/espressif/esp-

idf/tree/master/examples/peripherals/uart/uart_echo_rs485>.

KONZEN, J. **UNIDADE DE CONDICIONAMENTO DE SINAIS COMO ELEMENTO DE INTERFACE ENTRE TRANSFORMADORES DE INSTRUMENTAÇÃO E DISPOSITIVO DE AQUISIÇÃO DE DADOS.** Santa Maria: Universidade Federal de Santa Maria, 2019. 94 p.

LIMA, G. **Saiba tudo sobre o IDE - Integrated Development Environment**. 2022. Alura. Disponível em: <https://www.alura.com.br/artigos/o-que-e-uma-ide>.

MARAFON, D. **INTEGRAÇÃO JAVASERVER FACES E AJAX ESTUDO DA INTEGRAÇÃO ENTRE AS TECNOLOGIAS JSF E AJAX.** Santa Catarina: Universidade Federal de Santa Catarina, 2006. 221 p.

MIRANDA, L. MONITORAMENTO DE PARÂMETROS AMBIENTAIS DE UM LEITO HOSPITALAR UTILIZANDO ESP32. Amazônia: Universidade Do Estado Do Amazonas, 2019.44 p.

MURTA, G. **Conhecendo o ESP32 – usando ESP-IDF (4)**. 2018. BLOG DO GUSTAVO MURTA. Disponível em: <https://jgamblog.wordpress.com/2018/02/09/conhecendo-o-esp32-usando-esp-idf-4/>.

MURTA, J. G. A. **Conhecendo o ESP32 – Introdução (1)**. 2018. Eletrogate. Disponível em:<https://blog.eletrogate.com/conhecendo-o-esp32-introducao-1/>.

PELLIZZONI, L. **DESENVOLVIMENTO DE ARQUITETURA DE SOFTWARE REUTILIZÁVEL PARA IMPLEMENTAÇÃO DE APLICAÇÕES EM .NET**. Rio Grande do Sul: Universidade de Caxias do Sul, 2016. 152 p.

PONTES, W. **IMPLANTAÇÃO DE ESTAÇÃO SOLARIMÉTRICA E DE SISTEMA SUPERVISÓRIO COM SCADABR E PLATAFORMA IoT EM USINA FOTOVOLTAICA NA UNILAB-CE.** Fortaleza: Universidade Federal do Ceará, 2021. 149 p.

RIBEIRO, F. **INTERNET DAS COISAS: DA TEORIA À PRÁTICA.** Ouro Preto: Universidade Federal de Ouro Preto, 2019. 58 p.

RIOS, O. **Protocolos e Serviços de Redes**. Espírito Santo: Instituto Federal Espiríto Santo, 2012. 80 p.

SALERNO, C. *et al.* **SISTEMA AUTOMÁTICO PARA ANÁLISE E DIAGNÓSTICO COMPUTACIONAL EM TEMPO REAL DE INSTALAÇÕES ELÉTRICAS**. Minas Gerais: Universidade Federal de Uberlândia, 2005. 4 p.

SANTOS, B. *et al.* **Indústria 4.0: Desafios e oportunidades**. *In*: DESENVOLVIMENTO, R. P. e (Ed.). Disponível em: <http://revistas.cefet-rj.br/index.php/producaoedesenvolvimento>. Rio De Janeiro: [*s.n.*], 2018. p. 111–124.

SCHNEIDER. **ATV312 Communication variables manual**. 2009. Schneider

Electric. Disponível em: <https://www.se.com/ww/en/download/document/BBV51701/>.

SCHNEIDER. **ATV312 Modbus communication manual**. 2009. Schneider Electric. Disponível em: <https://www.se.com/ww/en/download/document/BBV52816/>.

SCHNEIDER. **Altivar 312 Inversores de frequência para motores assíncronos**. 2016. Schneider Electric. Disponível em: <https://www.se.com/br/pt/faqs/FA279257/>.

SILVA, E. **DESENVOLVIMENTO DO FRAMEWORK JAVA-FÁCIL**. São Paulo: Fundação Educacional do Município de Assis, 2011. 51 p.

SILVA, K. **DESENVOLVIMENTO DE SISTEMA AUTOMATIZADO DE BAIXO CUSTO PARA AQUISIÇÃO DE DADOS DE UMIDADE RELATIVA E DE TEMPERATURA DO AR.** São Paulo: Universidade de São Paulo, 2000. 82 p.

SONGRUO. **ci: Enable esp32c6 example, test_apps, and unit tests CI build stage**. 2022. GitHub. Disponível em: < https://github.com/espressif/esp-idf/tree/master/examples/storage/spiffs>.

SOUZA, F. **SISTEMA EMBARCADO DE AQUISIÇÃO DE DADOS INTEGRADO COM A WEB.** Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2005. 291 p.

SOUZA, L. **POTENCIOSTATO DE BAIXO CUSTO COM TRANSMISSÃO DE DADOS SEM FIO.** São Paulo: Universidade Estadual Paulista (UNESP), 2021. 55 p.

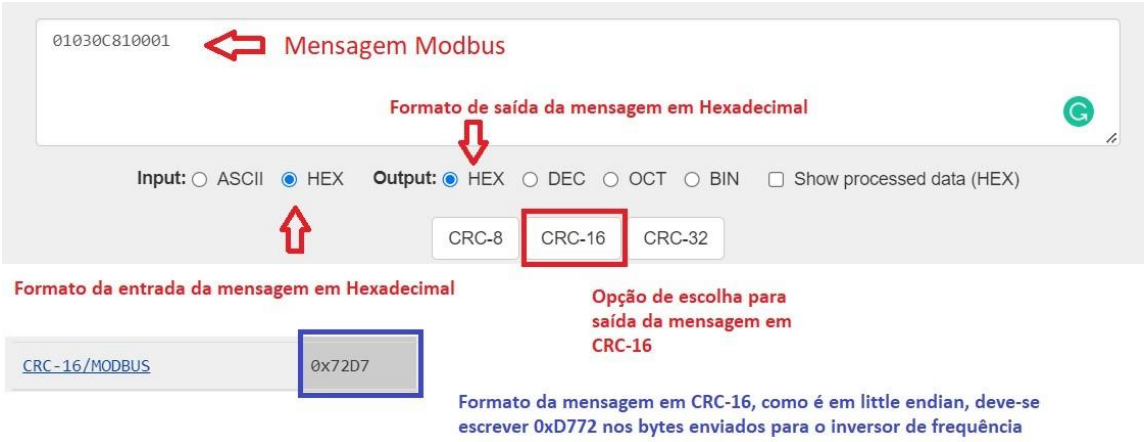
USEFULANGLE. **HTML5 Canvas Tutorial - How to Draw a Graphical Co-ordinate System with Grids and Axes**. 2016. Usefulangle. Disponível em: <https://usefulangle.com/post/19/html5-canvas-tutorial-how-to-draw-graphical-coordinate-system-with-grids-and-axis>.

W3SCHOOLS. **tryhtml_input_test**. 2022. W3schools. Disponível em: < https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_input_test>.

YDESP. **CI: optimize wifi get started example test**. 2022. GitHub. Disponível em:<https://github.com/espressif/espidf/tree/master/examples/wifi/getting_started/station>.

APÊNDICE A – Exemplo Elaborados pelo autor

Figura 19 – Exemplo de uso do conversor online



Fonte: Autoria própria (2022)

Figura 20 – Exemplo Deslocamento de bits

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
Endereço Escravo	Código de solicitação	Número de bytes lidos	Dados	Dados	CRC-16
			15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
			bits	bits	

Parte retirada do código:

```
uint8_t Estado1 = (uint8_t)data[4];
Estado1 = Estado1 << 5;
Estado1 = Estado1 >> 7;
Estado = Estado1;
```

bit de interesse: bit 2 Enable operation

Estado1 = Estado1 <<5; irá deslocar o bit 5 posições

posição original do bit: 0 0 0 0 0 x 0 0

posição nova do bit: x 0 0 0 0 0 0 0

Estado1 = Estado1 >>7; irá deslocar o bit 7posições

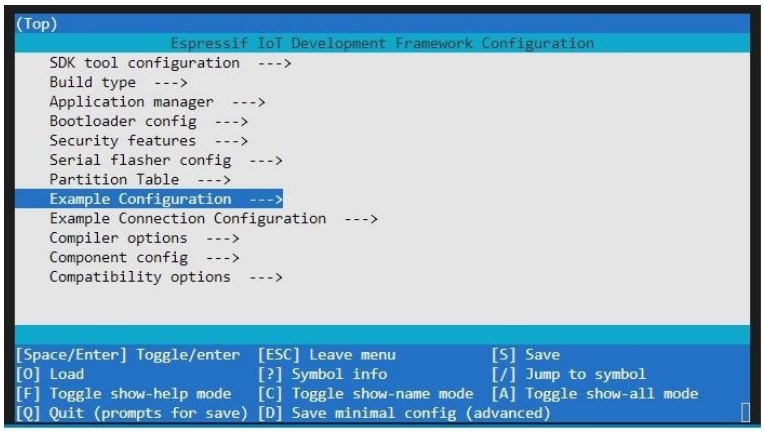
posição final do bit: 0 0 0 0 0 0 0 x

Fonte: Autoria própria (2022)

APÊNDICE B – Configuração para se conectar na rede wi-fi da universidade

O atalho para acessar o Terminal do ESP-IDF é "ctrl+shift+p"e digita na caixa de diálogo: ESP-IDF: Open Esp-IDF Terminal e seleciona a opção correspondente a pesquisa, assim o comando que deve ser utilizado é o "idf.py menuconfig". a caixa de seleção e configuração quese mostrará em seguida na IDE pode ser observado na figura 21.

Figura 21 – Menu de configuração

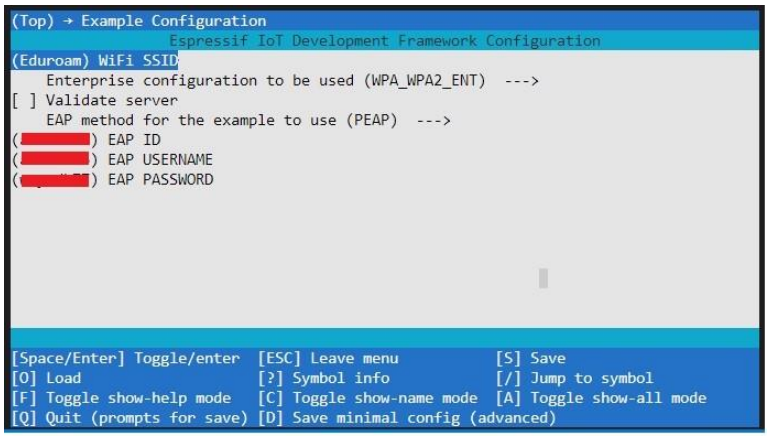


Fonte: Autoria própria (2022)

Ao selecionar a opção Example Configuration é possível configurar a rede wi-fi que deseja se conectar, o usuário e a senha como observa-se na figura 22.

Nas Opções do submenu Example Configuration, na opção Enterprise configuration to be used (WPA_WPA2_ENT) deve selecionar a opção WPA_WPA2_ENT. Na opção EAP method for the exaple to use (PEAP) deve selecionar a opção *Protected Extensible Authentication Protocol* (PEAP).

Figura 22 – Submenu de configuração Example Configuration



Fonte: Autoria própria (2022)

APÊNDICE C – Código

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "esp_wpa2.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_netif.h"
#include <sys/param.h>
#include "esp_eth.h"
#include "protocol_examples_common.h"
#include "esp_tls_crypto.h"
#include <esp_http_server.h>
#include "driver/uart.h"
#include "freertos/queue.h"
#include "sdkconfig.h"
#include "driver/i2c.h"
#include "lwip/err.h"
#include "lwip/sys.h"
#include "driver/adc.h"
#include "esp_adc_cal.h"
#include "esp_spiffs.h"

esp_timer_handle_t periodic_timer;
#define DEFAULT_VREF    1100        //Use adc2_vref_to_gpio() to obtain a better estimate
#define NO_OF_SAMPLES   600         //Multisampling
static const adc_channel_t channel = ADC_CHANNEL_7;     //GPIO34 if ADC1, GPIO14 if ADC2
static const adc_bits_width_t width = ADC_WIDTH_BIT_12;
static const adc_atten_t atten = ADC_ATTEN_DB_0;
static const adc_unit_t unit = ADC_UNIT_1;
static esp_adc_cal_characteristics_t *adc_chars;

float GraficoCorrente[20];
float GraficoConsumo[20];
char GraficoHorarios[20][20];

FILE* arquivo;
static void check_efuse(void)
{
#if CONFIG_IDF_TARGET_ESP32
    //Check if TP is burned into eFuse
    if (esp_adc_cal_check_efuse(ESP_ADC_CAL_VAL_EFUSE_TP) == ESP_OK) {
        printf("eFuse Two Point: Supported\n");
    } else {
        printf("eFuse Two Point: NOT supported\n");
    }
    //Check Vref is burned into eFuse
    if (esp_adc_cal_check_efuse(ESP_ADC_CAL_VAL_EFUSE_VREF) == ESP_OK) {
        printf("eFuse Vref: Supported\n");
    } else {
        printf("eFuse Vref: NOT supported\n");
    }
#elif CONFIG_IDF_TARGET_ESP32S2
    if (esp_adc_cal_check_efuse(ESP_ADC_CAL_VAL_EFUSE_TP) == ESP_OK) {
        printf("eFuse Two Point: Supported\n");
    } else {
        printf("Cannot retrieve eFuse Two Point calibration values. Default calibration values will be
used.\n");
    }
#else
#error "This example is configured for ESP32/ESP32S2."
#endif
}

```

```
70  static void print_char_val_type(esp_adc_cal_value_t val_type)
71  {
72      if (val_type == ESP_ADC_CAL_VAL_EFUSE_TP) {
73          printf("Characterized using Two Point Value\n");
74      } else if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
75          printf("Characterized using eFuse Vref\n");
76      } else {
77          printf("Characterized using Default Vref\n");
78      }
79  }
80
81  #define MODOWIFI 0 //SE = 0 -> COMUM, SE 1 -> WPA2 ENTERPRISE
82
83  uint8_t LigarMotorSentido = 0;
84  uint8_t LigarMotor = 3;
85  float CorrenteMotor = 0;
86  uint8_t SentidoRotacao = 0;
87  float Frequencia = 0;
88  uint8_t Estado = 0;
89  float consumo=0;
90  float MaiorCorrente=0;
91  uint8_t Segundos, Minutos, Horas, Dia, Mes, Ano = 0;
92  uint8_t salvar=0;
93
94  #define EXAMPLE_ESP_WIFI_SSID      "DIGITE USUÁRIO AQUI"
95  #define EXAMPLE_ESP_WIFI_PASS      "DIGITE SENHA AQUI"
96  #define EXAMPLE_ESP_MAXIMUM_RETRY  10
97  #define WIFI_CONNECTED_BIT BIT0
98  #define WIFI_FAIL_BIT      BIT1
99  #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
100 static EventGroupHandle_t s_wifi_event_group;
101
102 #define CONFIG_I2C_MASTER_SCL 22
103 #define CONFIG_I2C_MASTER_SDA 21
104 #define I2C_MASTER_SCL_IO           CONFIG_I2C_MASTER_SCL      /*!< GPIO number used for I2C master
105 clock */
106 #define I2C_MASTER_SDA_IO           CONFIG_I2C_MASTER_SDA      /*!< GPIO number used for I2C master
107 data  */
108 #define I2C_MASTER_NUM              0                          /*!< I2C master i2c port number, the
109 number of i2c peripheral interfaces available will depend on the chip */
110 #define I2C_MASTER_FREQ_HZ          50*1000                    /*!< I2C master clock frequency */
111 #define I2C_MASTER_TX_BUF_DISABLE   0                          /*!< I2C master doesn't need buffer */
112 #define I2C_MASTER_RX_BUF_DISABLE   0                          /*!< I2C master doesn't need buffer */
113 #define I2C_MASTER_TIMEOUT_MS       1000
114
115 #define MPU9250_SENSOR_ADDR                 0x68        /*!< Slave address of the MPU9250 sensor */
116 #define MPU9250_WHO_AM_I_REG_ADDR           0x00        /*!< Register addresses of the "who am I"
117 register */
118
119 #define CONFIG_ECHO_UART_RXD 16
120 #define CONFIG_ECHO_UART_TXD 17
121 #define CONFIG_ECHO_UART_RTS 18
122 #define UART_PIN_NO_CHANGE (-1)
123
124 #define CONFIG_ECHO_UART_PORT_NUM 1
125 #define CONFIG_ECHO_UART_BAUD_RATE 19200
126
127 #define ECHO_TEST_TXD   (CONFIG_ECHO_UART_TXD)
128 #define ECHO_TEST_RXD   (CONFIG_ECHO_UART_RXD)
129
130 // RTS for RS485 Half-Duplex Mode manages DE/~RE
131 #define ECHO_TEST_RTS   (CONFIG_ECHO_UART_RTS)
132
133 // CTS is not used in RS485 Half-Duplex Mode
134 #define ECHO_TEST_CTS   (UART_PIN_NO_CHANGE)
135
136 #define BUF_SIZE        (127)
137 #define BAUD_RATE       (CONFIG_ECHO_UART_BAUD_RATE)
138
```

```
// Read packet timeout
#define PACKET_READ_TICS        (100 / portTICK_PERIOD_MS)
#define ECHO_TASK_STACK_SIZE    (2048)
#define ECHO_TASK_PRIO          (10)
#define ECHO_UART_PORT          (CONFIG_ECHO_UART_PORT_NUM)

#define ECHO_READ_TOUT          (3) // 3.5T * 8 = 28 ticks, TOUT=3 -> ~24..33 ticks

static esp_err_t mpu9250_register_read(uint8_t reg_addr, uint8_t *data, size_t len)
{
    return i2c_master_write_read_device(I2C_MASTER_NUM, MPU9250_SENSOR_ADDR, &reg_addr, 1, data, len,
I2C_MASTER_TIMEOUT_MS / portTICK_RATE_MS);
}

/**
 * @brief Write a byte to a MPU9250 sensor register
 */

static  esp_err_t  mpu9250_register_write_byte(uint8_t  reg_addr,  uint8_t  data)
{
    int ret;
    uint8_t write_buf[2] = {reg_addr, data};
    ret = i2c_master_write_to_device(I2C_MASTER_NUM, MPU9250_SENSOR_ADDR, write_buf, sizeof(write_buf),
I2C_MASTER_TIMEOUT_MS / portTICK_RATE_MS);
    return ret;
}

/**
 * @brief i2c master initialization
 */

static esp_err_t i2c_master_init(void)
{
    int i2c_master_port = I2C_MASTER_NUM;
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };
    i2c_param_config(i2c_master_port, &conf);
    return i2c_driver_install(i2c_master_port, conf.mode, I2C_MASTER_RX_BUF_DISABLE,
I2C_MASTER_TX_BUF_DISABLE, 0);
}

const int uart_num = ECHO_UART_PORT;
uart_config_t uart_config = {
    .baud_rate = BAUD_RATE,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_EVEN,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    .rx_flow_ctrl_thresh = 122,
    .source_clk = UART_SCLK_APB,
};

void GetURI(char *string,char *result)
{
    static unsigned int num = 0, pos = 0;
    for(unsigned int i = 0; i<pos;i++)
    {
        string++;
    }
    while(*string != '\0')
    {
        if(*string == '=')
        {
```

```
            string++;
            pos++;
            while(*string != '&')
            {
                *result = *string;
                result++;
                string++;
                pos++;
                if(*string == '\0')
                {
                    num = 0;
                    pos = 0;
                    *result = '\0';
                    break;
                }
            }
            *result = '\0';
            num++;
            break;
        }
        string++;
        pos++;
    }
}

static void echo_send(const int port, const char* str, uint8_t length)
{
    if (uart_write_bytes(port, str, length) != length) {
        ESP_LOGE("UART", "Send data critical failure.");
        // add your code to handle sending failure here
        abort();
    }
}

void EnableOperation(uint8_t parametro)
{
    char string[8];
    string[0] = 0x01;
    string[1] = 0x06;
    string[2] = 0x21;
    string[3] = 0x35;
    if(parametro==0)
    {
        string[4] = 0x00;
        string[6] = 0xD3;
        string[7] = 0xFC;
    }
    else if (parametro==1)
    {
        string[4] = 0x08;
        string[6] = 0xD4;
        string[7] = 0x3C;
    }
    string[5] = 0x0F;
    echo_send(uart_num, string, 8);
    ESP_LOGI("MODBUS","Enable operation.");
}

void frequency40()
{
    char string[8];
    string[0] = 0x01;
    string[1] = 0x06;
    string[2] = 0x21;
    string[3] = 0x36;
    string[4] = 0x01;
    string[5] = 0x90;
    string[6] = 0x62;
    string[7] = 0x04;
```

```
277         echo_send(uart_num, string, 8);
278         //ESP_LOGI("MODBUS","Frequency.");
279     }
280 
281     void current()
282     {
283         char string[8];
284         string[0] = 0x01;
285         string[1] = 0x03;
286         string[2] = 0x0C;
287         string[3] = 0x84;
288         string[4] = 0x00;
289         string[5] = 0x01;
290         string[6] = 0xC7;
291         string[7] = 0x73;
292         echo_send(uart_num, string, 8);
293         //ESP_LOGI("MODBUS","current.");
294     }
295 
296     void frequency60()
297     {
298         char string[8];
299         string[0] = 0x01;
300         string[1] = 0x06;
301         string[2] = 0x21;
302         string[3] = 0x36;
303         string[4] = 0x02;
304         string[5] = 0x58;
305         string[6] = 0x63;
306         string[7] = 0x62;
307         echo_send(uart_num, string, 8);
308         ESP_LOGI("MODBUS","Frequency.");
309     }
310 
311     void ReadFrequency()
312     {
313         char string[8];
314         string[0] = 0x01;
315         string[1] = 0x03;
316         string[2] = 0x21;
317         string[3] = 0x36;
318         string[4] = 0x00;
319         string[5] = 0x01;
320         string[6] = 0x6E;
321         string[7] = 0x38;
322         echo_send(uart_num, string, 8);
323         //ESP_LOGI("MODBUS","ReadFrequency.");
324     }
325 
326     void EnviarFaultStateReset()
327     {
328         char string[8];
329         string[0] = 0x01;
330         string[1] = 0x06;
331         string[2] = 0x21;
332         string[3] = 0x35;
333         string[4] = 0x00;
334         string[5] = 0x80;
335         string[6] = 0x92;
336         string[7] = 0x58;
337         echo_send(uart_num, string, 8);
338         ESP_LOGI("MODBUS","Fault reset enviado.");
339     }
340 
341     void EnviarShutdown()
342     {
343         char string[8];
344         string[0] = 0x01;
345         string[1] = 0x06;
```

```
346         string[2] = 0x21;
347         string[3] = 0x35;
348         string[4] = 0x00;
349         string[5] = 0x06;
350         string[6] = 0x13;
351         string[7] = 0xFA;
352         echo_send(uart_num, string, 8);
353         ESP_LOGI("MODBUS","Shutdown  enviado.");
354     }
355
356     void EnviarSwitchOn()
357     {
358         char string[8];
359         string[0] = 0x01;
360         string[1] = 0x06;
361         string[2] = 0x21;
362         string[3] = 0x35;
363         string[4] = 0x00;
364         string[5] = 0x07;
365         string[6] = 0xD2;
366         string[7] = 0x3A;
367         echo_send(uart_num, string, 8);
368         ESP_LOGI("MODBUS","Switchon  enviado.");
369     }
370
371     void LerStatus()
372     {
373         char string[8];
374         string[0] = 0x01;
375         string[1] = 0x03;
376         string[2] = 0x0C;
377         string[3] = 0x81;
378         string[4] = 0x00;
379         string[5] = 0x01;
380         string[6] = 0xD7;
381         string[7] = 0x72;
382         echo_send(uart_num, string, 8);
383         //ESP_LOGI("MODBUS","Ler  status  enviado.");
384     }
385
386     void LerDados(uint8_t parametro)
387     {
388         // Allocate buffers for UART
389         //ESP_LOGI("UART", "Esperando dados...");
390         uint8_t* data = (uint8_t*) malloc(BUF_SIZE);
391         int len = 0;
392         while(len==0)
393         {
394             len = uart_read_bytes(uart_num, data, BUF_SIZE, PACKET_READ_TICS);
395             //Write data back to UART
396             if (len > 0) {
397                 //ESP_LOGI("UART", "Received %u bytes:", len);
398                 for (int i = 0; i < len; i++) {
399                     //ESP_LOGI("UART","0x%.2X ", (uint8_t)data[i]);
400                 }
401             }
402         }
403         if(parametro == 0) // CORRENTE
404         {
405             //static float maiorcorrente = 0; //por ser estática é uma variável que guarda o valor
406     anterior, ela só é 0 no momento da criação
407             float correntereal;
408             uint16_t corrente = (uint8_t)data[3];
409             corrente = corrente<<8;
410             corrente = corrente + data[4];
411             correntereal=corrente;
412             CorrenteMotor = correntereal/10;
413             if(CorrenteMotor>MaiorCorrente)
414             {
```

```
            MaiorCorrente=CorrenteMotor;
        }
        printf("%f\n maiorcorrente",MaiorCorrente);
    }
    else if(parametro == 1) // Status
    {
        uint8_t Estado1 = (uint8_t)data[4];
        Estado1 = Estado1 << 5;
        Estado1 = Estado1 >> 7;
        Estado = Estado1;
        uint8_t SentidoRotacao1 = (uint8_t)data[3];
        SentidoRotacao = SentidoRotacao1 >> 7;
    }
    else if(parametro == 2) // frequência
    {
        uint16_t frequencia1 = (uint16_t)data[3];
        frequencia1 = frequencia1<<8;
        frequencia1 = frequencia1 + data[4];
        Frequencia = frequencia1/10;

    }
}

uint8_t ConectadoWifi = 0;

#define EXAMPLE_WIFI_SSID CONFIG_EXAMPLE_WIFI_SSID
#define EXAMPLE_EAP_METHOD CONFIG_EXAMPLE_EAP_METHOD
#define EXAMPLE_EAP_ID CONFIG_EXAMPLE_EAP_ID
#define EXAMPLE_EAP_USERNAME CONFIG_EXAMPLE_EAP_USERNAME
#define EXAMPLE_EAP_PASSWORD CONFIG_EXAMPLE_EAP_PASSWORD

static EventGroupHandle_t wifi_event_group;

static esp_netif_t *sta_netif = NULL;

const int CONNECTED_BIT = BIT0;

static const char *TAG = "example";

#ifdef CONFIG_EXAMPLE_VALIDATE_SERVER_CERT
extern uint8_t ca_pem_start[] asm("_binary_ca_pem_start");
extern uint8_t ca_pem_end[]   asm("_binary_ca_pem_end");
#endif /* CONFIG_EXAMPLE_VALIDATE_SERVER_CERT */

#ifdef CONFIG_EXAMPLE_EAP_METHOD_TLS
extern uint8_t client_crt_start[] asm("_binary_client_crt_start");
extern uint8_t client_crt_end[]   asm("_binary_client_crt_end");
extern uint8_t client_key_start[] asm("_binary_client_key_start");
extern uint8_t client_key_end[]   asm("_binary_client_key_end");
#endif /* CONFIG_EXAMPLE_EAP_METHOD_TLS */

#if defined CONFIG_EXAMPLE_EAP_METHOD_TTLS
esp_eap_ttls_phase2_types TTLS_PHASE2_METHOD = CONFIG_EXAMPLE_EAP_METHOD_TTLS_PHASE_2;
#endif /* CONFIG_EXAMPLE_EAP_METHOD_TTLS */

static httpd_handle_t server = NULL;

static esp_err_t dados_get_handler(httpd_req_t *req)
{
    httpd_resp_sendstr_chunk(req, "<!DOCTYPE html>");
    httpd_resp_sendstr_chunk(req, "<html>");
    httpd_resp_sendstr_chunk(req, "<head>");
    httpd_resp_sendstr_chunk(req, "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-
8\" />");
    httpd_resp_sendstr_chunk(req, "<style type=\"text/css\">");
    httpd_resp_sendstr_chunk(req, "body {");
    httpd_resp_sendstr_chunk(req, "      margin: 0;");
    httpd_resp_sendstr_chunk(req, "}");
    httpd_resp_sendstr_chunk(req, "canvas {");
```

```
484         httpd_resp_sendstr_chunk(req, "    display: block;");
485         httpd_resp_sendstr_chunk(req, "    margin: 40px auto;");
486         httpd_resp_sendstr_chunk(req, "}");
487         httpd_resp_sendstr_chunk(req, "</style>");
488         httpd_resp_sendstr_chunk(req, "</head>");
489         httpd_resp_sendstr_chunk(req, "<body>");
490         httpd_resp_sendstr_chunk(req, "<h1 style=\"text-align: center;\">Gr&aacute;fico:</h1>");
491         httpd_resp_sendstr_chunk(req, "<canvas id=\"my-canvas\" width=\"1200\" height=\"400\"></canvas>");
492         httpd_resp_sendstr_chunk(req, "<script type=\"text/javascript\" charset=\"utf-8\">");
493         httpd_resp_sendstr_chunk(req, "function drawPoint(context, x, y, label, color, size) {");
494         httpd_resp_sendstr_chunk(req, "      if (color == null) {");
495         httpd_resp_sendstr_chunk(req, "         color = '#000';");
496         httpd_resp_sendstr_chunk(req, "    }");
497         httpd_resp_sendstr_chunk(req, "    if (size == null) {");
498         httpd_resp_sendstr_chunk(req, "         size = 5;");
499         httpd_resp_sendstr_chunk(req, "    }");
500         httpd_resp_sendstr_chunk(req, "    var radius = 0.5 * size;");
501         httpd_resp_sendstr_chunk(req, "      var pointX = Math.round(x - radius);");
502         httpd_resp_sendstr_chunk(req, "    var pointY = Math.round(y  - radius);");
503         httpd_resp_sendstr_chunk(req, "    context.beginPath();");
504         httpd_resp_sendstr_chunk(req, "    context.fillStyle = color;");
505         httpd_resp_sendstr_chunk(req, "    context.fillRect(pointX, pointY, size, size);");
506         httpd_resp_sendstr_chunk(req, "    context.fill();      ");
507         httpd_resp_sendstr_chunk(req, "    if (label) {");
508         httpd_resp_sendstr_chunk(req, "         var textX = Math.round(x);");
509         httpd_resp_sendstr_chunk(req, "         var textY = Math.round(pointY - 5);");
510         httpd_resp_sendstr_chunk(req, "         context.font = 'Italic 14px Arial';");
511         httpd_resp_sendstr_chunk(req, "         context.fillStyle = color;");
512         httpd_resp_sendstr_chunk(req, "         context.textAlign = 'center';");
513         httpd_resp_sendstr_chunk(req, "         context.fillText(label, textX, textY);");
514         httpd_resp_sendstr_chunk(req, "      }");
515         httpd_resp_sendstr_chunk(req, "}");
516         httpd_resp_sendstr_chunk(req, "var grid_size = 35;");
517         httpd_resp_sendstr_chunk(req, "var x_axis_distance_grid_lines = 10;");
518         httpd_resp_sendstr_chunk(req, "var y_axis_distance_grid_lines = 1;");
519         httpd_resp_sendstr_chunk(req, "var x_axis_starting_point = { number: 0.3, suffix: '' };");
520         httpd_resp_sendstr_chunk(req, "var y_axis_starting_point = { number: 0.3, suffix: '' };");
521         httpd_resp_sendstr_chunk(req, "var canvas = document.getElementById(\"my-canvas\");");
522         httpd_resp_sendstr_chunk(req, "var ctx = canvas.getContext(\"2d\");");
523         httpd_resp_sendstr_chunk(req, "var canvas_width = canvas.width;");
524         httpd_resp_sendstr_chunk(req, "var canvas_height = canvas.height;");
525         httpd_resp_sendstr_chunk(req, "var num_lines_x = Math.floor(canvas_height/grid_size);");
526         httpd_resp_sendstr_chunk(req, "var num_lines_y = Math.floor(canvas_width/grid_size);");
527         httpd_resp_sendstr_chunk(req, "const dadoscorrente = [");
528         for(unsigned int i = 0; i<20; i++)
529         {
530             httpd_resp_sendstr_chunk(req, GraficoHorarios[i]);
531             if(i<19)
532             {
533                 httpd_resp_sendstr_chunk(req, ",");
534             }
535             if(i==19)
536             {
537                 httpd_resp_sendstr_chunk(req, "];");
538             }
539         }
540
541         httpd_resp_sendstr_chunk(req, "for(var i=0; i<=num_lines_x; i++) {");
542         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
543         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1; ");
544         httpd_resp_sendstr_chunk(req, "    if(i == x_axis_distance_grid_lines) ");
545         httpd_resp_sendstr_chunk(req, "        ctx.strokeStyle = \"#000000\";");
546         httpd_resp_sendstr_chunk(req, "    else");
547         httpd_resp_sendstr_chunk(req, "        ctx.strokeStyle = \"#e9e9e9\";");
548         httpd_resp_sendstr_chunk(req, "    if(i == num_lines_x) {");
549         httpd_resp_sendstr_chunk(req, "        ctx.moveTo(0, grid_size*i);");
550         httpd_resp_sendstr_chunk(req, "        ctx.lineTo(canvas_width, grid_size*i);");
551         httpd_resp_sendstr_chunk(req, "    }");
552         httpd_resp_sendstr_chunk(req, "    else {");
```

```
553         httpd_resp_sendstr_chunk(req, "        ctx.moveTo(0, grid_size*i+0.5);");
554         httpd_resp_sendstr_chunk(req, "        ctx.lineTo(canvas_width, grid_size*i+0.5);");
555         httpd_resp_sendstr_chunk(req, "    }");
556         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
557         httpd_resp_sendstr_chunk(req, "}");
558         httpd_resp_sendstr_chunk(req, "for(i=0; i<=num_lines_y; i++) {");
559         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
560         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
561         httpd_resp_sendstr_chunk(req, "    if(i == y_axis_distance_grid_lines) ");
562         httpd_resp_sendstr_chunk(req, "        ctx.strokeStyle = \"#000000\";");
563         httpd_resp_sendstr_chunk(req, "    else");
564         httpd_resp_sendstr_chunk(req, "        ctx.strokeStyle = \"#e9e9e9\";");
565         httpd_resp_sendstr_chunk(req, "    if(i == num_lines_y) {");
566         httpd_resp_sendstr_chunk(req, "        ctx.moveTo(grid_size*i, 0);");
567         httpd_resp_sendstr_chunk(req, "        ctx.lineTo(grid_size*i,   canvas_height);");
568         httpd_resp_sendstr_chunk(req, "    }");
569         httpd_resp_sendstr_chunk(req, "    else {");
570         httpd_resp_sendstr_chunk(req, "        ctx.moveTo(60*i+0.5, 0);");
571         httpd_resp_sendstr_chunk(req, "        ctx.lineTo(60*i+0.5,   canvas_height);");
572         httpd_resp_sendstr_chunk(req, "    }");
573         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
574         httpd_resp_sendstr_chunk(req, "}");
575         httpd_resp_sendstr_chunk(req,     "ctx.translate(y_axis_distance_grid_lines*60,
576     x_axis_distance_grid_lines*grid_size);");
577         httpd_resp_sendstr_chunk(req, "for(i=1; i<(num_lines_y - y_axis_distance_grid_lines); i++) {");
578         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
579         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
580         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
581         httpd_resp_sendstr_chunk(req, "    ctx.moveTo(60*i+0.5, -3);");
582         httpd_resp_sendstr_chunk(req, "    ctx.lineTo(60*i+0.5, 3);");
583         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
584         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
585         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'center';");
586         httpd_resp_sendstr_chunk(req, "    ctx.fillText(dadoscorrente[i], 60*i -2,  15);");
587         httpd_resp_sendstr_chunk(req, "}");
588         httpd_resp_sendstr_chunk(req, "for(i=1; i<y_axis_distance_grid_lines; i++) {");
589         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
590         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
591         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
592         httpd_resp_sendstr_chunk(req, "    ctx.moveTo(-grid_size*i+0.5, -3);");
593         httpd_resp_sendstr_chunk(req, "    ctx.lineTo( -grid_size*i+0.5,  3);");
594         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
595         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
596         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'end';");
597         httpd_resp_sendstr_chunk(req, "    ctx.fillText(-x_axis_starting_point.number*i    +
598     x_axis_starting_point.suffix, -grid_size*i+3, 15);");
599         httpd_resp_sendstr_chunk(req, "}");
600         httpd_resp_sendstr_chunk(req, "for(i=1; i<(num_lines_x - x_axis_distance_grid_lines); i++) {");
601         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
602         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
603         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
604         httpd_resp_sendstr_chunk(req, "    ctx.moveTo( -3, grid_size*i+0.5);");
605         httpd_resp_sendstr_chunk(req, "    ctx.line To(3, grid_size*i+0.5);");
606         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
607         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
608         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'start';");
609         httpd_resp_sendstr_chunk(req, "    ctx.fillText(-y_axis_starting_point.number*i  +
610     y_axis_starting_point.suffix, 8, grid_size*i+3);");
611         httpd_resp_sendstr_chunk(req, "}");
612         httpd_resp_sendstr_chunk(req, "for(i=1; i<x_axis_distance_grid_lines; i++) {");
613         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
614         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
615         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
616         httpd_resp_sendstr_chunk(req, "    ctx.moveTo( -3, -grid_size*i+0.5);");
617         httpd_resp_sendstr_chunk(req, "    ctx.lineTo(3, -grid_size*i+0.5);");
618         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
619         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
620         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'start';");
```

```
    httpd_resp_sendstr_chunk(req, "    ctx.fillText((y_axis_starting_point.number*i).toFixed(2) +
y_axis_starting_point.suffix, 8, -grid_size*i+3);");
    httpd_resp_sendstr_chunk(req, "}");
    for(unsigned int i = 0; i < 18 ; i++)
    {
        float x = 60*(i+1), y = GraficoCorrente[i]*(-35/0.3);
        char temp[20] = "";
        sprintf(temp,"%f",x);
        httpd_resp_sendstr_chunk(req, "drawPoint(ctx, ");
        httpd_resp_sendstr_chunk(req, temp);
        httpd_resp_sendstr_chunk(req, ",");
        sprintf(temp,"%f",y);
        httpd_resp_sendstr_chunk(req, temp);
        httpd_resp_sendstr_chunk(req, ", \"\", \"#FF0000\", 5);");
    }
    httpd_resp_sendstr_chunk(req, "var ctx = canvas.getContext(\"2d\");");
    httpd_resp_sendstr_chunk(req, "ctx.fillStyle = \"black\";");
    httpd_resp_sendstr_chunk(req, "ctx.rotate(-1.57075);");
    httpd_resp_sendstr_chunk(req, "ctx.font = \"20px Arial\";");
    httpd_resp_sendstr_chunk(req, "ctx.fillText(\"Corrente (A)\", 90, -20);");
    httpd_resp_sendstr_chunk(req, "ctx.rotate(1.57075);");
    httpd_resp_sendstr_chunk(req, "ctx.font = \"20px Arial\";");
    httpd_resp_sendstr_chunk(req, "ctx.fillText(\"Tempo (s)\", 500, 40);");
    httpd_resp_sendstr_chunk(req, "</script>");
    httpd_resp_sendstr_chunk(req, "</body>");

    httpd_resp_sendstr_chunk(req, NULL);
    return ESP_OK;
}

static esp_err_t consumo_get_handler(httpd_req_t *req)
{
    httpd_resp_sendstr_chunk(req, "<!DOCTYPE html>");
    httpd_resp_sendstr_chunk(req, "<html>");
    httpd_resp_sendstr_chunk(req, "<head>");
    httpd_resp_sendstr_chunk(req, "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-
8\" />");
    httpd_resp_sendstr_chunk(req, "<style type=\"text/css\">");
    httpd_resp_sendstr_chunk(req, "body {");
    httpd_resp_sendstr_chunk(req, "      margin: 0;");
    httpd_resp_sendstr_chunk(req, "}");
    httpd_resp_sendstr_chunk(req, "canvas {");
    httpd_resp_sendstr_chunk(req, "    display: block;");
    httpd_resp_sendstr_chunk(req, "    margin: 40px auto;");
    httpd_resp_sendstr_chunk(req, "}");
    httpd_resp_sendstr_chunk(req, "</style>");
    httpd_resp_sendstr_chunk(req, "</head>");
    httpd_resp_sendstr_chunk(req, "<body>");
    httpd_resp_sendstr_chunk(req, "<h1 style=\"text-align: center;\">Gr&aacute;fico:</h1>");
    httpd_resp_sendstr_chunk(req, "<canvas id=\"my-canvas\" width=\"1200\" height=\"400\"></canvas>");
    httpd_resp_sendstr_chunk(req, "<script type=\"text/javascript\" charset=\"utf-8\">");
    httpd_resp_sendstr_chunk(req, "function drawPoint(context, x, y, label, color, size) {");
    httpd_resp_sendstr_chunk(req, "      if (color == null) {");
    httpd_resp_sendstr_chunk(req, "         color = '#000';");
    httpd_resp_sendstr_chunk(req, "    }");
    httpd_resp_sendstr_chunk(req, "    if (size == null) {");
    httpd_resp_sendstr_chunk(req, "        size = 5;");
    httpd_resp_sendstr_chunk(req, "    }");
    httpd_resp_sendstr_chunk(req, "    var radius = 0.5 * size;");
    httpd_resp_sendstr_chunk(req, "      var pointX = Math.round(x - radius);");
    httpd_resp_sendstr_chunk(req, "    var pointY = Math.round(y - radius);");
    httpd_resp_sendstr_chunk(req, "    context.beginPath();");
    httpd_resp_sendstr_chunk(req, "    context.fillStyle = color;");
    httpd_resp_sendstr_chunk(req, "    context.fillRect(pointX, pointY, size, size);");
    httpd_resp_sendstr_chunk(req, "    context.fill();     ");
    httpd_resp_sendstr_chunk(req, "    if (label) {");
    httpd_resp_sendstr_chunk(req, "        var textX = Math.round(x);");
    httpd_resp_sendstr_chunk(req, "        var textY = Math.round(pointY - 5);");
    httpd_resp_sendstr_chunk(req, "        context.font = 'Italic 14px Arial';");
```

```
690       httpd_resp_sendstr_chunk(req, "        context.fillStyle = color;");
691       httpd_resp_sendstr_chunk(req, "        context.textAlign = 'center';");
692       httpd_resp_sendstr_chunk(req, "        context.fillText(label, textX, textY);");
693       httpd_resp_sendstr_chunk(req, "      }");
694       httpd_resp_sendstr_chunk(req, "}");
695       httpd_resp_sendstr_chunk(req, "var grid_size = 35;");
696       httpd_resp_sendstr_chunk(req, "var x_axis_distance_grid_lines = 10;");
697       httpd_resp_sendstr_chunk(req, "var y_axis_distance_grid_lines = 1;");
698       httpd_resp_sendstr_chunk(req, "var x_axis_starting_point = { number: 0.2, suffix: '' };");
699       httpd_resp_sendstr_chunk(req, "var y_axis_starting_point = { number: 0.2, suffix: '' };");
700       httpd_resp_sendstr_chunk(req, "var  canvas =  document.getElementById(\"my-canvas\");");
701       httpd_resp_sendstr_chunk(req, "var ctx = canvas.getContext(\"2d\");");
702       httpd_resp_sendstr_chunk(req, "var canvas_width = canvas.width;");
703       httpd_resp_sendstr_chunk(req, "var canvas_height = canvas.height;");
704       httpd_resp_sendstr_chunk(req, "var num_lines_x = Math.floor(canvas_height/grid_size);");
705       httpd_resp_sendstr_chunk(req,  "var  num_lines_y  =  Math.floo r(canvas_width/grid_size);");
706       httpd_resp_sendstr_chunk(req,  "const dadoscorrente = [");
707       for(unsigned int i = 0; i<20; i++)
708       {
709           httpd_resp_sendstr_chunk(req,  GraficoHorarios[i]);
710           if(i<19)
711           {
712               httpd_resp_sendstr_chunk(req,  ",");
713           }
714           if(i==19)
715           {
716               httpd_resp_sendstr_chunk(req,  "];");
717           }
718       }
719
720       httpd_resp_sendstr_chunk(req, "for(var i=0; i<=num_lines_x; i++) {");
721       httpd_resp_sendstr_chunk(req, "     ctx.beginPath();");
722       httpd_resp_sendstr_chunk(req, "     ctx.lineWidth = 1; ");
723       httpd_resp_sendstr_chunk(req, "     if(i == x_axis_distance_grid_lines) ");
724       httpd_resp_sendstr_chunk(req, "         ctx.strokeStyle = \"#000000\";");
725       httpd_resp_sendstr_chunk(req, "     else");
726       httpd_resp_sendstr_chunk(req, "         ctx.stroke Style  =  \"#e9e9e9\";");
727       httpd_resp_sendstr_chunk(req, "     if(i == num_lines_x) {");
728       httpd_resp_sendstr_chunk(req, "         ctx.moveTo(0, grid_size*i);");
729       httpd_resp_sendstr_chunk(req, "         ctx.lineTo(canvas_width,  grid_size*i);");
730       httpd_resp_sendstr_chunk(req, "     }");
731       httpd_resp_sendstr_chunk(req, "     else {");
732       httpd_resp_sendstr_chunk(req, "         ctx.moveTo(0, grid_size*i+0.5);");
733       httpd_resp_sendstr_chunk(req, "         ctx.lineTo(canvas_width,  grid_size*i+0.5);");
734       httpd_resp_sendstr_chunk(req, "     }");
735       httpd_resp_sendstr_chunk(req, "     ctx.stroke();");
736       httpd_resp_sendstr_chunk(req, "}");
737       httpd_resp_sendstr_chunk(req, "for(i=0; i<=num_lines_y; i++) {");
738       httpd_resp_sendstr_chunk(req, "     ctx.beginPa th();");
739       httpd_resp_sendstr_chunk(req, "     ctx.lineWidth = 1;");
740       httpd_resp_sendstr_chunk(req, "     if(i == y_axis_distance_grid_lines) ");
741       httpd_resp_sendstr_chunk(req, "         ctx.strokeStyle = \"#000000\";");
742       httpd_resp_sendstr_chunk(req, "     else");
743       httpd_resp_sendstr_chunk(req, "         ctx.strokeStyle = \"#e9e9e9\";");
744       httpd_resp_sendstr_chunk(req, "     if(i == num_lines_y) {");
745       httpd_resp_sendstr_chunk(req, "         ctx.moveTo(grid_size*i,  0);");
746       httpd_resp_sendstr_chunk(req, "         ctx.lineTo(grid_size*i,    canvas_height);");
747       httpd_resp_sendstr_chunk(req, "     }");
748       httpd_resp_sendstr_chunk(req, "     else {");
749       httpd_resp_sendstr_chunk(req, "         ctx.moveTo(60*i+0.5,  0);");
750       httpd_resp_sendstr_chunk(req, "         ctx.lineTo(60*i+0.5,    canvas_height);");
751       httpd_resp_sendstr_chunk(req, "     }");
752       httpd_resp_sendstr_chunk(req, "     ctx.stroke();");
753       httpd_resp_sendstr_chunk(req, "}");
754       httpd_resp_sendstr_chunk(req,     "ctx.translate(y_axis_distance_grid_lines*60,
755   x_axis_distance_grid_lines*grid_size);");
756       httpd_resp_sendstr_chunk(req, "for(i=1; i<(num_lines_y - y_axis_distance_grid_lines); i++) {");
757       httpd_resp_sendstr_chunk(req, "     ctx.beginPath();");
758       httpd_resp_sendstr_chunk(req, "     ctx.lineWidth = 1;");
```

```
759         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
760         httpd_resp_sendstr_chunk(req, "    ctx.moveTo(60*i+0.5, -3);");
761         httpd_resp_sendstr_chunk(req, "    ctx.lineTo(60*i+0.5, 3);");
762         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
763         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
764         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'center';");
765         httpd_resp_sendstr_chunk(req, "    ctx.fillText(dadoscorrente[i], 60*i -2, 15);");
766         httpd_resp_sendstr_chunk(req, "}");
767         httpd_resp_sendstr_chunk(req, "for(i=1; i<y_axis_distance_grid_lines; i++) {");
768         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
769         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
770         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
771         httpd_resp_sendstr_chunk(req, "    ctx.moveTo(-grid_size*i+0.5, -3);");
772         httpd_resp_sendstr_chunk(req, "    ctx.lineTo( -grid_size*i+0.5, 3);");
773         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
774         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
775         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'end';");
776         httpd_resp_sendstr_chunk(req, "    ctx.fillText(-x_axis_starting_point.number*i    +
777 x_axis_starting_point.suffix, -grid_size*i+3, 15);");
778         httpd_resp_sendstr_chunk(req, "}");
779         httpd_resp_sendstr_chunk(req, "for(i=1; i<(num_lines_x - x_axis_distance_grid_lines); i++) {");
780         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
781         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
782         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
783         httpd_resp_sendstr_chunk(req, "    ctx.moveTo( -3, grid_size*i+0.5);");
784         httpd_resp_sendstr_chunk(req, "    ctx.lineTo(3, grid_size*i+0.5);");
785         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
786         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
787         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'sta rt';");
788         httpd_resp_sendstr_chunk(req, "    ctx.fillText(-y_axis_starting_point.number*i    +
789 y_axis_starting_point.suffix, 8, grid_size*i+3);");
790         httpd_resp_sendstr_chunk(req, "}");
791         httpd_resp_sendstr_chunk(req, "for(i=1; i<x_axis_distance_grid_lines; i++) {");
792         httpd_resp_sendstr_chunk(req, "    ctx.beginPath();");
793         httpd_resp_sendstr_chunk(req, "    ctx.lineWidth = 1;");
794         httpd_resp_sendstr_chunk(req, "    ctx.strokeStyle = \"#000000\";");
795         httpd_resp_sendstr_chunk(req, "    ctx.moveTo(-3, -grid_size*i+0.5);");
796         httpd_resp_sendstr_chunk(req, "    ctx.lineTo(3, -grid_size*i+0.5);");
797         httpd_resp_sendstr_chunk(req, "    ctx.stroke();");
798         httpd_resp_sendstr_chunk(req, "    ctx.font = '12px Arial';");
799         httpd_resp_sendstr_chunk(req, "    ctx.textAlign = 'start';");
800         httpd_resp_sendstr_chunk(req, "    ctx.fillText((y_axis_starting_point.number*i).toFixed(2) +
801 y_axis_starting_point.suffix, 8, -grid_size*i+3);");
802         httpd_resp_sendstr_chunk(req, "}");
803         for(unsigned int i = 0; i < 18 ; i++)
804         {
805             float x = 60*(i+1), y = GraficoConsumo[i]*(-35/0.2);
806             char temp[20] = "";
807             sprintf(temp,"%f",x);
808             httpd_resp_sendstr_chunk(req, "drawPoint(ctx, ");
809             httpd_resp_sendstr_chunk(req, temp);
810             httpd_resp_sendstr_chunk(req, ",");
811             sprintf(temp,"%f",y);
812             httpd_resp_sendstr_chunk(req, temp);
813             httpd_resp_sendstr_chunk(req, ", \"\", \"#FF0000\", 5);");
814         }
815         httpd_resp_sendstr_chunk(req, "var ctx = canvas.getContext(\"2d\");");
816         httpd_resp_sendstr_chunk(req, "ctx.fillStyle = \"black\";");
817         httpd_resp_sendstr_chunk(req, "ctx.rotate(-1.57075);");
818         httpd_resp_sendstr_chunk(req, "ctx.font = \"20px Arial\";");
819         httpd_resp_sendstr_chunk(req, "ctx.fillText(\"Consumo (KVAh)\", 90, -20);");
820         httpd_resp_sendstr_chunk(req, "ctx.rotate(1.57075);");
821         httpd_resp_sendstr_chunk(req, "ctx.font = \"20px Arial\";");
822         httpd_resp_sendstr_chunk(req, "ctx.fillText(\"Tempo (s)\", 500, 40);");
823         httpd_resp_sendstr_chunk(req, "</script>");
824         httpd_resp_sendstr_chunk(req, "</body>");
825 
826         httpd_resp_sendstr_chunk(req, NULL);
827         return ESP_OK;
```

```
}

/* An HTTP GET handler */
static esp_err_t hello_get_handler(httpd_req_t *req)
{
     char string[10] = "";
    //sprintf(string,"%f",CorrenteMotor);
    char string1[513];
    char resultado[50];
    sprintf(string1,"%s",req->uri);
    GetURI(string1,resultado);
    printf(resultado);
    if(Estado == 0)
    {
        if(strcmp(resultado,"LigarDireto")==0) //tem que fazer como string
        {
            LigarMotor = 1;
            LigarMotorSentido=0;
        }
        else if(strcmp(resultado,"LigarReverso")==0) //tem que fazer como string
        {
            LigarMotor = 1;
            LigarMotorSentido=1;
        }
    }
    else if(Estado == 1)
    {
        if(strcmp(resultado,"Desligar")==0)
        {
            LigarMotor = 0;
        }
    }

    //httpd_resp_sendstr_chunk(req, "Corrente do motor:<br>");
    httpd_resp_sendstr_chunk(req, "<h1 style=\"text-align: left;\">Inversor</h1>");
    httpd_resp_sendstr_chunk(req,  "<p><span  style= \"color:
#ff0000;\"><strong>A&ccedil;&atilde;o</strong></span></p>");
    httpd_resp_sendstr_chunk(req, "<p>Escolha uma Op&ccedil;&atilde;o</p>");
    httpd_resp_sendstr_chunk(req, "<form action=\"/hello\">");
    httpd_resp_sendstr_chunk(req, "<label  for=\"Selecao\"></label>");
    httpd_resp_sendstr_chunk(req, "<select name=\"Selecao\" id=\"Selecao\">");
    httpd_resp_sendstr_chunk(req, "<option value=\"LigarDireto\">Ligar Direto</option>");
    httpd_resp_sendstr_chunk(req, "<option value=\"LigarReverso\">Ligar Reverso</option>");
    httpd_resp_sendstr_chunk(req, "<option value=\"Desligar\">Desligar</option>");
    httpd_resp_sendstr_chunk(req, "</select>");
    httpd_resp_sendstr_chunk(req, "<br><br>");
    httpd_resp_sendstr_chunk(req, "<input type=\"submit\" value=\"Enviar\">");
    httpd_resp_sendstr_chunk(req, "</form>");
    httpd_resp_sendstr_chunk(req, "<h4><span style=\"color:
#0000ff;\"><strong>Status</strong></span></h4>");
    if(Estado==1)
    {
        httpd_resp_sendstr_chunk(req, "<p>Estado(on/off): ON</p>");
    }
    else
    {
        httpd_resp_sendstr_chunk(req, "<p>Estado(on/off): OFF</p>");
    }

    if(SentidoRotacao==0)
    {
        httpd_resp_sendstr_chunk(req, "<p>Sentido de Rota&ccedil;&atilde;o: Direto</p>");
    }
    else
    {
        httpd_resp_sendstr_chunk(req, "<p>Sentido de Rota&ccedil;&atilde;o: Reverso</p>");
    }

    httpd_resp_sendstr_chunk(req, "<p>Corrente: ");
```

```
    //char string[10];
    sprintf(string,"%.03f",CorrenteMotor);
    httpd_resp_sendstr_chunk(req, string);
    httpd_resp_sendstr_chunk(req, " A</p>");
    sprintf(string,"%.03f",Frequencia);
    httpd_resp_sendstr_chunk(req, "<p>Frenqu&ecirc;ncia de Opera&ccedil;&atilde;o: ");
    httpd_resp_sendstr_chunk(req, string);
    httpd_resp_sendstr_chunk(req, " Hz</p>");
    httpd_resp_sendstr_chunk(req, NULL);
    return ESP_OK;
}

static const httpd_uri_t hello = {
    .uri       = "/hello",
    .method    = HTTP_GET,
    .handler   = hello_get_handler,
    /* Let's pass response string in user
     * context to demonstrate it's usage */
    .user_ctx  = "Hello World!"
};

static const httpd_uri_t dados = {
    .uri       = "/dados",
    .method    = HTTP_GET,
    .handler   = dados_get_handler,
    /* Let's pass response string in user
     * context to demonstrate it's usage */
    .user_ctx  = "Hello World!"
};

static const httpd_uri_t consumopag = {
    .uri       = "/consumo",
    .method    = HTTP_GET,
    .handler   = consumo_get_handler,
    /* Let's pass response string in user
     * context to demonstrate it's usage */
    .user_ctx  = "Hello World!"
};

esp_err_t http_404_error_handler(httpd_req_t *req, httpd_err_code_t err)
{
    if (strcmp("/hello", req->uri) == 0) {
        httpd_resp_send_err(req, HTTPD_404_NOT_FOUND, "/hello URI is not available");
        /* Return ESP_OK to keep underlying socket open */
        return ESP_OK;
    } else if (strcmp("/echo", req->uri) == 0) {
        httpd_resp_send_err(req, HTTPD_404_NOT_FOUND, "/echo URI is not available");
        /* Return ESP_FAIL to close underlying socket */
        return ESP_FAIL;
    }
    /* For any other URI send 404 and close socket */
    httpd_resp_send_err(req, HTTPD_404_NOT_FOUND, "Some 404 error message");
    return ESP_FAIL;
}

static httpd_handle_t start_webserver(void)
{
    httpd_handle_t server = NULL;
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.lru_purge_enable = true;

    // Start the httpd server
    ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port); //aqui inicia o servidor e
cadastra as páginas web
    if (httpd_start(&server, &config) == ESP_OK) {
        // Set URI handlers
        ESP_LOGI(TAG, "Registering URI handlers");
        httpd_register_uri_handler(server, &hello);
        httpd_register_uri_handler(server, &dados);
```

```
        httpd_register_uri_handler(server, &consumopag);
        #if CONFIG_EXAMPLE_BASIC_AUTH
        httpd_register_basic_auth(server);
        #endif
        return server;
    }
    ESP_LOGI(TAG, "Error starting server!");
    return NULL;
}

static void event_handler(void* arg, esp_event_base_t event_base,
                                int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) { //quando inicia o perifério
        ConectadoWifi = 0;
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) { //quando
desconecta da rede
        ConectadoWifi = 0;
        esp_wifi_connect();
        if(MODOWIFI == 1)
        {
            xEventGroupClearBits(wifi_event_group,  CONNECTED_BIT);
        }
        else
        {
            xEventGroupClearBits(s_wifi_event_group,  CONNECTED_BIT);
        }
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ConectadoWifi = 1;
        printf("Conectou no wifi e iniciou servidor.");
        server = start_webserver();
        if(MODOWIFI == 1)
        {
            xEventGroupSetBits(wifi_event_group,    CONNECTED_BIT);
        }
        else
        {
            xEventGroupSetBits(s_wifi_event_group,  CONNECTED_BIT);
        }
    }
}

void wifi_init_sta(void)
{
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t  instance_any_id;
    esp_event_handler_instance_t  instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_got_ip));

    wifi_config_t wifi_config = {
```

```
        .sta = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .password = EXAMPLE_ESP_WIFI_PASS,
            /* Setting a password implies station will connect to all security modes including WEP/WPA.
             * However these modes are deprecated and not advisable to be used. Incase your Access
point
             * doesn't support WPA2, these mode can be enabled by commenting below line */
             .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    ESP_ERROR_CHECK(esp_wifi_start() );

    ESP_LOGI(TAG, "wifi_init_sta finished.");

    /* Waiting until either the connection is established (WIFI_CONNECTED_BIT) or connection failed for
the maximum
     * number of re-tries (WIFI_FAIL_BIT). The bits are set by event_handler() (see above) */
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
            pdFALSE,
            pdFALSE,
            portMAX_DELAY);

    /* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event
actually
     * happened. */
    if (bits & WIFI_CONNECTED_BIT) {
        ESP_LOGI(TAG, "connected to ap SSID:%s password:%s",
                 EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    } else if (bits & WIFI_FAIL_BIT) {
        ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",
                 EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    } else {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }
}

static void initialise_wifi(void)
{
#ifdef CONFIG_EXAMPLE_VALIDATE_SERVER_CERT
    unsigned int ca_pem_bytes = ca_pem_end - ca_pem_start;
#endif /* CONFIG_EXAMPLE_VALIDATE_SERVER_CERT */

#ifdef CONFIG_EXAMPLE_EAP_METHOD_TLS
    unsigned int client_crt_bytes = client_crt_end - client_crt_start;
    unsigned int client_key_bytes = client_key_end - client_key_start;
#endif /* CONFIG_EXAMPLE_EAP_METHOD_TLS */

    ESP_ERROR_CHECK(esp_netif_init());
    wifi_event_group = xEventGroupCreate();
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    sta_netif = esp_netif_create_default_wifi_sta();
    assert(sta_netif);

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_wifi_init(&cfg) );
    ESP_ERROR_CHECK( esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL) );
    ESP_ERROR_CHECK( esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &event_handler, NULL) );
    ESP_ERROR_CHECK( esp_wifi_set_storage(WIFI_STORAGE_RAM) );
    wifi_config_t wifi_config = {
        .sta = {
            .ssid = EXAMPLE_WIFI_SSID,
#if defined (CONFIG_EXAMPLE_WPA3_192BIT_ENTERPRISE)
            .pmf_cfg = {
                .required = true
            },
#endif
```

```
        },
    };
    ESP_LOGI(TAG, "Setting WiFi configuration SSID %s...", wifi_config.sta.ssid);
    ESP_ERROR_CHECK( esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK( esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_set_identity((uint8_t *)EXAMPLE_EAP_ID,
strlen(EXAMPLE_EAP_ID)) );

#if defined(CONFIG_EXAMPLE_VALIDATE_SERVER_CERT) || \
    defined(CONFIG_EXAMPLE_WPA3_ENTERPRISE) || \
    defined(CONFIG_EXAMPLE_WPA3_192BIT_ENTERPRISE)
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_set_ca_cert(ca_pem_start, ca_pem_bytes) );
#endif /* CONFIG_EXAMPLE_VALIDATE_SERVER_CERT */ /* EXAMPLE_WPA3_ENTERPRISE */

#ifdef CONFIG_EXAMPLE_EAP_METHOD_TLS
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_set_cert_key(client_crt_start, client_crt_bytes,\
                client_key_start, client_key_bytes, NULL, 0) );
#endif /* CONFIG_EXAMPLE_EAP_METHOD_TLS */

#if defined CONFIG_EXAMPLE_EAP_METHOD_PEAP || CONFIG_EXAMPLE_EAP_METHOD_TTLS
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EXAMPLE_EAP_USERNAME,
strlen(EXAMPLE_EAP_USERNAME)) );
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EXAMPLE_EAP_PASSWORD,
strlen(EXAMPLE_EAP_PASSWORD)) );
#endif /* CONFIG_EXAMPLE_EAP_METHOD_PEAP || CONFIG_EXAMPLE_EAP_METHOD_TTLS */

#if defined CONFIG_EXAMPLE_EAP_METHOD_TTLS
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_set_ttls_phase2_method(TTLS_PHASE2_METHOD) );
#endif /* CONFIG_EXAMPLE_EAP_METHOD_TTLS */
#if defined (CONFIG_EXAMPLE_WPA3_192BIT_ENTERPRISE)
    ESP_LOGI(TAG, "Enabling 192 bit certification");
    ESP_ERROR_CHECK(esp_wifi_sta_wpa2_set_suiteb_192bit_certification(true));
#endif
    ESP_ERROR_CHECK( esp_wifi_sta_wpa2_ent_enable() );
    ESP_ERROR_CHECK( esp_wifi_start() );
}

static void wpa2_enterprise_example_task(void *pvParameters)
{
    while(1)
    {
        if(salvar==1)
        {
            ESP_ERROR_CHECK(esp_timer_stop(periodic_timer));
            fclose(arquivo);
            char string[100];
            FILE* f = fopen("/spiffs/dados.txt", "a"); //abre o arquivo
            arquivo = f;
            if (f == NULL)
            {
                ESP_LOGE("FILE", "Failed to open file for writing");
                //esp_restart();
                fclose(arquivo);
                //return;
            }
            else
            {
                sprintf(string,"%x/%x/%x;%i:%i:%i;%f;%f;\n",Dia, Mes, Ano, Horas, Minutos,
Segundos,MaiorCorrente,consumo); //Separar dados com ;
                fprintf(f,string);
                fclose(f);
            }
            consumo = 0;
            MaiorCorrente=0;
            struct stat st;
            if (stat("/spiffs/dados.txt", &st) == 0) {
                ESP_LOGI("FILE", "Reading file");
                FILE* f = fopen("/spiffs/dados.txt", "r");
                arquivo = f;
```

```
            if (f == NULL) {
                ESP_LOGE("FILE", "Failed to open file for reading");
                fclose(arquivo);
            }
            else
            {
                char bufferler[100];
                while(fgets(bufferler,sizeof(bufferler),f)!=NULL)
                {
                    printf(bufferler);
                    printf("\n");
                }
                fclose(f);
            }
        }
        ESP_ERROR_CHECK(esp_timer_start_periodic(periodic_timer,  1*1000000));
        salvar=0;
    }
    if(ConectadoWifi == 2)
    {
        if(LigarMotor == 1)
        {
            LigarMotor = 2;
            //vTaskDelay(pdMS_TO_TICKS(500));
            EnableOperation(LigarMotorSentido);
            LerDados(-1);
        }
        else if(LigarMotor == 0)
        {
            LigarMotor = 2;
            //vTaskDelay(pdMS_TO_TICKS(500));
            EnviarShutdown();
            LerDados(-1);
        }
    }
    if(ConectadoWifi == 1)
    {
        ConectadoWifi = 2;
        EnviarFaultStateReset();
        LerDados(-1);
        //vTaskDelay(pdMS_TO_TICKS(5000));
        LerStatus();
        LerDados(-1);
        //vTaskDelay(pdMS_TO_TICKS(500));
        /* frequency40(); */
        frequency60();
        LerDados(-1);
        //vTaskDelay(pdMS_TO_TICKS(500));
        EnviarShutdown(-1);
        LerDados(-1);
        //vTaskDelay(pdMS_TO_TICKS(500));
        LerStatus();
        LerDados(-1);
        //vTaskDelay(pdMS_TO_TICKS(500));
        EnviarSwitchOn();
        LerDados(-1);
        //vTaskDelay(pdMS_TO_TICKS(500));
        LerStatus();
        LerDados(-1);
        /* vTaskDelay(pdMS_TO_TICKS(500));
        EnableOperation();
        LerDados(); */
    }
    vTaskDelay(pdMS_TO_TICKS(1000));
    //float anterior=CorrenteMotor; //pega o valor do registrador antes de ler o registrador
    current();
    LerDados(0);
    //float proximo=CorrenteMotor; //pega o valor depois do registrador estar ligado */
    ReadFrequency();
```

```
1242            LerDados(2);
1243            LerStatus();
1244            LerDados(1);
1245        }
1246    }
1247
1248    static void periodic_timer_callback(void* arg)
1249    {
1250        consumo=consumo+220*CorrenteMotor*1.73*1/3600;
1251        static int contador=0;
1252        static int contador2=0;
1253        fclose(arquivo);
1254        contador++;
1255        contador2++;
1256        Segundos=Segundos+1;
1257        if(Segundos>=60)
1258        {
1259            Segundos=0;
1260            Minutos=Minutos+1;
1261        }
1262        if(Minutos>=60)
1263        {
1264            Minutos=0;
1265            Horas=Horas+1;
1266        }
1267        if(contador2==5)
1268        {
1269            contador2=0;
1270            //printf("Estourou o Timmer\n");
1271            for(unsigned int i = 0; i<18;i++)
1272            {
1273                GraficoCorrente[i] = GraficoCorrente[i+1];
1274                GraficoConsumo[i] = GraficoConsumo[i+1];
1275                sprintf(GraficoHorarios[i+1],GraficoHorarios[i+2]);
1276            }
1277            GraficoCorrente[18] = CorrenteMotor;
1278            GraficoConsumo[18] = consumo;
1279            sprintf(GraficoHorarios[18],"\"%i:%i:%i\"",Horas,Minutos,Segundos);
1280        }
1281
1282        if(contador==5)
1283        {
1284            contador=0;
1285            salvar=1;
1286        }
1287
1288        //printf("%f\n", maiorcorrente);
1289        //uint8_t contador=0;
1290
1291    }
1292
1293    void Init_SPIFFS(void)
1294    {
1295        esp_vfs_spiffs_conf_t conf = {
1296          .base_path = "/spiffs",
1297          .partition_label = NULL,
1298          .max_files = 5,
1299          .format_if_mount_failed = true
1300        };
1301        esp_err_t ret = esp_vfs_spiffs_register(&conf);
1302
1303        if (ret != ESP_OK) {
1304            if (ret == ESP_FAIL) {
1305                ESP_LOGE(TAG, "Failed to mount or format filesystem");
1306            } else if (ret == ESP_ERR_NOT_FOUND) {
1307                ESP_LOGE(TAG, "Failed to find SPIFFS partition");
1308            } else {
1309                ESP_LOGE(TAG, "Failed to initialize SPIFFS (%s)", esp_err_to_name(ret));
1310            }
```

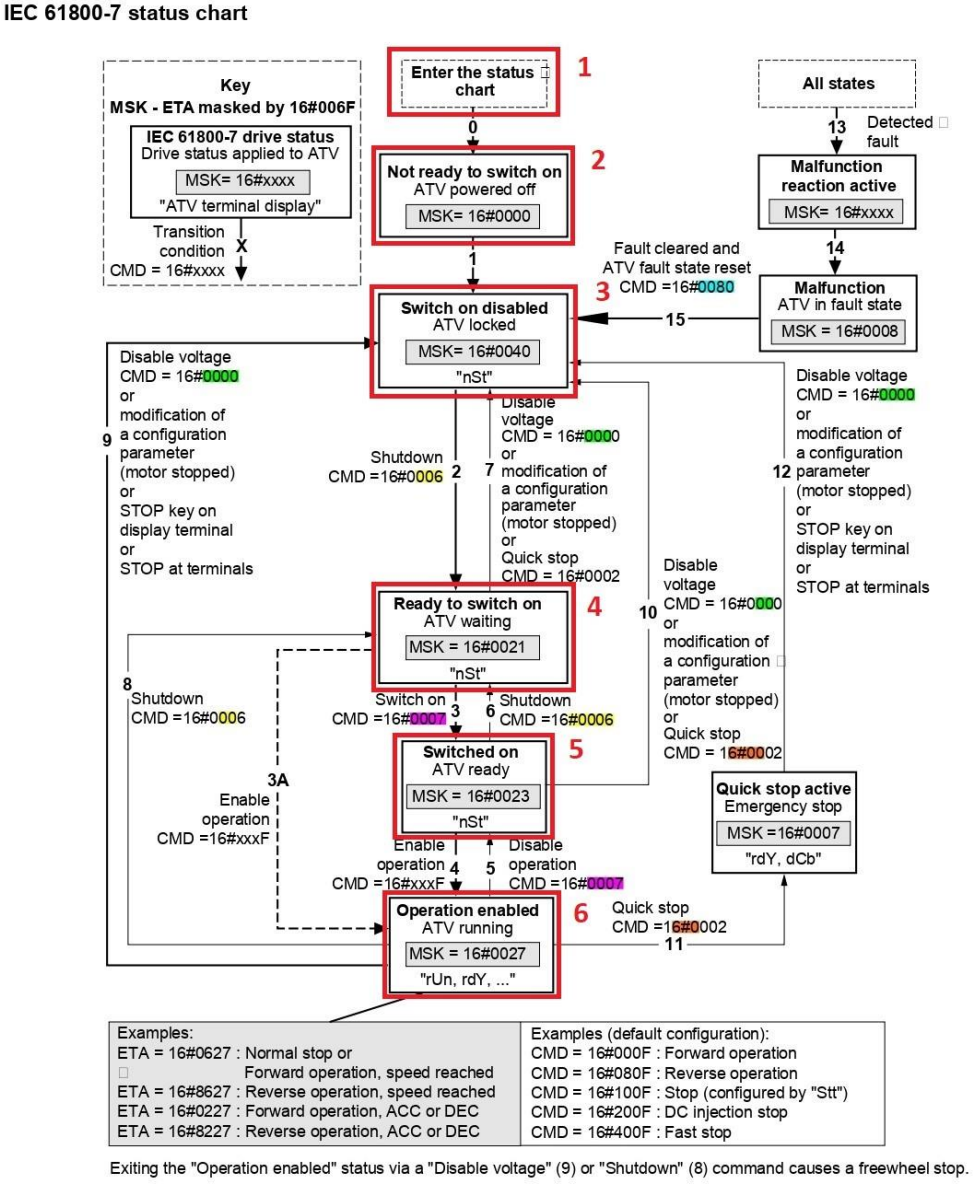
```
1311            return;
1312        }
1313        size_t total = 0, used = 0;
1314        ret = esp_spiffs_info(conf.partition_label, &total, &used);
1315        if (ret != ESP_OK) {
1316            ESP_LOGE(TAG, "Failed to get SPIFFS partition information (%s). Formatting...",
1317    esp_err_to_name(ret));
1318            esp_spiffs_format(conf.partition_label);
1319            return;
1320        } else {
1321            ESP_LOGI(TAG, "Partition size: total: %d, used: %d", total, used);
1322        }
1323
1324        // Check consistency of reported partiton size info.
1325        if (used > total) {
1326            ESP_LOGW(TAG, "Number of used bytes cannot be larger than total. Performing SPIFFS_check().");
1327            ret = esp_spiffs_check(conf.partition_label);
1328            // Could be also used to mend broken files, to clean unreferenced pages, etc.
1329            // More info at https://github.com/pellepl/spiffs/wiki/FAQ#powerlosses-contd-when-should-i-run-
1330    spiffs_check
1331            if (ret != ESP_OK) {
1332                ESP_LOGE(TAG, "SPIFFS_check() failed (%s)", esp_err_to_name(ret));
1333                return;
1334            } else {
1335                ESP_LOGI(TAG, "SPIFFS_check() successful");
1336            }
1337        }
1338    }
1339
1340    void app_main(void)
1341    {
1342        for(unsigned int i = 0; i<20;i++)
1343        {
1344            GraficoCorrente[i] = 0;
1345            if(i<19)
1346            {
1347                sprintf(GraficoHorarios[i],"\"00:00:00\"");
1348            }
1349            else{
1350                sprintf(GraficoHorarios[i],"\"\"");
1351            }
1352        }
1353        ESP_ERROR_CHECK( nvs_flash_init() );
1354        ESP_ERROR_CHECK(esp_netif_init());
1355
1356        Init_SPIFFS();
1357
1358        unlink("/spiffs/dados.txt"); //serve pra deletar
1359
1360        const esp_timer_create_args_t periodic_timer_args = { //isso configura o periférico
1361                .callback = &periodic_timer_callback,
1362                /* name is optional, but may help identify the timer when debugging */
1363                .name = "periodic"
1364        };
1365        //esp_timer_handle_t         periodic_timer;
1366        ESP_ERROR_CHECK(esp_timer_create(&periodic_timer_ args, &periodic_timer)); //criar o timer
1367
1368        ESP_ERROR_CHECK(esp_timer_start_periodic(periodic_timer, 1*1000000)); //inicia
1369
1370        uint8_t data[3];
1371        ESP_ERROR_CHECK(i2c_master_init());
1372        ESP_LOGI(TAG, "I2C initialized successfully");
1373
1374    //mpu9250_register_write_byte(0x00, 0x20); //define os segundos  // não apagar, usado na primeira vez
1375    //mpu9250_register_write_byte(0x01, 0x25); //define as minutors
1376    //mpu9250_register_write_byte(0x02, 0x16); //define as horas
1377    //mpu9250_register_write_byte(0x04, 0x30); //define o dia
1378    //mpu9250_register_write_byte(0x05, 0x09); //define o mês
1379    //mpu9250_register_write_byte(0x06, 0x22); //define o ano
```

```
ESP_ERROR_CHECK(mpu9250_register_read(0x00, data, 3));

    //ESP_LOGI(TAG, "WHO_AM_I = %X", data[0]);
    for(unsigned int i=0;i<3;i++)
    {
        if(i==0)
        {
            ESP_LOGI(TAG, "Segundos = %X", data[i]); //printa os 3
        }
        if(i==1)
        {
            ESP_LOGI(TAG, "Minutos = %X", data[i]); //printa os 3
        }
        if(i==2)
        {
            ESP_LOGI(TAG, "Horas = %X", data[i]); //printa os 3
        }
    }
    float segundos = data[0];
    float minutos = data[1];
    float horas = data[2];
    Segundos = ((uint8_t)(segundos/16)*10)+(uint8_t)(((segundos/16)-(uint8_t)(segundos/16))*16);
    Minutos =  ((uint8_t)(minutos/16)*10)+(uint8_t)(((minutos/16)-(uint8_t)(minutos/16))*16);
    Horas = ((uint8_t)(horas/16)*10)+(uint8_t)(((horas/16)-(uint8_t)(horas/16))*16);

    ESP_ERROR_CHECK(mpu9250_register_read(0x04, data, 3));

     for(unsigned int i=0;i<3;i++)
    {
         if(i==0)
        {
            ESP_LOGI(TAG, "Dia = %X", data[i]); //printa os 3
        }
        if(i==1)
        {
            ESP_LOGI(TAG, "Mês = %X", data[i]); //printa os 3
        }
        if(i==2)
        {
            ESP_LOGI(TAG, "Ano = %X", data[i]); //printa os 3
        }
    }
    Dia = data[0];
    Mes = data[1];
    Ano = data[2];
    //Check if Two Point or Vref are burned into eFuse
    check_efuse();

    //Configure ADC
    if (unit == ADC_UNIT_1) {
        adc1_config_width(width);
        adc1_config_channel_atten(channel,  atten);
    } else {
        adc2_config_channel_atten((adc2_channel_t)channel,  atten);
    }

    //Characterize ADC
    adc_chars = calloc(1, sizeof(esp_adc_cal_characteristics_t));
    esp_adc_cal_value_t val_type = esp_adc_cal_characterize(unit, atten, width, DEFAULT_VREF,
adc_chars);
    print_char_val_type(val_type);

            // Set UART log level
    esp_log_level_set("UART", ESP_LOG_INFO);

    ESP_LOGI("UART", "Start RS485 application test and configure UART.");

    // Install UART driver (we don't need an event queue here)
    // In this example we don't even use a buffer for sending data.
```

```
1449        ESP_ERROR_CHECK(uart_driver_install(uart_num, BUF_SIZE * 2, 0, 0, NULL, 0));
1450
1451        // Configure UART parameters
1452        ESP_ERROR_CHECK(uart_param_config(uart_num,  &uart_config));
1453
1454        ESP_LOGI("UART", "UART set pins, mode and install driver.");
1455
1456        // Set UART pins as per KConfig settings
1457        ESP_ERROR_CHECK(uart_set_pin(uart_num, ECHO_TEST_TXD, ECHO_TEST_RXD, ECHO_TEST_RTS,
1458    ECHO_TEST_CTS));
1459
1460        // Set RS485 half duplex mode
1461        ESP_ERROR_CHECK(uart_set_mode(uart_num,  UART_MODE_RS485_HALF_DUPLEX));
1462
1463        // Set read timeout of UART TOUT feature
1464        ESP_ERROR_CHECK(uart_set_rx_timeout(uart_num,  ECHO_READ_TOUT));
1465
1466        //ESP_ERROR_CHECK(esp_event_loop_create_default());
1467        if(MODOWIFI == 1)
1468        {
1469            initialise_wifi();
1470            xTaskCreate(&wpa2_enterprise_example_task, "wpa2_enterprise_example_task", 4096, NULL, 5,
1471    NULL);
1472            vTaskPrioritySet( wpa2_enterprise_example_task, tskIDLE_PRIORITY + 2);
1473        }
1474        else
1475        {
1476            wifi_init_sta();
1477            xTaskCreate(&wpa2_enterprise_example_task, "wpa2_enterprise_example_task", 4096, NULL, 5,
1478    NULL);
1479            vTaskPrioritySet( wpa2_enterprise_example_task, tskIDLE_PRIORITY + 2);
1480        }
1481
1482    }
```

ANEXO A – Fluxogram para acionamento do inversor de frequência

Figura 23 – Página 10 do manual



Fonte: Adaptado de Schneider (2009a)

ANEXO B – Funções Modbus Inversor de Frequência

Figura 24 – Funções Modbus inversor de frequência

Code (decimal)	Function name	Broadcasting	Max. value of N	Modbus standard name
3	Read N output words	NO	29 words max.	Read Holding Registers
6	Write one output word	YES	–	Preset Single Register
16	Write N output words	YES	27 words max.	Preset Multiple Regs
43	Identification	NO	–	Read Device Identification

Fonte: Schneider (2009b)

B.1 Supervisão e controle em modo de linha

Figura 25 – Dados das Funções Modbus inversor de frequência

CMD control word (W8501)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Fault state reset	0	0	0	Enable operation	Quick stop (active at 0)	Enable voltage	Switch on

bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
(1)	(1)	(1)	(1)	(1)	0	0	0

(1) This bit action depends on the LAC "Access levels" parameter and the functions configured by the user.
For example, to use bit 15 to switch the ramp, simply configure LAC = L3 (Access to advanced functions and management of mixed modes) and set the "Ramp switching rPS" configuration parameter to Cd15.

Command	Transition address	Final state	bit 7	bit 3	bit 2	bit 1	bit 0	Typical value of CMD (W8501)
			Reset	Enable operation	Quick stop	Enable voltage	Switch on	
Shut down	2, 6, 8	Ready to switch on	x	x	1	1	0	16#0006
Switch on	3	Switched on	x	x	1	1	1	16#0007
Enable operation	4	Operation enabled	x	1	1	1	1	16#000F
Disable operation	5	Switched on	x	0	1	1	1	16#0007
Disable voltage	7, 9, 10, 12	Switch on disabled	x	x	x	0	x	16#0000
Quick stop	11	Quick stop active	x	x	0	1	x	16#0002
	7, 10	Switch on disabled						
Fault state reset	15	Switch on disabled	0 → 1	x	x	x	x	16#0080

x: State not significant
0 → 1: Change from 0 to 1

Fonte: Schneider (2009a)

ANEXO C – Status Inversor de Frequência

Figura 26 – *Eta Status* Inversor de frequência

ETA status word (W3201)

bit 7	bit 6	bit 5	bit 4
Alarm	Switch on disabled	Quick stop active at 0	0

bit 3	bit 2	bit 1	bit 0
Malfunction	Operation enabled	Switched on	Ready to switch on

bit 15	bit 14	bit 13	bit 12
Direction of rotation	Stop via STOP key	0	0

bit 11	bit 10	bit 9	bit 8
Reference exceeded	Reference reached	Forced local mode (active at 0)	0

State	bit 6	bit 5	bit 3	bit 2	bit 1	bit 0	MSK = ETA (W3201) masked by 16#006F
	Switch on disabled	Quick stop	Malfunction	Operation enabled	Switched on	Ready to switch on	
Not ready to switch on	0	x	0	0	0	0	16#0000 16#0020
Switch on disabled	1	x	0	0	0	0	16#0040 16#0060
Ready to switch on	0	1	0	0	0	1	16#0021
Switched on	0	1	0	0	1	1	16#0023
Operation enabled	0	1	0	1	1	1	16#0027
Malfunction	0	x	1	0	0	0	16#0008 16#0028
Malfunction reaction active	0	x	1	1	1	1	16#000F 16#002F
Quick stop active	0	0	0	1	1	1	16#0007

x: State not significant

Fonte: Schneider (2009a)

C.1 Status Word

Figura 27 – Registrador de *Status* inversor de frequência

Modbus address	CANopen address	Code	Read/ Write	Name/Description/Possible values	
3201	2002 / 2	**ETA**	R	**Status word** bit 0: Ready to switch on bit 1: Switched on bit 2: Operation enabled bit 3 = 0: No detected fault bit 3 = 1: Malfunction, detected fault (FAI) bit 4: Voltage disabled (still equals 0) bit 5: Quick stop bit 6: Switch on disabled bit 7 = 0: No alarm bit 7 = 1: Alarm present	DATA 4
				bit 8: Reserved bit 9 = 0: Forced local mode in progress (FLO) bit 9 = 1: No forced local mode bit 10 = 0: Reference not reached (transient state) bit 10 = 1: Reference reached (steady state) bit 11 = 0: LFRD reference normal bit 11 = 1: LFRD reference exceeded (< LSP or > HSP) Note: LFRD is expressed in rpm, LSP and HSP in Hz bits 12 and 13: Reserved bit 14 = 0: No stop imposed by STOP key on built-in keypad or on the remote display terminal bit 14 = 1: Stop imposed by STOP key on built-in keypad or on the remote display terminal bit 15 = 0: Forward rotation (output frequency) bit 15 = 1: Reverse rotation (output frequency)	DATA 3

Fonte: Adaptado de Schneider (2009a)