

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA EM ELETRÔNICA**

JOÃO VICTOR MACIEL DA VEIGA

MONITORAMENTO EM TEMPO REAL DE INVERSORES FOTOVOLTAICOS

FLORIANÓPOLIS, 2026.

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA EM ELETRÔNICA**

JOÃO VICTOR MACIEL DA VEIGA

MONITORAMENTO EM TEMPO REAL DE INVERSORES FOTOVOLTAICOS

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro em Eletrônica.

Orientador:
Prof. Jony Laureano Silveira, Dr.

Coorientador:
Fábio Cabral Pacheco, Tecg

FLORIANÓPOLIS, 2026.

Ficha de identificação da obra elaborada pelo autor.

Veiga, João Victor Maciel da
Monitoramento em Tempo Real de Inversores Fotovoltaicos
/ João Victor Maciel da Veiga; orientação de Jony Laureano
Silveira; coorientação de Fábio Cabral Pacheco.
- Florianópolis, SC, 2026.
73 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal
de Santa Catarina, Câmpus Florianópolis. Bacharelado
em Engenharia Eletrônica. Departamento Acadêmico
de Eletrônica.
Inclui Referências.

1. Energia solar. 2. Monitoramento em tempo real.
3. SunSpec ModBus. 4. ESP32-C6. I. Laureano Silveira,
Jony . II. Cabral Pacheco, Fábio. III. Instituto Federal
de Santa Catarina. IV. Monitoramento em Tempo Real
de Inversores Fotovoltaicos.

MONITORAMENTO EM TEMPO REAL DE INVERSORES FOTOVOLTAICOS

JOÃO VICTOR MACIEL DA VEIGA

Este trabalho foi julgado adequado para obtenção do título de Bacharel em Engenharia Eletrônica e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 16 de julho, 2026

Banca Examinadora:

Documento assinado digitalmente
 JONY LAUREANO SILVEIRA
Data: 11/08/2026 17:42:07-0300
Verifique em <https://validar.iti.gov.br>

Jony Laureano da Silveira, Dr. Eng.
DAELN / CP-FLN / IFSC
Orientador

Documento assinado digitalmente
 FABIO CABRAL PACHECO
Data: 11/08/2026 16:23:11-0300
Verifique em <https://validar.iti.gov.br>

Fábio Cabral Pacheco, Tecg.
DAELN / CP-FLN / IFSC
Coorientador

Documento assinado digitalmente
 PAULO RICARDO TELLES RANGEL
Data: 07/08/2026 21:17:33-0300
Verifique em <https://validar.iti.gov.br>

Paulo Ricardo Telles Rangel, Msc. Eng.
DAELN / CP-FLN / IFSC

Documento assinado digitalmente
 RAFAEL NILSON RODRIGUES
Data: 11/08/2026 13:48:18-0300
Verifique em <https://validar.iti.gov.br>

Rafael Nilson Rodrigues, Dr. Eng.
DAE / CP-FLN / IFSC

Não há exemplo maior de dedicação do que o da nossa família.
À minha querida família, que tanto admiro,
dedico o resultado do esforço realizado ao longo deste percurso.

AGRADECIMENTOS

Gostaria de começar os agradecimentos aos meus pais, Carlos e Viviane, por todo carinho suporte e atenção dada a minha vida toda, em especial no desenvolvimento desse trabalho. À minha noiva, Jéssica, pelo carinho e motivação dados. À meu irmão, André, pela parceria que sempre tivemos. Também agradeço a todos as outras pessoas de minha família e amigos que sempre estiveram presentes em diversos momentos de minha vida.

Agradeço também aos Orientadores desse trabalho, ao Cabral por dedicar seu tempo além de seus compromissos para auxiliar nas eventuais duvidas durante o processo de realização do trabalho. E ao professor Jony por aceitar o convite e auxiliar nas inúmeras correções na elaboração desse documento.

Por ultimo agradeço a todos os professores e demais docentes do IFSC que auxiliaram na minha formação e foram fundamentais para o entendimento dos conceitos necessário para desenvolvimento desse projeto. Em especial ao professor Rafael que foi meu orientador no projeto de pesquisa do PGEN que foi crucial para meu desenvolvimento acadêmico e profissional.

“O homem não teria alcançado o possível se,
repetidas vezes, não tivesse tentado o impossível.”

- Max Weber

RESUMO

Em virtude do crescente emprego de painéis fotovoltaicos, faz-se indispensável o controle e o monitoramento desses dispositivos. Para tanto, o protocolo de comunicação SunSpec ModBus mostra-se uma boa alternativa para o monitoramento em tempo real, permitindo, por conseguinte, o controle da geração. Este trabalho apresenta o desenvolvimento de um sistema que possibilita o monitoramento de inversores. Para tanto, foi utilizado o microcontrolador ESP32-C6, equipado com o protocolo SunSpec ModBus para comunicação com o inversor e o protocolo MQTT para comunicação com o servidor. O servidor emprega o ThingsBoard como fundamento, instalado em uma máquina Linux. Diante dos resultados obtidos nesse trabalho, é possível acompanhar o comportamento do sistema de geração fotovoltaica em tempo real por meio de gráficos e tabelas apresentados em um dashboard. Dessa forma, foi desenvolvido um hardware que se mostrou eficaz na aquisição e envio de dados.

Palavras-chave: Energia solar. Monitoramento em tempo real. SunSpec ModBus. ESP32-C6.

ABSTRACT

Due to the growing use of photovoltaic panels, it has become essential to control and monitor these devices. To this end, the SunSpec ModBus communication protocol proves to be a good alternative for real-time monitoring, thereby enabling control of power generation. This paper presents the development of a system that enables the monitoring of inverters. To this end, the ESP32-C6 microcontroller was used, equipped with the SunSpec ModBus protocol for communication with the inverter and the MQTT protocol for communication with the server. The server is based on ThingsBoard, installed on a Linux machine. Based on the results obtained in this study, it is possible to monitor the behavior of the photovoltaic generation system in real time through graphs and tables displayed on a dashboard. Thus, hardware was developed that proved effective in acquiring and transmitting data.

Keywords: Solar Energy. Real-time monitoring. SunSpec ModBus. ESP32-C6.

LISTA DE FIGURAS

Figura 1 – site PGEN	15
Figura 2 – Geração excedente	16
Figura 3 – Geração excedente	17
Figura 4 – Diagrama de conexão do inversor <i>on-grid</i>	18
Figura 5 – Inversor	19
Figura 6 – Arquitetura ThingsBoard	24
Figura 7 – Arquitetura ESP32-C6	26
Figura 8 – OFDMA	27
Figura 9 – Funcionamento IDF	28
Figura 10 – Componentes presentes na IDF	29
Figura 11 – Simulador	32
Figura 12 – qModMaster	32
Figura 13 – Log de comunicação do OModSim durante o teste	33
Figura 14 – Dados obtidos no qModMaster através do teste	34
Figura 15 – Loop USB	35
Figura 16 – Diferenças entre RS485, RS232 e TTL	36
Figura 17 – Interceptação dos dados	37
Figura 18 – Fluxograma de aquisição dos dados	38
Figura 19 – OModSim com os registradores aleatórios	40
Figura 20 – Conversão de dados em JSON	42
Figura 21 – Converter tabela para JSON	43
Figura 22 – Cabeçalho Excel	45
Figura 23 – Fluxograma do <i>script</i>	47
Figura 24 – Aba de cadeia de regras no ThingsBoard	47
Figura 25 – Conexão ESP32 e Computador via RS485	49
Figura 26 – Exemplo de conexão no manual	49
Figura 27 – Conexão física realizada	49
Figura 28 – Esquema da conexão física	50
Figura 29 – Zoom do esquema da conexão física	50
Figura 30 – Requisição feita pelo ESP	51
Figura 31 – Resposta dada pelo inversor	51
Figura 32 – Configuração usada no OModSim	53
Figura 33 – Esquemático da fonte chaveada 3.3 Volts	54
Figura 34 – Esquemático do conversor para RS485	55
Figura 35 – Esquemático do ESP32-C6 e ligações base	56
Figura 36 – Bottom Layer	56
Figura 37 – Top Layer	56

Figura 38 – Layout KiCAD	57
Figura 39 – Confeção na fresa	58
Figura 40 – Fresagem finalizada	58
Figura 41 – Fonte 3.3V soldada	59
Figura 42 – Ripple da fonte	60
Figura 43 – RS485 soldado	60
Figura 44 – Adaptação	61
Figura 45 – Sinal antes do divisor	62
Figura 46 – Sinal depois do divisor	62
Figura 47 – Circuito completo	63
Figura 48 – CC	64
Figura 49 – Potencias	65
Figura 50 – Corrente	66
Figura 51 – Tensão CA	66
Figura 52 – Energia acumulada e Temperatura	67
Figura 53 – Energia acumulada entre dias	67
Figura 54 – Qualidade da energia	68
Figura 55 – Potencia Ativa e Reativa	68
Figura 56 – Operação com baixa energia gerada	69
Figura 57 – Operação com baixa energia gerada	69
Figura 58 – Operação com baixa energia gerada 2	70

LISTA DE TABELAS

Tabela 1 – Tabela de identificação ID 1	20
Tabela 2 – Tabela de dados ID 11X	21
Tabela 3 – Tabela de dados ID 10X	22
Tabela 4 – Detalhes da comunicação	52
Tabela 5 – Controle do modo de inicialização do chip	62

LISTA DE ABREVIATURAS E SIGLAS

DER *Distributed Energy Resource.*

ESP-IDF *Espressif Internet of things Development Framework.*

IP *Internet Protocol.*

ISA *Instruction Set Architecture.*

JSON *JavaScript Object Notation.*

MQTT *Message Queuing Telemetry Transport.*

PGEN *Portal de Gerenciamento Energético Nacional.*

PTH *Pin Through Hole.*

RFEPCT *Rede Federal de Educação Profissional, Científica e Tecnológica.*

RTOS *Real-Time Operating System.*

RTU *Remote Terminal Unit.*

SMD *Surface Mounted Device.*

TBEL *ThingsBoard expression language.*

TCP *Transmission Control Protocol.*

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Justificativa	15
1.2	Definição do Problema	16
1.3	Objetivo Geral	17
1.4	Objetivos Específicos	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Inversor	18
2.2	Protocolo Sunspec ModBus	19
2.3	Servidor	24
2.4	Microcontrolador	25
2.5	ESP-IDF	28
3	METODOLOGIA	30
3.1	ModBus	30
3.1.1	Comunicação entre qModMaster e o Simulador em Python	31
3.1.2	Comunicação entre qModMaster e o OModSim	33
3.1.3	Comunicação entre o ESP32 e o Simulador em python	35
3.1.4	Comunicação entre o ESP32 e o OModSim	36
3.1.5	Desenvolvimento do algoritmo	38
3.2	Thingsboard	45
3.3	Inversor Fronius	48
3.4	Hardware	54
3.4.1	Esquemático	54
3.4.2	Layout	56
3.4.3	Confecção e Testes	57
4	RESULTADOS E DISCUSSÕES	64
5	CONSIDERAÇÕES FINAIS	71
5.1	Sugestões para trabalhos futuros	71
	REFERÊNCIAS	72

1 INTRODUÇÃO

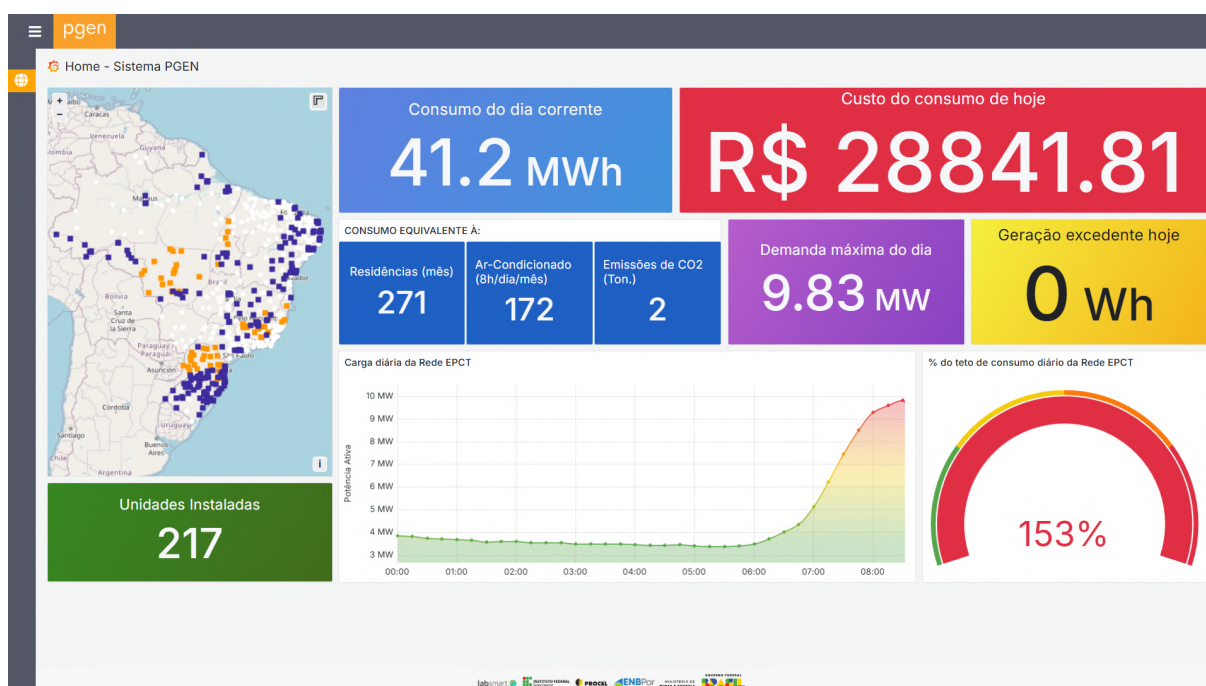
De acordo com a ANEEL (2025), a rede elétrica brasileira tem experimentado um crescimento substancial na geração solar, representando mais de um terço da expansão da rede no período de janeiro a novembro de 2025. A energia solar é caracterizada como uma fonte de energia limpa, porém, sua geração está sujeita às condições climáticas, especialmente à incidência solar. A variabilidade climática compromete a previsibilidade e o controle da geração de energia.

O objetivo deste trabalho é desenvolver um dispositivo para monitoramento em tempo real da geração fotovoltaica por meio da comunicação com os inversores via protocolo ModBus. Dessa forma, será possível identificar com precisão a quantidade e o momento da geração de energia, tanto para fins financeiros quanto para integração com a rede e administração do *curtailment*. Uma das definições de *curtailment*, segundo Bird *et al.* (2016), é o corte ou a redução da geração de energias renováveis, normalmente as solares e eólicas, devido à capacidade da rede de energia absorver esses volumes. Esse cenário acarreta perdas no investimento, visto que tais energias renováveis são disponibilizadas em determinados intervalos de tempo e, caso não sejam utilizadas para a geração de energia no momento em que estão disponíveis, o potencial será perdido.

Adicionalmente, há uma demanda do sistema Portal de Gerenciamento Energético Nacional [PGEN], que faz parte do Projeto Energias Renováveis e Eficiência Energética, na Rede Federal de Educação Profissional, Científica e Tecnológica [RFEPCT] o EnergIFE. Atualmente, o projeto mais relevante desse sistema é o PGEN Monitor, um site que fornece informações de consumo em tempo real. A página inicial do site, ilustrada na Figura 1, exibe a localização das unidades e a quantidade total instalada, além do consumo total de todas as unidades operantes e seus equivalentes, como dinheiro, quantidade de residências, ar-condicionado e emissão de CO₂. O site também apresenta a somatória das demandas de cada unidade e de sua contribuição para a rede, com suas respectivas gerações (majoritariamente solar). O gráfico mostra o consumo total, distribuído em intervalos de 5 minutos. Ademais, o site permite que os gestores de cada unidade da RFEPCT apliquem técnicas de eficiência energética com mais facilidade, pois ele também conta com páginas individuais para cada unidade.

Atualmente, o monitoramento realizado pelo PGEN cobre cerca de um terço da rede, disponibilizando, em tempo real, dados do consumo total de cada subestação instalada — normalmente equivalente ao consumo de todo o campus —, e como expansão desse sistema se tem a ideia de desenvolver o PGEN-Solar, que disponibilizará, em tempo real e de forma pública, os dados da geração de energia solar

Figura 1 – site PGEN



Fonte: <https://pgen.app/>(LabSmart, [202-])

em cada local. Dessa forma, o presente trabalho propõe o desenvolvimento de um protótipo inicial, a fim de servir de base para o novo projeto.

1.1 Justificativa

A implementação da energia fotovoltaica tem se desenvolvido bastante nos últimos tempos, tornando-se mais presente em diversas camadas da rede elétrica. No entanto, esse método de geração apresenta uma particularidade: a variação de geração ao longo do dia, o que pode gerar instabilidade na rede.

Um fator a ser considerado é que picos de geração sobrecarregam a rede de forma semelhante aos picos de consumo. Assim, alguns locais com geração solar muito grande, que podem acabar gerando mais que a própria demanda contratada, precisam pagar multas por ultrapassar essa demanda, mesmo nunca consumindo mais que ela. O monitoramento pode auxiliar um sistema que use essa energia a não injetá-la na rede nesses momentos ou controlar os inversores de modo a não exceder a potência limite.

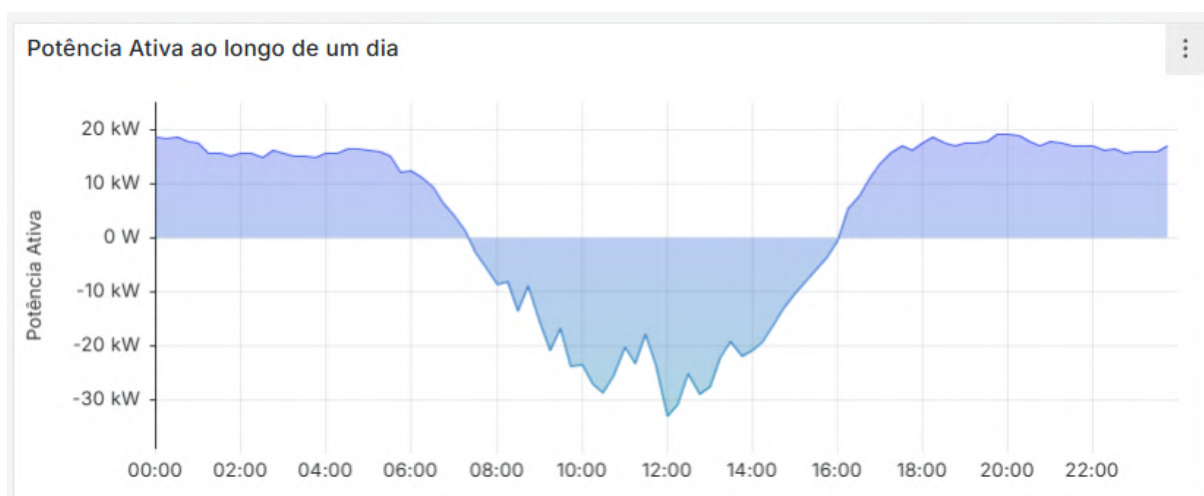
Outra questão da geração fotovoltaica é a falsa sensação de energia gratuita, pois a geração é baseada na incidência de raios solares nas placas, o que, em si, não gera gastos. Porém, os painéis e os inversores têm custos de fabricação, instalação e manutenção que não podem ser ignorados. Com um monitoramento preciso, é possível determinar se e quando a energia gerada é, de fato, lucrativa para quem a instalou.

Ademais, o conjunto de informações adquiridas pelo monitoramento possibilita uma melhor compreensão acerca da geração realizada, incluindo a eficiência da energia gerada e sua conversão, bem como o fator de potência e a frequência. Tal fato contribui para a análise da qualidade da energia e de sua variação diante das condições climáticas ao decorrer do dia.

1.2 Definição do Problema

A demanda para monitoramento da geração de energia solar foi sendo percebida ao visualizar as curvas de carga pelo PGEN Monitor principalmente em dias de alta geração e baixo consumo, que pode ser visualizada na figura 2 que apresenta a curva de carga em um domingo num campus com geração solar, onde em determinados momentos do dia a geração ultrapassa o consumo sendo mostrada como potência negativa, porém só com isso não dá para separar o que é geração e o que está sendo consumido assim não se sabe o que pode ser reduzido e quanto está sendo gerado para calcular a eficiência do investimento em geração solar.

Figura 2 – Geração excedente



Fonte: [https://pgen.app/\(LabSmart, \[202-\]\)](https://pgen.app/(LabSmart, [202-]))

Além disso em dias típicos, onde o consumo é maior que a geração, como pode ser visualizada na figura 3 que apresenta a curva de carga do mesmo campus no dia seguinte, a geração fica completamente indistinguível do consumo assim o quanto realmente está sendo consumido é mascarado pela geração. Dessa forma com um monitoramento da geração e cruzamento de dados seria possível saber o quanto é consumido para assim analisar se tem necessidade de redução e também entender melhor se o investimento com placas solares está sendo lucrativo pois além da energia gerada essa diferença entre o consumo sem solar e com solar pode permitir uma contratação de demanda menor ainda mais considerando que a incidência solar é

maior justamente nos meses mais quentes, que no Brasil normalmente significa mais consumo devido aos gastos com refrigeração.

Figura 3 – Geração excedente



Fonte: [https://pgen.app/\(LabSmart, \[202-\]\)](https://pgen.app/(LabSmart, [202-]))

Tendo em vista a necessidade de monitorar a geração solar, resta saber como integrar o monitoramento aos inversores já instalados de maneira regular e robusta.

1.3 Objetivo Geral

A implementação de um sistema de monitoramento em tempo real, utilizando o protocolo ModBus com o microcontrolador, visando a coleta de dados de painéis fotovoltaicos e o envio dos mesmos para um banco de dados.

1.4 Objetivos Específicos

Com o objetivo geral definido são necessários os seguintes objetivos específicos para alcançá-lo:

- Estudar o protocolo ModBus e desenvolver um método para aquisição de dados em intervalos regulares;
- Configurar um ambiente de testes para comunicação do microcontrolador com um inversor, e testar a frequência limite da aquisição de dados;
- Confeccionar uma placa de circuito integrado para o microcontrolador acessar os periféricos necessários para comunicação com o inversor tendo mais confiabilidade;
- Configurar um servidor para envio das informações lidas no inversor para um banco de dados
- Gerar gráficos para visualização dos dados em tempo real.

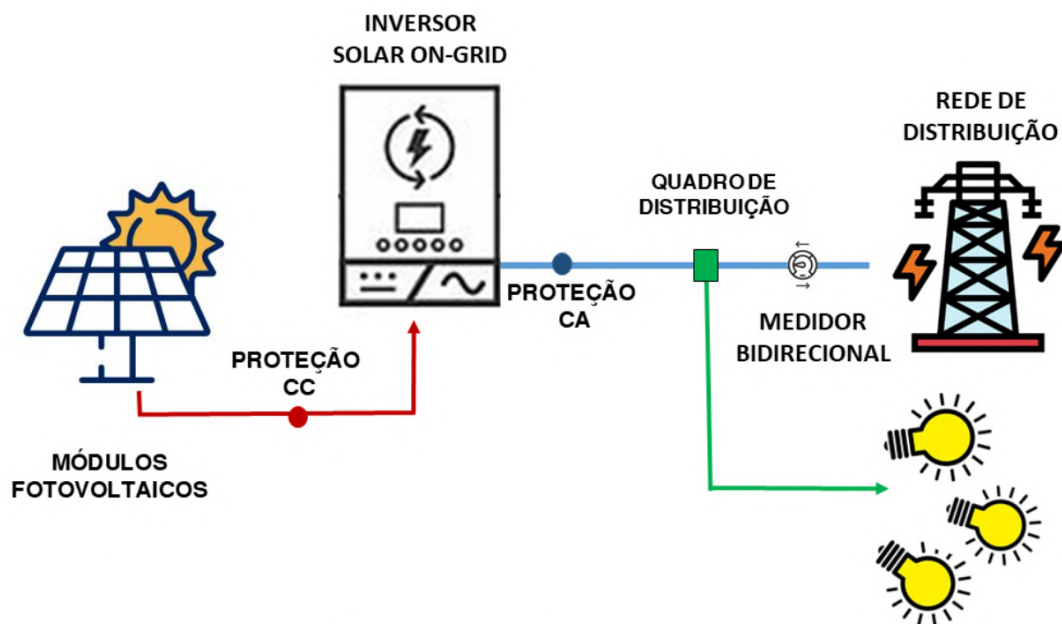
2 FUNDAMENTAÇÃO TEÓRICA

Esta seção do trabalho tem por objetivo explicar o funcionamento dos dispositivos e protocolos utilizados necessários para o desenvolvimento do protótipo e assim consiga alcançar os objetivos de forma clara.

2.1 Inversor

As placas solares são capazes de gerar energia apenas em corrente contínua. Por sua vez, a distribuição de energia é feita em corrente alternada. Para que haja a conversão de energia entre as correntes contínua e alternada, é necessário que seja utilizado um inversor. A Figura 4 apresenta um esquema ilustrativo de como um inversor *on-grid* opera. Dada sua função central no controle da geração, o inversor deve mensurar diversos parâmetros suscetíveis de análise.

Figura 4 – Diagrama de conexão do inversor *on-grid*



Adaptado de ENERGÊS (2020)

A geração solar pode ser realizada por meio de três principais tipos de inversores: *on-grid*, *off-grid* e híbrido. Neste projeto, será utilizado o primeiro tipo. Os inversores *on-grid* são caracterizados pela ausência de bateria, sendo necessária a presença de um medidor bidirecional. Dessa forma, quando há geração excedente, ela é injetada na rede elétrica, e, quando os painéis não produzem energia suficiente, a energia necessária é fornecida pela rede elétrica. Os inversores *off-grid*, por sua vez, demandam o uso de baterias devido à sua natureza desconectada da rede elétrica. O excedente de energia é, portanto, armazenado para utilização em períodos de geração reduzida. Em geral, tais sistemas são inteiramente dependentes de energia

própria, uma vez que não estão conectados à rede. Em síntese, os inversores híbridos são constituídos pela combinação dos inversores anteriores. Em certos casos, há a presença de dois tipos de inversores no interior do dispositivo, que conta com um sistema de controle para determinar qual tipo deve ser utilizado. Ademais, o dispositivo está conectado à rede elétrica e pode injetar ou receber energia, caso necessário. Além disso, o inversor híbrido também possui baterias, que armazenam e utilizam a própria energia.(ENERGÊS, 2020)

Para este trabalho foi usado o inversor da Fronius Galvo 2.5-1, que opera no sistema *on-grid* monofásico dimensionado para 2.5 kW, este foi empregado sobretudo em virtude de sua disponibilidade no laboratório e de sua capacidade de operar com o protocolo Sunspec ModBus, o que possibilita sua utilização em inversores de diversos outros fornecedores. Na figura 5 pode-se observar uma foto do inversor no laboratório.

Figura 5 – Inversor



Fonte: Autoria própria

2.2 Protocolo Sunspec ModBus

De acordo com a própria instituição, a SunSpec Alliance ([s.d.]) é uma aliança sem fins lucrativos que congrega mais de 180 grandes empresas em todo o mundo com o propósito de promover o avanço dos padrões de Recursos Energéticos Distribuídos (*Distributed Energy Resource* [DER], em inglês) por meio de diferentes

frentes, tais como interoperabilidade, cibersegurança e resiliência da rede de energia. Os padrões estabelecidos pela SunSpec têm como objetivo principal simplificar a integração e operação de sistemas DER em escala residencial, comercial e utilitária, em torno de redes elétricas inteligentes. A implementação de tais diretrizes pode resultar em uma redução de custos, promover a inovação e acelerar a transição para uma energia mais limpa e segura no futuro.

Com o propósito de padronizar a comunicação com inversores de múltiplos fabricantes, a Sunspec Alliance desenvolveu o protocolo Sunspec ModBus e estabeleceu um acordo com os fabricantes para disponibilizar esse meio de comunicação de forma padrão em seus produtos. Como resultado, foram desenvolvidas tabelas separadas contendo conjuntos de informações ou comandos possíveis aos inversores. Desta forma, cada fabricante pode criar o seu próprio mapa de tabelas, obtendo informações e comandos personalizados. Um inversor pode possuir diversas tabelas configuradas, iniciando sempre com a tabela de identificação apresentada na tabela 1. No caso do presente trabalho, serão utilizadas tabelas mais antigas, devido ao inversor disponível ser de um modelo anterior à última padronização da DER, que estabeleceu um sistema de tabelas geral para diversos dispositivos, incluindo, mas não só, os inversores, e que criou as tabelas da família 7XX. Deste modo, os novos inversores não necessitam de identificar a tabela seguinte, uma vez que sempre terão todas na mesma sequência.

Tabela 1 – Tabela de identificação ID 1

<i>Start</i>	<i>End</i>	<i>Size</i>	<i>R/W</i>	<i>“Function codes”</i>	<i>Name</i>	<i>Type</i>
40001	40002	2	R	0x03	SID	uint32
40003	40003	1	R	0x03	ID	uint16
40004	40004	1	R	0x03	L	uint16
40005	40020	16	R	0x03	Mn	string
40021	40036	16	R	0x03	Md	string
40037	40044	8	R	0x03	Opt	string
40045	40052	8	R	0x03	Vr	string
40053	40068	16	R	0x03	SN	string
40069	40069	1	R	0x03	DA	uint16

Adaptado de: [https://manuals.fronius.com/\(FRONIUS, \[s.d.\]\)](https://manuals.fronius.com/(FRONIUS, [s.d.]))

O inversor utilizado conta com as tabelas 101, mostrada na tabelas 3, e 111, mostrada na tabela 2, o ultimo digito dessas tabelas informa o numero de fases do inversor mas as tabelas para o inversor monofásico é a mesma que a do trifásico apenas armazenando “NaN” ou “0xFFFF” nas fases não usadas. Também possuindo outras tabelas seguintes que não serão utilizadas nesse trabalho pois tratam principalmente da parte de controle, assim com as tabelas utilizadas já se tem todas as

informações de monitoramento relevantes.

Tabela 2 – Tabela de dados ID 11X

Tabela de dados ID “11X”						
<i>Start</i>	<i>End</i>	<i>Size</i>	<i>R/W</i>	<i>“Function codes”</i>	<i>Name</i>	<i>Type</i>
40070	40070	1	R	0x03	ID	uint16
40071	40071	1	R	0x03	L	uint16
40072	40073	2	R	0x03	A	float32
40074	40075	2	R	0x03	AphA	float32
40076	40077	2	R	0x03	AphB	float32
40078	40079	2	R	0x03	AphC	float32
40080	40081	2	R	0x03	PPVphAB	float32
40082	40083	2	R	0x03	PPVphBC	float32
40084	40085	2	R	0x03	PPVphCA	float32
40086	40087	2	R	0x03	PhVphA	float32
40088	40089	2	R	0x03	PhVphB	float32
40090	40091	2	R	0x03	PhVphC	float32
40092	40093	2	R	0x03	W	float32
40094	40095	2	R	0x03	Hz	float32
40096	40097	2	R	0x03	VA	float32
40098	40099	2	R	0x03	VAr	float32
40100	40101	2	R	0x03	PF	float32
40102	40103	2	R	0x03	WH	float32
40104	40105	2	R	0x03	DCA	float32
40106	40107	2	R	0x03	DCV	float32
40108	40109	2	R	0x03	DCW	float32
40110	40111	2	R	0x03	TmpCab	float32
40112	40113	2	R	0x03	TmpSnk	float32
40114	40115	2	R	0x03	TmpTrns	float32
40116	40117	2	R	0x03	TmpOt	float32
40118	40118	1	R	0x03	St	enum16
40119	40119	1	R	0x03	StVnd	enum16
40120	40121	2	R	0x03	Evt1	bitfield32
40122	40123	2	R	0x03	Evt2	bitfield32
40124	40125	2	R	0x03	EvtVnd1	bitfield32
40126	40127	2	R	0x03	EvtVnd2	bitfield32
40128	40129	2	R	0x03	EvtVnd3	bitfield32
40130	40131	2	R	0x03	EvtVnd4	bitfield32

Adaptado de: [https://manuals.fronius.com/\(FRONIUS, \[s.d.\]\)](https://manuals.fronius.com/(FRONIUS, [s.d.]))

Essas tabelas já permitem uma boa análise da energia gerada e a diferença entre as tabelas 10X e 11X é apenas o tipo de variável para armazenamento das informações. Enquanto a tabela 10X armazena utilizando inteiros com uma variável extra de escalar para cada grupo de informações, a tabela 11X armazena em ponto flutuante. Ademais, o protocolo em questão permite a conexão e o controle de múltiplos inversores de diferentes fabricantes por um único *master*.

Tabela 3 – Tabela de dados ID 10X

<i>Start</i>	<i>End</i>	<i>Size</i>	<i>R/W</i>	<i>“Function codes”</i>	<i>Name</i>	<i>Type</i>
40070	40070	1	R	0x03	ID	uint16
40071	40071	1	R	0x03	L	uint16
40072	40072	1	R	0x03	A	uint16
40073	40073	1	R	0x03	AphA	uint16
40074	40074	1	R	0x03	AphB	uint16
40075	40075	1	R	0x03	AphC	uint16
40076	40076	1	R	0x03	A_SF	sunssf
40077	40077	1	R	0x03	PPVphAB	uint16
40078	40078	1	R	0x03	PPVphBC	uint16
40079	40079	1	R	0x03	PPVphCA	uint16
40080	40080	1	R	0x03	PhVphA	uint16
40081	40081	1	R	0x03	PhVphB	uint16
40082	40082	1	R	0x03	PhVphC	uint16
40083	40083	1	R	0x03	V_SF	sunssf
40084	40084	1	R	0x03	W	int16
40085	40085	1	R	0x03	W_SF	sunssf
40086	40086	1	R	0x03	Hz	uint16
40087	40087	1	R	0x03	Hz_SF	sunssf
40088	40088	1	R	0x03	VA AC	int16
40089	40089	1	R	0x03	VA_SF	sunssf
40090	40090	1	R	0x03	VAr AC	int16
40091	40091	1	R	0x03	VAr_SF	sunssf
40092	40092	1	R	0x03	PF AC	int16
40093	40093	1	R	0x03	PF_SF	sunssf
40094	40095	2	R	0x03	WH AC	acc32
40096	40096	1	R	0x03	WH_SF	sunssf
40097	40097	1	R	0x03	DCA DC	uint16
40098	40098	1	R	0x03	DCA_SF	sunssf
40099	40099	1	R	0x03	DCV DC	uint16
40100	40100	1	R	0x03	DCV_SF	sunssf
40101	40101	1	R	0x03	DCW DC	int16
40102	40102	1	R	0x03	DCW_SF	sunssf
40103	40103	1	R	0x03	TmpCab	int16
40104	40104	1	R	0x03	TmpSnk	int16
40105	40105	1	R	0x03	TmpTrns	int16
40106	40106	1	R	0x03	TmpOt	int16
40107	40107	1	R	0x03	Tmp_SF	sunssf
40108	40108	1	R	0x03	St	enum16
40109	40109	1	R	0x03	StVnd	enum16
40110	40111	2	R	0x03	Evt1	bitfield32
40112	40113	2	R	0x03	Evt2	bitfield32
40114	40115	2	R	0x03	EvtVnd1	bitfield32
40116	40117	2	R	0x03	EvtVnd2	bitfield32
40118	40119	2	R	0x03	EvtVnd3	bitfield32
40120	40121	2	R	0x03	EvtVnd4	bitfield32

Adaptado de: <https://manuals.fronius.com/>(FRONIUS, [s.d.])

Para tanto, a SunSpec empregou o protocolo de comunicação do tipo mestre-escravo Modbus como fundamento. Nesse modelo, o mestre envia um comando e o escravo responde. O ModBus constitui-se na estrutura da mensagem que contém o endereço do escravo, a função utilizada, os dados e o algoritmo de checagem, nessa ordem. Na mensagem de requisição, os dados incluem o endereço do registrador inicial, a quantidade de registradores (caso seja um modo de múltiplos registradores) e, quando é uma função de escrita, os valores a serem escritos. Em resposta, replicam-se o endereço e a função, e os dados dependem do modo. No caso de leitura, informam-se apenas a quantidade de registradores lidos e os valores. No caso de escrita única, informa-se o endereço e o novo valor. No caso de múltiplas escritas, informam-se apenas o endereço inicial e o número de registradores escritos. (Witte Software, [s.d.])

Essas três funções, descritas como exemplo, são as suportadas pelo inversor, e seus códigos são:

- 03 (0x03) leitura de registradores
- 06 (0x06) escrita de um registrador
- 16 (0x10) escrita de múltiplos registradores

O protocolo SunSpec ModBus, apresenta duas variações: *Transmission Control Protocol* [TCP] e *Remote Terminal Unit* [RTU]. Neste trabalho, será utilizada a variação RTU, uma vez que o módulo Wi-Fi (necessário para o funcionamento da variação TCP) estará ocupado com a conexão à rede, que utiliza transmissão serial assíncrona via RS-485. Nesse caso, o sistema opera com apenas um mestre (o ESP32-C6) e pode ter um ou mais escravos. Neste estudo, será utilizado apenas um elemento escravo, que é o inversor.

No estudo em questão, as tabelas utilizadas terão apenas necessidade da função 03 de leitura dos registradores. A função 6 ou 16 de escrita, disponíveis em outras tabelas do mesmo mapa são úteis para a resolução de alguns problemas citados, como controlar os inversores automaticamente, requerem um salto de complexidade que não seria viável no tempo de execução deste trabalho.

Com o propósito de garantir o envio preciso das informações, este protocolo emprega o CRC16 para verificar se a mensagem transmitida foi recebida sem incorreções. O CRC16 executa uma operação aritmética em cada elemento da mensagem e o resultado dessa operação, ocupando 16 bits, é concatenado ao final da mensagem.

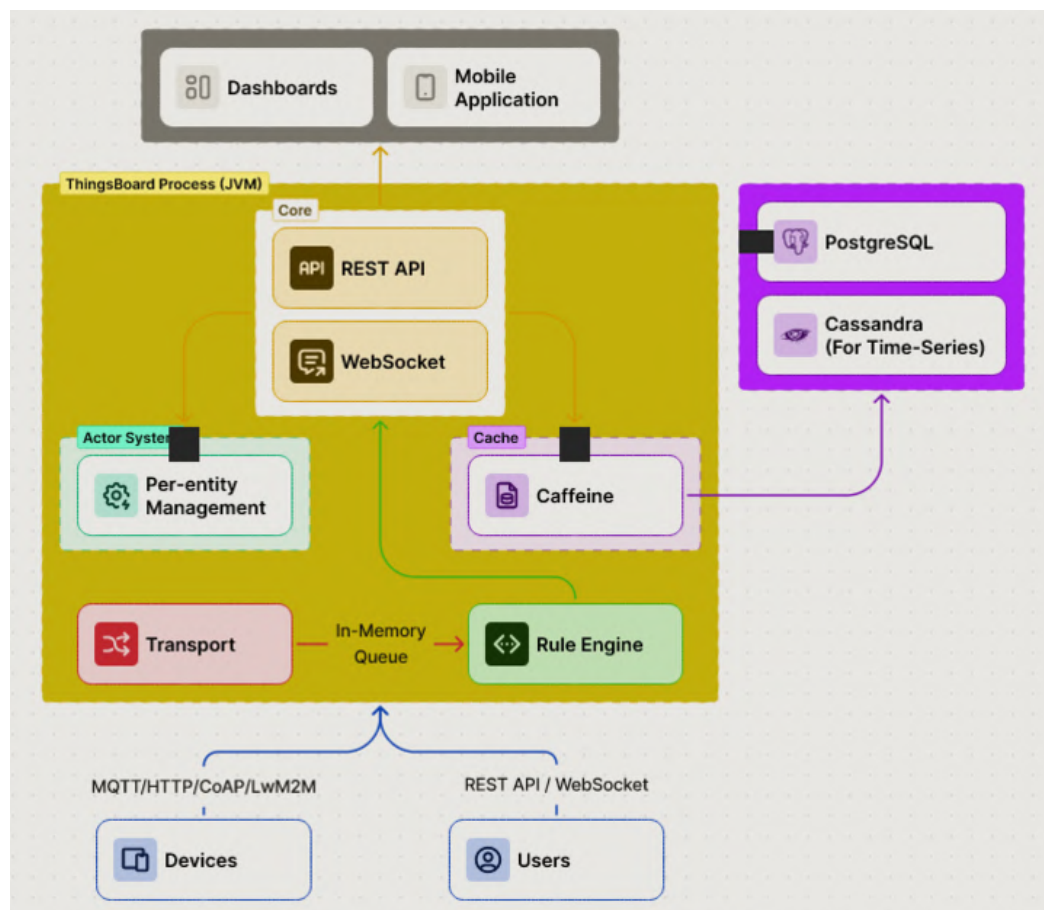
Em caso de erro durante o processamento do comando, uma mensagem de erro é emitida como resposta. Nesse caso, o bit mais significativo da função é estabelecido como 1 (equivalente a 0x80 na codificação). Posteriormente, um código de

exceção é apresentado, detalhando a razão do erro. Dentre os fatores que podem ocasionar tais equívocos, destacam-se a falha do inversor ao processar o comando, a falha física no envio da mensagem, que pode ser identificada por meio da verificação do CRC16, o comando ser inválido por pedir registradores que não podem ser acessados, ultrapassar o limite de 125 registradores por comando ou o código da função não ser uma das três suportadas. (FRONIUS, [s.d.])

2.3 Servidor

Com o objetivo de ter um servidor de forma mais prática foi definido que o ThingsBoard será utilizado nesse trabalho por já ter sido usado em provas de conceito anteriores no laboratório e ter uma arquitetura de fácil entendimento, sendo detalhada na figura 6 que mostra um diagrama de blocos explicando como é processada as informações no servidor, os dispositivos (*devices*) se comunicam via MQTT, HTTP, CoAP ou LwM2M com o servidor então essa comunicação é processada pelo motor de regras antes de ir para o núcleo do servidor que armazena as informações em um servidor PostgreSQL e converte os dados para os *dashboards* configurados que podem ser acessados pelos usuários via WebSocket ou REST API.

Figura 6 – Arquitetura ThingsBoard



Fonte: ThingsBoard ([s.d.])a

Assim o ThingsBoard constitui-se em uma plataforma *OpenSource*, que possibilita a aquisição, o processamento, a visualização e o controle de dispositivos, de modo a facilitar a elaboração do servidor necessário para o funcionamento do trabalho em questão.

Dentre os quatro meios de comunicação citados o MQTT será usado nesse trabalho pois trata-se de um protocolo de mensagem leve *broker-based* no modo *publish/subscribe*, no qual há a utilização de um agente central (*broker*) para o controle de agentes que realizam *subscribe*, isto é, a leitura de um tópico, bem como agentes que executam *publish*, ou seja, a publicação em um tópico.

Segundo Yuan (2021) o protocolo em questão é caracterizado por ser de código aberto e simples, o que o torna ideal para circunstâncias que envolvam limitação de rede ou dispositivos com reduzida capacidade de processamento. A base para a conexão com a rede é constituída pelo protocolo *TCP/Internet Protocol* [IP]. (YUAN, 2021)

Para o desenvolvimento de funções nessa plataforma, pode-se utilizar o linguagem própria da plataforma, denominada *ThingsBoard expression language* [TBEL], que foi otimizada para tal finalidade. Outra possibilidade é utilizar o JavaScript, linguagem de programação amplamente empregada em páginas da web e que serviu de base para o desenvolvimento do TBEL. Dessa maneira, uma vez que ambos são fundamentados em JavaScript, para estruturar as mensagens será utilizado o *JavaScript Object Notation* [JSON] que é uma formatação leve para troca de dados que possui suporte em diversas linguagens de programação assim podendo ser utilizado o mesmo formato no servidor e no microcontrolador que faz a aquisição dos dados.

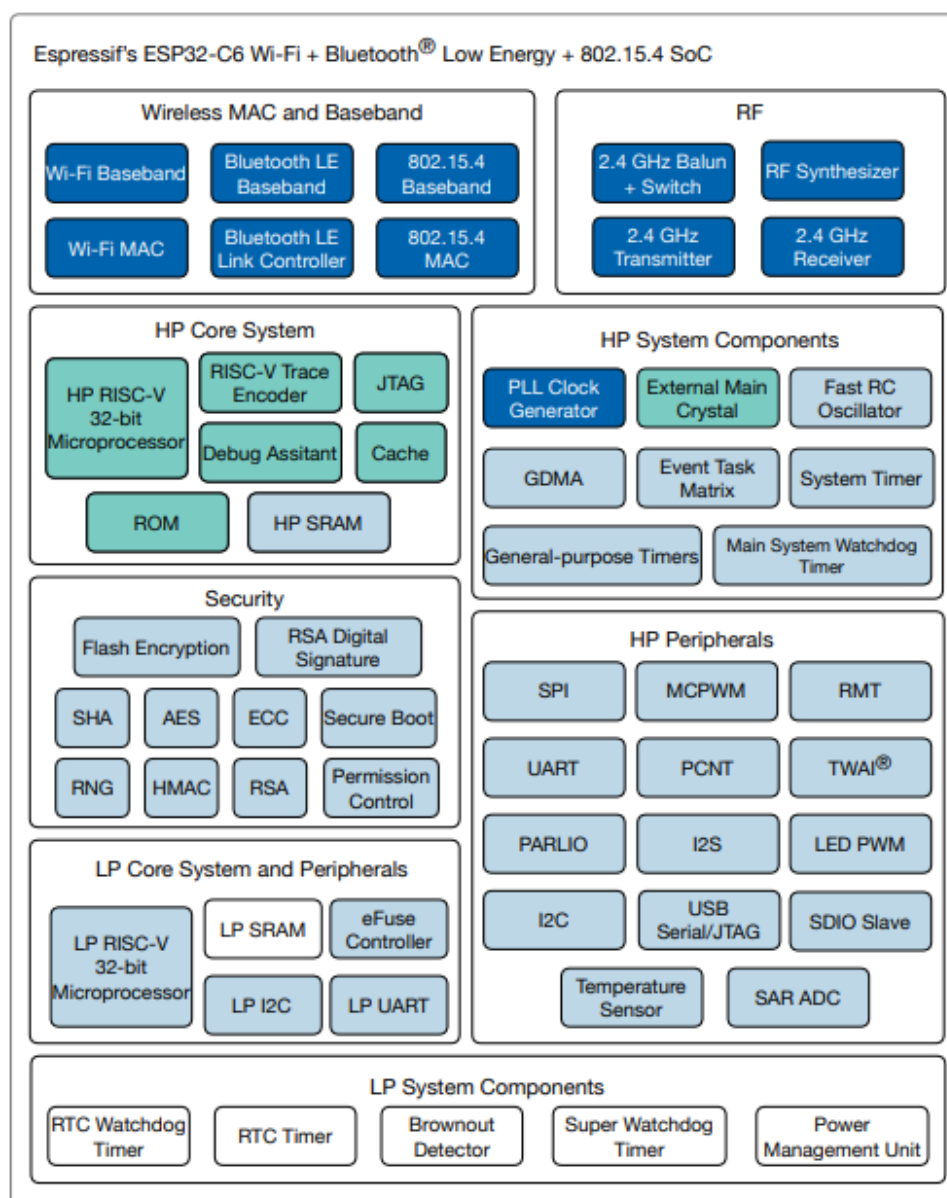
Para utilizar o ThingsBoard, é possível empregar um servidor em nuvem disponibilizado pelo próprio sistema ou executar um servidor em máquina local. Neste trabalho, a melhor opção é utilizar o servidor em máquina local, a fim de facilitar a comunicação, dado que a rede interna da instituição apresenta limitações em razão da segurança quando se trata de utilizar servidores externos.

2.4 Microcontrolador

O microcontrolador ESP32-C6 foi selecionado devido ao seu suporte à arquitetura RISC-V e ao Wi-Fi 6. O Wi-Fi se torna útil devido à possibilidade de comunicação futura via TCP e à eliminação da necessidade de ponto de rede em instalações. O Wi-Fi 6 permite maior estabilidade e mais dispositivos conectados simultaneamente, o que é útil considerando que os inversores podem ser instalados em locais que não necessariamente tem uma boa cobertura de Wi-Fi e estarão conectados à rede da instituição, que é usada por uma grande quantidade de pessoas. Assim, pode-se operar com uma conexão mais limitada, se necessário. Ademais, a arquitetura RISC-V conta

com uma extensa comunidade, o que resulta na disponibilidade de diversas bibliotecas e documentação mais extensa. Na figura 7 é mostrado um diagrama contendo todos os blocos dentro do microcontrolador e seus periféricos, nesse projeto serão utilizados além dos blocos essenciais os blocos do Wi-Fi, UART, USB(serial/JTAG). A seguir, serão elencadas as razões pelas quais essas tecnologias apresentam tais vantagens.

Figura 7 – Arquitetura ESP32-C6

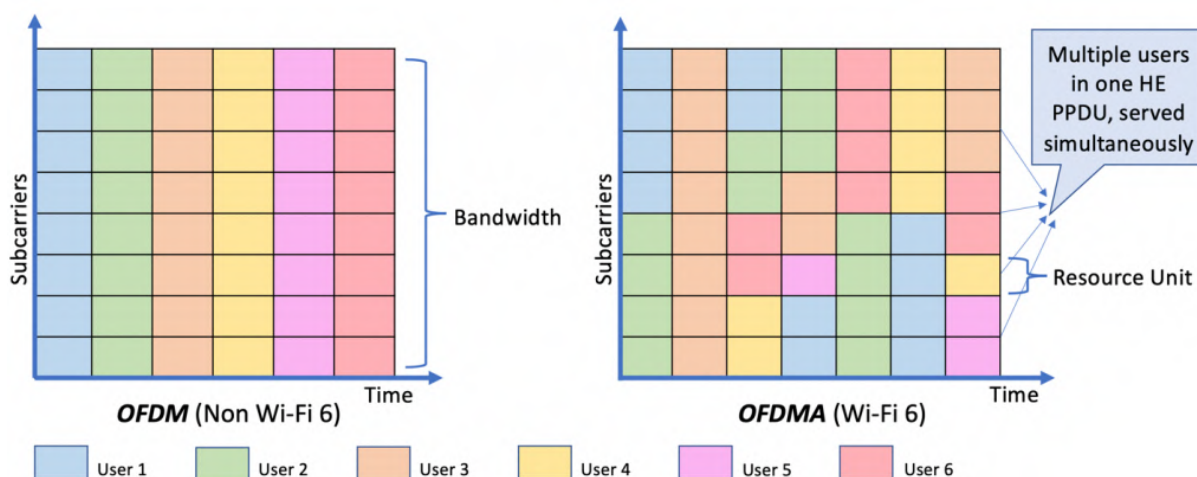


Fonte: Espressif (2023a, p.2)

O Wi-Fi 6, também conhecido como Wi-Fi 802.11ax, constitui um avanço significativo na tecnologia de conexão sem fio. Esse avanço permite uma série de melhorias, dentre as quais se destacam maior estabilidade na transmissão de dados, a possibilidade de conexão simultânea de mais dispositivos, maior velocidade e melhor aproveitamento energético. A melhoria na estabilidade e na conexão simultânea de

dispositivos é consequência do acesso múltiplo por divisão de frequência ortogonal (OFDMA), ilustrado na figura 8. Nesse modelo, os canais são subdivididos, permitindo que múltiplos dispositivos operem de modo mais eficiente em conjunto. A tecnologia emprega conjuntos de serviços básicos sobrepostos (OBSS), que possibilitam filtrar os ruídos na transmissão de forma mais eficiente. Essas tecnologias, em conjunto, auxiliam no aumento da velocidade total de transmissão, já que cada dispositivo alcança uma conexão melhor. A inovação que contribui para a eficiência energética é o *Target Wake Time* (TWT), que otimiza a velocidade de ativação e desativação da comunicação. Dessa forma, a antena de rádio, operando por um período reduzido, resulta em economia de energia. (Intel, [202-])

Figura 8 – OFDMA



Fonte: Trivedi (2020)

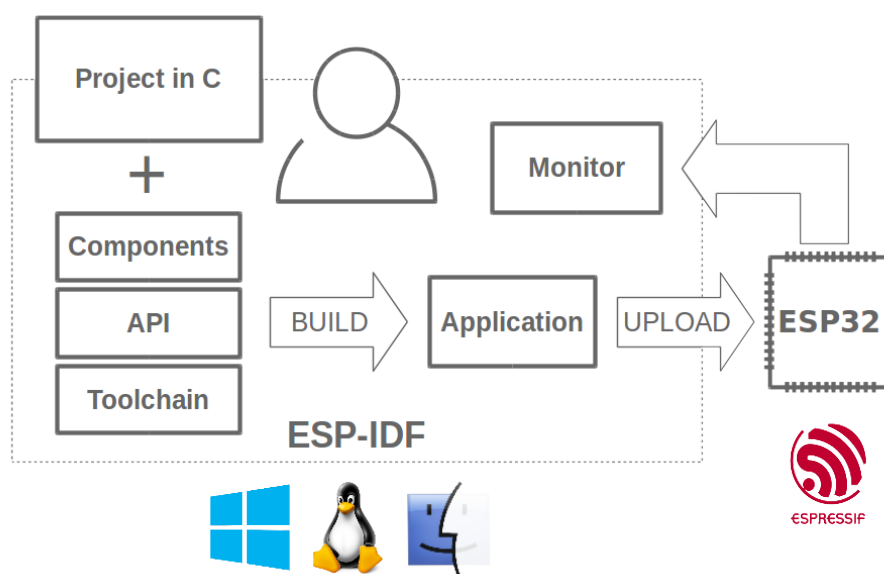
RISC-V constitui uma arquitetura de conjunto de instruções universal (*Instruction Set Architecture* [ISA]), cujas características incluem a capacidade de atender a diferentes tamanhos de processadores, funcionar com uma ampla variedade de softwares e linguagens de programação, ser eficiente para todos os tipos de microarquitetura, oferecer suporte a uma ampla especialização para atuar como base para aceleradores personalizados e ser estável, ou seja, a ISA base não deve mudar. O RISC-V apresenta diferenças em relação às demais ISAs, pois é um sistema mais recente e pertence a uma fundação aberta e sem fins lucrativos. Dessa forma, torna-se muito mais difícil de ser descontinuado, pois permite livre acesso às informações necessárias para criar o RISC-V de forma independente. Além disso, o RISC-V é modular, ou seja, a base nunca muda, o que o torna mais estável. As inovações são acrescentadas como módulos que permitem expandir a capacidade de processamento de formas específicas. Dessa maneira, caso haja a necessidade de funções mais avançadas, não é preciso incluir todas as outras funções avançadas, como em outras ISAs. (David Patterson; Andrew Waterman, 2017)

Diante do microcontrolador apresentar as características desejadas, além de ser de fácil acesso e dispor de uma ampla gama de exemplos e bibliotecas disponibilizadas pela sua fabricante, a Espressif, foi definido como o microcontrolador utilizado para elaboração deste trabalho.

2.5 ESP-IDF

Para utilizar o microcontrolador é necessário a utilização de uma *framework* para converter os códigos criados para linguagem de máquina utilizada no processador, para tal será utilizado o *framework* oficial da fabricante do microcontrolador, a Espressif (2020), denominado Espressif *Internet of things Development Framework* (ESP-IDF), que permite o desenvolvimento de qualquer aplicação genérica utilizando linguagem de programação C/C++, e seu esquema de funcionamento pode ser visualizado na figura 9. (ESPRESSIF, 2020)

Figura 9 – Funcionamento IDF



Fonte: Espressif ([s.d.])

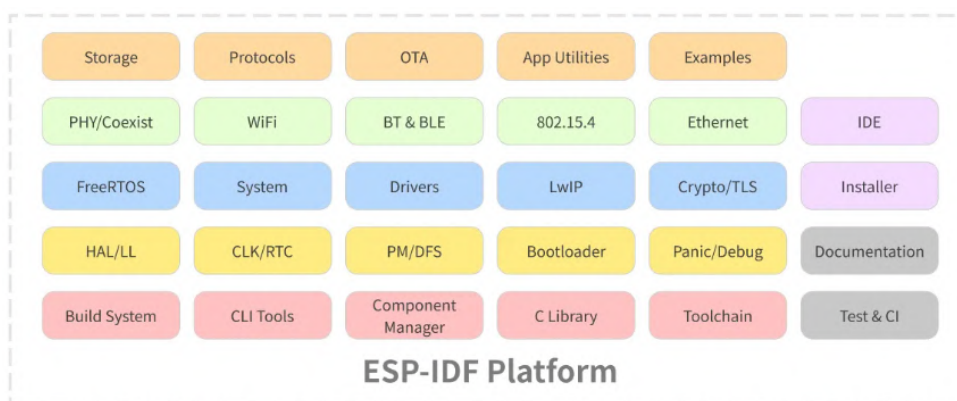
Segundo a Lee (2025) todos os blocos necessários para criação de uma aplicação são incluídos na IDF. Assim as principais bibliotecas utilizadas no projeto são:

- FreeRTOS, que possibilita o *Real-Time Operating System* [RTOS] que é o sistema operacional em tempo real, sendo um sistema leve que permitirá processos em multi-tarefas nesse caso permitindo que os dados sejam lidos em sequencia sem a necessidade do envio ter sido finalizado, podendo finalizar durante o processo de leitura ou conversão do dado;

- b) Drivers que permitem acessar os periféricos do microcontrolador como a UART e o USB;
- c) Protocolos que implementam as funções para utilização de por exemplo o MQTT e o próprio ModBus.

O ESP-IDF também é disponível gratuitamente e além de contar com essas e outras bibliotecas também dispões de exemplos que serão utilizados no decorrer desse trabalho. Na imagem 10 ilustra os blocos que compõem a *framework*.

Figura 10 – Componentes presentes na IDF



Fonte: Lee (2025)

3 METODOLOGIA

Este trabalho foi realizado de forma experimental, com o objetivo de criar um protótipo funcional que pudesse adquirir os dados do inversor e transmiti-los via Wi-Fi para um servidor. O presente capítulo, portanto, abordará a sequência de desenvolvimento utilizada para alcançar esse objetivo. Primeiramente, foi necessário compreender o protocolo e realizar testes via simulação e com um kit de desenvolvimento que possuísse o microcontrolador escolhido. Posteriormente, foi explicado o processo de desenvolvimento do algoritmo de aquisição e os problemas enfrentados para fazer com que o algoritmo funcionasse. Em seguida, foi descrita a configuração do servidor, bem como a criação dos painéis de controle para compreensão dos dados adquiridos. Então, foi estabelecida a conexão com o inversor e o algoritmo foi readaptado de acordo com novas informações descobertas durante a conexão. Por fim, foi desenvolvido o hardware, assim como o esquemático foi montado e os detalhes de roteamento do layout foram explicados. Finalizada a elaboração da placa, esta foi submetida a testes e adaptações com o intuito de aprimorá-la e garantir um funcionamento otimizado.

3.1 ModBus

Para otimizar a compreensão acerca do protocolo e da comunicação com o ESP32-C6, foi realizada a utilização de simuladores tanto de *slave* (inversor) quanto de *master* (ESP32-C6). Os simuladores utilizados foram:

- O software qModMaster foi empregado para simular o ESP32-C6 de maneira mais simplificada, com o objetivo de compreender o protocolo antes de adentrar nos aspectos específicos da programação.
- O simulador em Python desenvolvido pela SunSpec Alliance e adaptado para python 3 simula um escravo previamente mapeado em SunSpec ModBus. No entanto, carece de interface gráfica para auxiliar na compreensão e na configuração, além de por se tratar de um código mais antigo, este apresenta alguns problemas de sincronização na conexão física dificultando a comunicação com o ESP32-C6. Todavia, mostra-se útil para uma primeira análise com uma conexão por portas virtuais entre simuladores.
- O software OModSim permite a simulação de um escravo, apesar de não estar mapeado e possuir um limite de, no máximo, 200 registradores configurados. A plataforma gráfica do software auxilia

na conferência de informações e permite configurações que serão úteis ao longo do trabalho.

Assim foi definido uma sequência de testes para entender o funcionamento do ModBus e dos seus simuladores que começa utilizando o simulador de mestre, o software qModMaster para se comunicar com os simuladores de escravo, o simulador em Python e o software OModSim, respectivamente. Assim, primeiro entendendo o funcionamento com o simulador mapeado em uma conexão virtual e depois no OModSim ampliando com uma conexão física. Após o entendimento dos simuladores será testado com o ESP32-C6 até estabelecer conexão com pelo menos um dos simuladores e a partir dessa conexão será iniciado o desenvolvimento do algoritmo de aquisição de dados e envio para um servidor, bem como a configuração do servidor.

3.1.1 Comunicação entre qModMaster e o Simulador em Python

Com o propósito de entender o funcionamento do protocolo foi utilizado primeiramente um sistema com os dois simuladores, o software qModMaster e o simulador em Python para operar como o dispositivo de monitoramento e como inversor, respectivamente.

Para que a comunicação serial fosse viável, foi preciso gerar duas portas virtuais interligadas. Para criar e conectar essas portas virtuais foi utilizado o a ferramenta socat, que permite gerar conexões bidirecionais para transferência de dados por rede ou de forma serial. Neste caso, foram utilizadas as portas seriais para simular uma conexão física entre os simuladores pois dessa forma os atrasos gerados pela conexão física são atenuados. Assim, tendo uma conexão mais estável para um primeiro contato com o protocolo na prática. Para que fosse possível utilizar as portas virtuais, foi necessário realizar algumas alterações no qModMaster. Isso porque o programa utilizava um filtro de nomenclatura que buscava apenas portas que comesçassem com “tty”, “USB” ou “ACM”, enquanto a porta virtual gerada iniciava com “pts”. Após modificar o filtro o programa reconheceu as portas virtuais normalmente. Conforme demonstrado na Figura 12, o simulador qModMaster é apresentado após a leitura dos 100 primeiros valores em Python, iniciando a partir do endereço 40000, que corresponde ao endereço inicial para o protocolo SunSpec.

Conforme demonstrado na Figura 11 é possível observar o comando utilizado para iniciar o simulador no terminal, utilizando a porta virtual e o mapa já configurado, bem como as informações fornecidas durante a inicialização do processo, como a abertura da porta de comunicação definida na inicialização, “/dev/pts/3”, a taxa de transmissão, por padrão 9600, o tipo de paridade na comunicação, por padrão N, o id do escravo/inversor, por padrão 1, o endereço base para inicio da tabela, por padrão

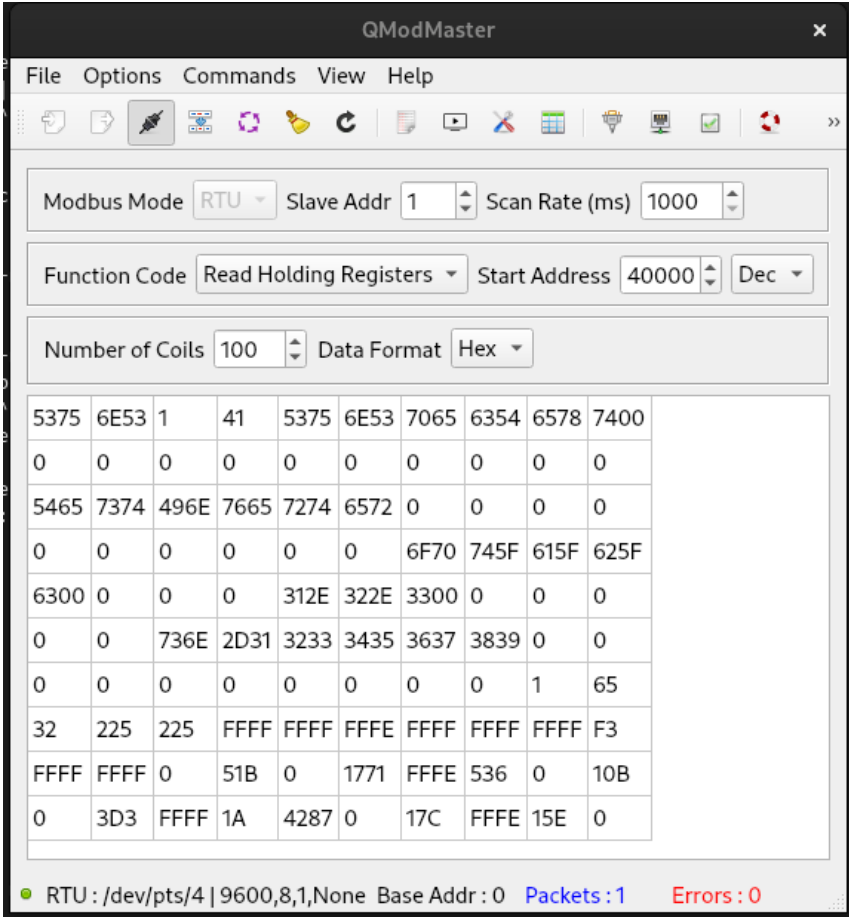
40000, bem como quando terminou de carregar o mapa e finaliza informando que está em operação.

Figura 11 – Simulador

```
[joao@toolbox modsim]$ python3 modsim.py -s /dev/pts/3 mbmap_test_device.xml
[<mbmap.ModbusMapRegs object at 0x7f6e1a407e90>]
RtuServer alt /dev/pts/3 is opened
to = 0.00171875 0.011458333333333334
Initialized modbus rtu simulator: baud = 9600 parity = N slave id = 1 base address = 40000
Modbus map loaded from mbmap_test_device.xml
Added modbus map block: address = 40000 count = 991
'quit' for closing the simulator
modbus_tk.simulator is running...
█
```

Fonte: Autoria própria

Figura 12 – qModMaster



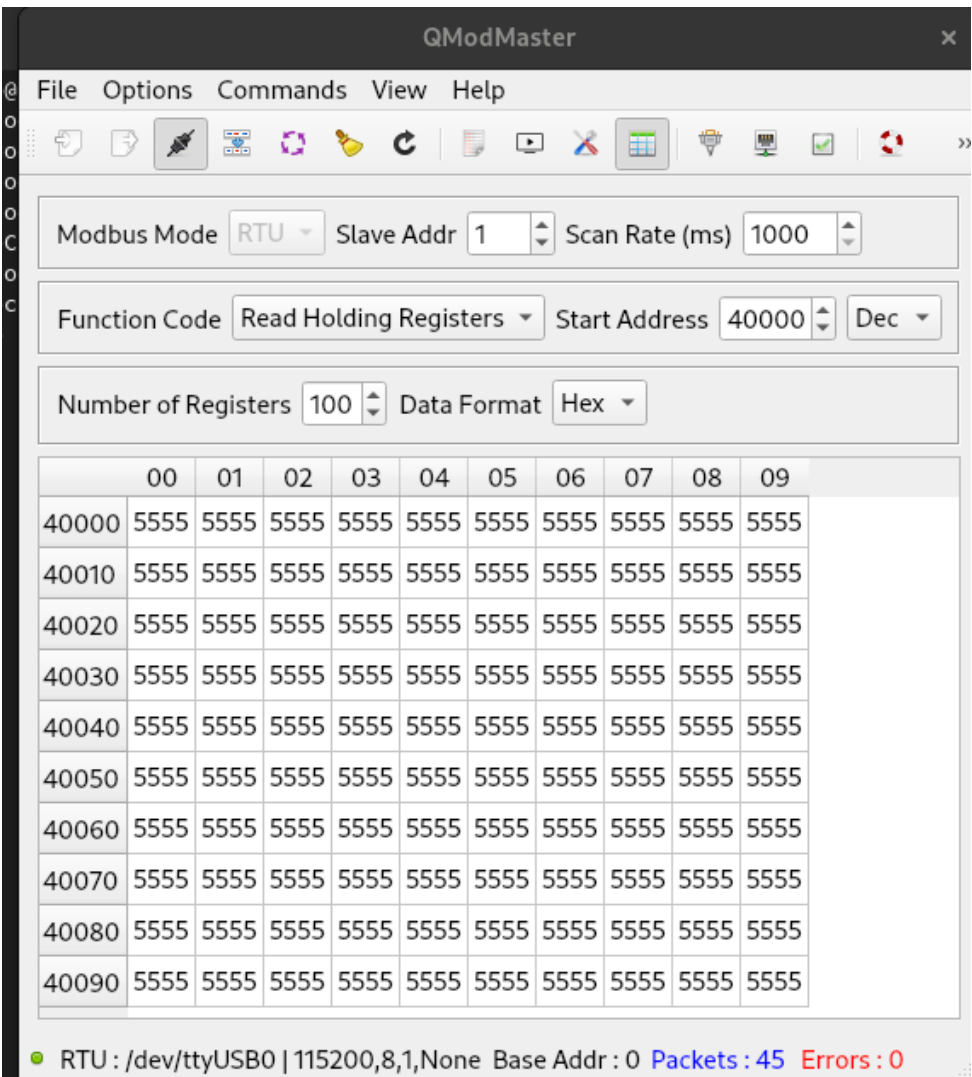
Fonte: Autoria própria

Com esses testes foi possível entender a estrutura da tabela mapeada no simulador assim servindo como primeiro contato de forma prática com o protocolo.

ambiente de testes mais adequado para otimizar a aplicação antes de conectá-la ao inverso. Procedeu-se à realização de ensaios com o intuito de compreender seu funcionamento e empregar a conexão por portas USB físicas.

Dessa forma, o QModMaster recebeu os dados de maneira adequada, conforme demonstrado na Figura 14. No caso do teste realizado, os registradores do OModSim foram zerados. Por consequência, na imagem todos os 100 registradores são lidos como “5555”. No entanto, outros testes foram realizados com os registradores, atribuindo-lhes valores aleatórios, com o objetivo de verificar se o valor “5555” era um caso específico.

Figura 14 – Dados obtidos no qModMaster através do teste

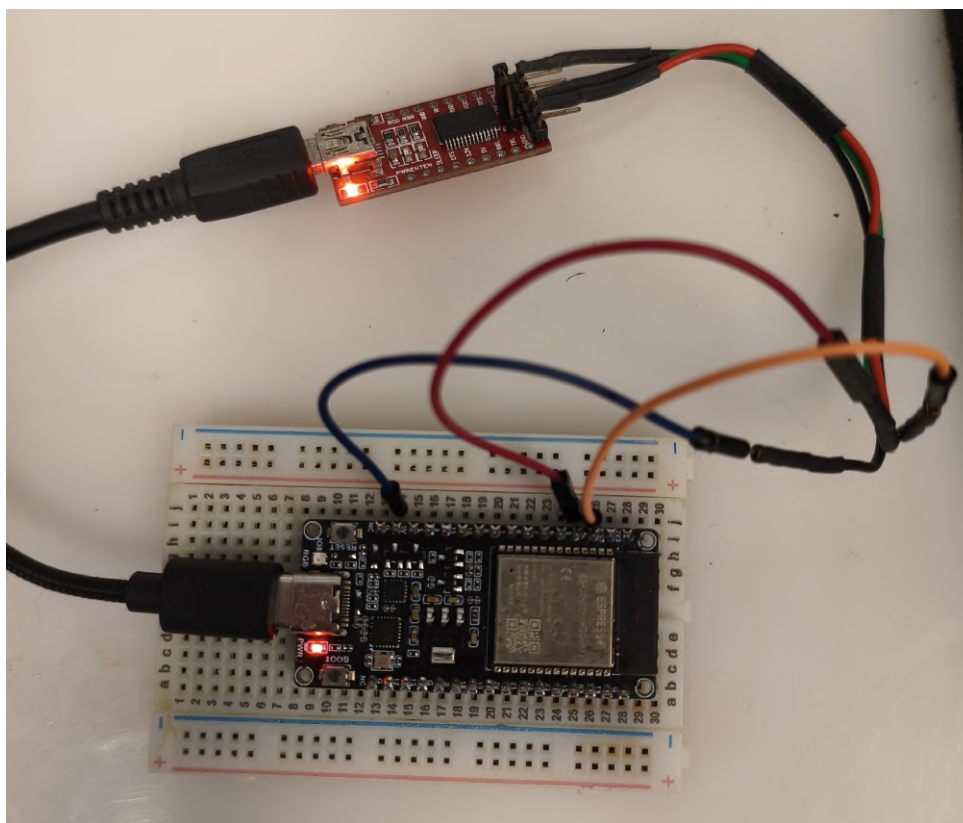


Fonte: Autoria Própria

Para a realização desta comunicação, foi necessário modificar o código exemplo de comunicação UART da Espressif para o ESP32-C6, de modo que ele atue como repetidor USB entre as duas portas USB físicas. A conexão física, ilustrada na Figura 15, foi realizada da seguinte maneira: duas portas USB foram utilizadas, sendo

que uma foi conectada na entrada do próprio ESP-C6 e a outra, em um dispositivo de conversão USB para UART chamado FTDI. Este, por sua vez, foi conectado nas portas UART do ESP32-C6 e no GND.

Figura 15 – Loop USB



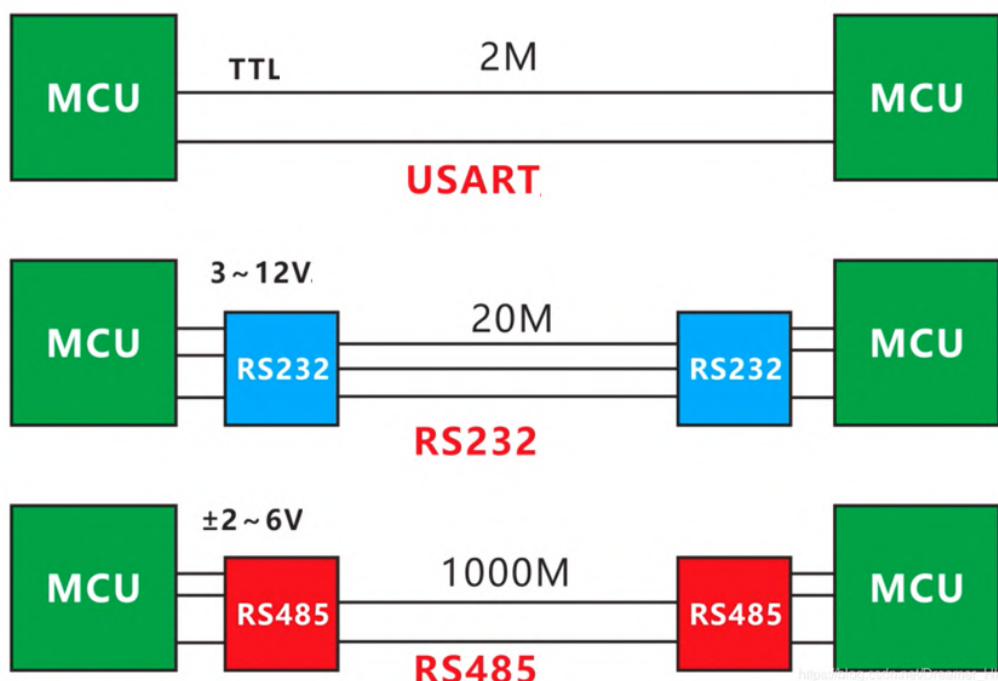
Fonte: Autoria Própria

3.1.3 Comunicação entre o ESP32 e o Simulador em python

Para dar continuidade aos testes agora substituindo o simulador qModMaster pelo microcontrolador para poder começar o desenvolvimento do código necessário para comunicação com o inversor foi utilizado o exemplo de mestre do ModBus, disponibilizado pela Espressif para o ESP32-C6, e um FTDI para comunicação serial com o computador, que estaria executando o simulador. o FTDI foi utilizado nessa etapa pois a comunicação via RS485 opera do mesmo modo para o microcontrolador, apenas com uma camada física a mais que permite maior distancia de comunicação, como mostrado figura 16, que apresenta um diagrama com as comunicações TTL utilizada pelo FTDI.

Foi realizada uma série de testes de comunicação com o simulador, que apresentou como resultado a mensagem de erro de requisição inválida. Após a alteração do código do simulador, foi possível visualizar no terminal como a requisição foi recebida. Foi observado que a requisição parecia correta, com apenas o caractere ":" no início. Com o objetivo de realizar testes, foi tentada a modificação do código para

Figura 16 – Diferenças entre RS485, RS232 e TTL



Fonte: <https://www.jsypowermeter.com/news/differences-between-rs485-ttl-and-rs232/> (JSY-MK LTD., 2024)

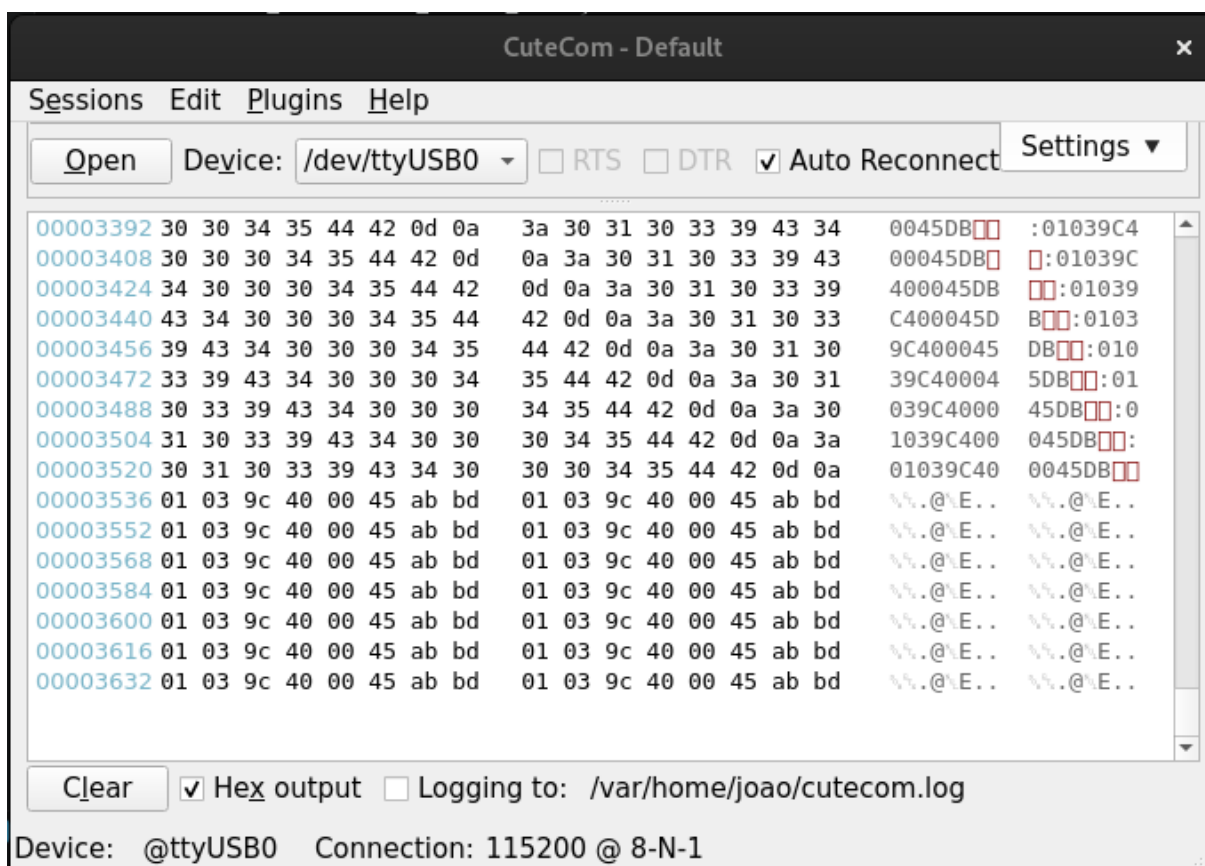
ignorar o primeiro caractere, contudo o erro persistiu. Dessa maneira, os testes foram continuados no outro simulador para verificar se o problema não estava no simulador em Python.

3.1.4 Comunicação entre o ESP32 e o OModSim

Para substituir o simulador em Python foi utilizado o software OModSim que possui uma interface gráfica que permite analisar o *log* com mais facilidade mantendo a mesma utilização do FTDI para comunicação entre o ESP e computador com simulador.

A utilização do OModSim não solucionou o problema, de modo que foi realizado um teste para interceptar o sinal. Utilizando o software CuteCom para ver a comunicação serial, ilustrado na figura 17 onde na direita está a interpretação do sinal hexadecimal para ASCII e na esquerda está o sinal recebido em hexadecimal sem tratamento. Assim foi constatado que o ESP32-C6 estava enviando ":" no início do sinal e "\r\n" no final. Ademais, a mensagem não se encontra em formato hexadecimal, isto é, encontrava-se em modo ASCII. Foi realizada uma busca no código para identificar a configuração para o modo "RTU". Após a configuração, que pode ser visualizada na mesma figura 17 nas ultimas linhas, foi estabelecida a conexão entre o ESP e o simulador.

Figura 17 – Intercepção dos dados



Fonte: Autoria própria

A saída no terminal foi copiada abaixo, onde é possível observar que o *log*, gerado pelo ESP32-C6 no terminal indicando que a característica 3 foi lida com sucesso. Como nesse momento ainda estava em teste estava-se utilizando um ciclo do código exemplo que testa varias requisições de diferentes informações, inclusive em funções distintas, assim as outras características testadas retornam erro. Foi constatado que o simulador disponibiliza apenas um tipo de função por vez. Para a realização dos testes, foi selecionado o tipo “holding register”.

```
E (1210) MASTER_TEST: Characteristic #1 (Humidity_1) read fail, err = 0x107 (ESP_ERR_TIMEOUT).
E (1210) MB_CONTROLLER_MASTER: mbc_master_get_parameter(75): Master get parameter failure, error=(0x108) (ESP_ERR_INVALID_RESPONSE).
E (1210) MASTER_TEST: Characteristic #2 (Temperature_1) read fail, err = 0x108 (ESP_ERR_INVALID_RESPONSE).
I (326) MASTER_TEST: Characteristic #3 Humidity_2 (%rH) value = 0.000000 (0x0x410001) read sucessful.
E (1210) MB_CONTROLLER_MASTER: mbc_master_get_parameter(75): Master get
```

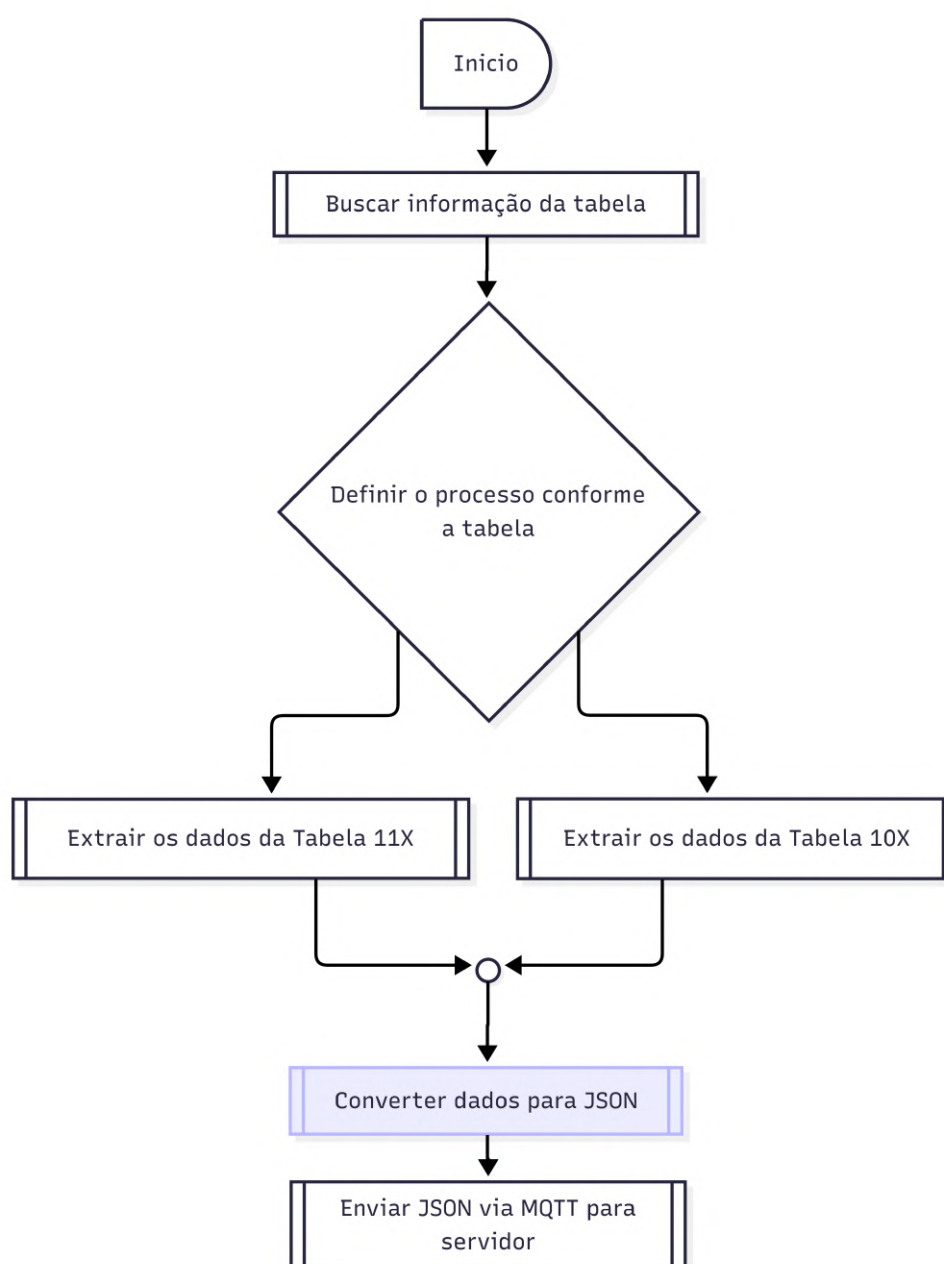
parameter failure, error=(0x107) (ESP_ERR_TIMEOUT).

E (1210) MASTER_TEST: Characteristic #4 (Temperature_2) read fail, err = 0x107 (ESP_ERR_TIMEOUT).

Para prosseguir com a realização dos testes, os valores dos registradores foram inseridos de modo aleatório, com o objetivo de distinguir quais registradores estão sendo lidos. No modo *holding register*, uma vez que a leitura é feita de forma única, não há necessidade de múltiplos envios diferentes.

3.1.5 Desenvolvimento do algoritmo

Figura 18 – Fluxograma de aquisição dos dados



Fonte: Autoria própria

Após a primeira comunicação entre o ESP32-C6 e o simulador ter sido bem-sucedida, foi iniciado o processo de desenvolvimento de um algoritmo capaz de identificar a tabela que será lida, ler a tabela e converter os dados de modo que o servidor consiga tratá-los de forma simples. Na figura 18, encontra-se um fluxograma que ilustra a lógica empregada nesse algoritmo.

A criação do algoritmo foi precedida por uma adaptação do código, com o objetivo de reduzir o número de requisições. Isso foi necessário porque o protocolo em questão permite a captura de diversos registradores, enquanto o código original estava configurado para obter os valores individualmente. Diante do exposto, foi realizada uma tentativa de modificação do “regsize” para requisitar múltiplos registradores. A resposta obtida pelo simulador foi equivalente àquela que seria esperada quando apenas um registrador fosse requisitado.

Com o uso do aplicativo CuteCom para a investigação do sinal, foi observado que a modificação no “regsize” alterou o sinal de saída de acordo com o esperado para adquirir múltiplos registradores. Dessa forma, o problema está no momento em que o ESP32-C6 lê essa resposta.

Ao escrever na tela o ponteiro como *string* de resposta inteiro, percebe-se que ele mantém o mesmo de quando adquiria apenas dois registradores. Após a análise dos parâmetros, foi possível identificar o parâmetro denominado “datasize”. Ao aumentá-lo, o ponteiro acompanha o incremento. Para efetivamente realizar a leitura, o ponteiro é equipado com um laço que percorre o ponteiro e escreve na tela como hexadecimal, preenchendo com zero. Dessa forma, o algoritmo é capaz de ler múltiplos registradores com uma única requisição. O *log* gerado pelo ESP32-C6 no terminal foi copiado abaixo, com a informação de que o dado foi lido com sucesso e, em seguida, os dados lidos são apresentados em formato hexadecimal.

```
I (326) MASTER_TEST: Modbus master stack initialized...
I (326) MASTER_TEST: [APP] Startup..
I (326) MASTER_TEST: [APP] Free memory: 43110 bytes
I (326) MASTER_TEST: [APP] IDF version: v5.5.1-dirty
I (326) MASTER_TEST: Start modbus test...
I (326) MASTER_TEST: Characteristic #0 ALL (0x) value = -0.000062
(0xb882a716)1 read sucessful.
DATA:
0xA716 0xB882 0x79FE 0x769C 0xC30C 0x2C64 0xFCA4 0x02E3 0xE79E 0x3D43
0x3560 0x211E 0xED9A 0x6A27 0x7E58 0x2B33 0xF3D1 0xB229 0x45A9 0x15FD
0x8521 0xF683 0xB7D8 0x54DF 0xCC7A 0xE183 0x6881 0x3B11 0xFF07 0x56A0
0x505C 0x7335 0xAAA5 0x8AC8 0xD9D1 0xBEF6 0xEE7D 0xCABE 0x0371 0x2D1D
```

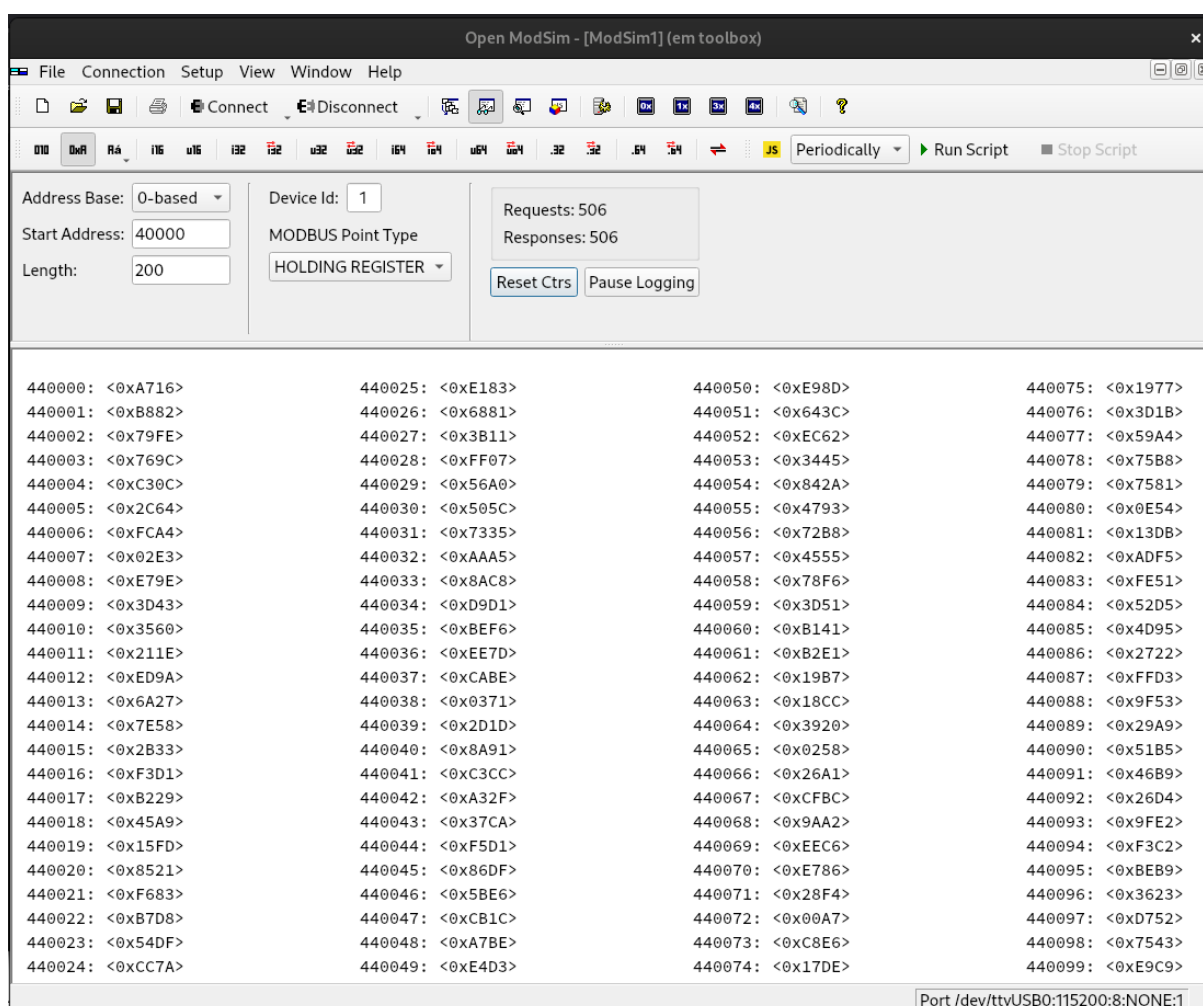
```

0x8A91 0xC3CC 0xA32F 0x37CA 0xF5D1 0x86DF 0x5BE6 0xCB1C 0xA7BE 0xE4D3
0xE98D 0x643C 0xEC62 0x3445 0x842A 0x4793 0x72B8 0x4555 0x78F6 0x3D51
0xB141 0xB2E1 0x19B7 0x18CC 0x3920 0x0258 0x26A1 0xCFBC 0x9AA2 0xEEC6
0xE786 0x28F4 0x00A7 0xC8E6 0x17DE 0x1977 0x3D1B 0x59A4 0x75B8 0x7581
0x0E54 0x13DB 0xADF5 0xFE51 0x52D5 0x4D95 0x2722 0xFFD3 0x9F53 0x29A9
0x51B5 0x46B9 0x26D4 0x9FE2 0xF3C2 0xBEB9 0x3623 0xD752 0x7543 0xE9C9
I (916) MASTER_TEST: Alarm triggered by cid #0.
I (916) MASTER_TEST: Destroy master...
I (916) main_task: Returned from app_main()

```

Para a coleta de dados, foi utilizada a função de definir valores aleatórios para todos os registradores no simulador OModSim. Dessa forma, foi possível verificar se todos os registradores estavam sendo lidos na ordem correta. A visualização do OModSim e a informação de seus registradores podem ser observadas na figura 19.

Figura 19 – OModSim com os registradores aleatórios



Fonte: Autoria própria

Posteriormente, foi imperativo desenvolver um código para processar os da-

dos recebidos em formato hexadecimal e convertê-los em um texto que seria enviado ao servidor. A fim de ilustrar o algoritmo empregado nessa função, consulte a figura 20. Para compreender os detalhes do laço que percorre o dado hexadecimal, consulte o fluxograma apresentado na figura 21. Para que o dado seja processado pelo servidor, o texto gerado foi feito no formato de mensagem suportado nativamente, o JSON.

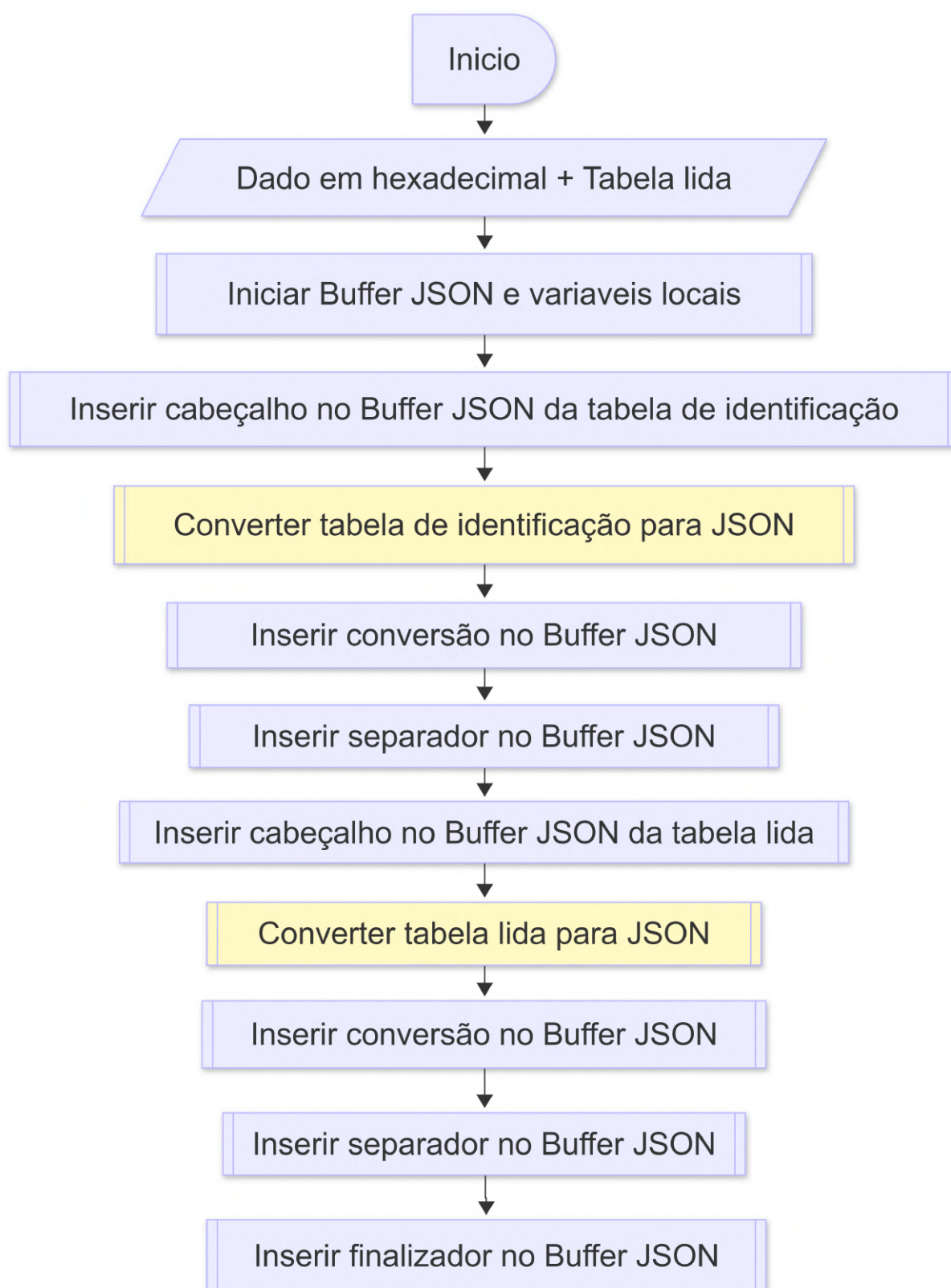
Dessa forma, com o propósito de identificar como separar o dado de acordo com cada linha da tabela, foi utilizado um código em Python para transformar as tabelas em Excel. Inicialmente, as informações foram divididas em dois vetores constantes em C. O primeiro vetor estabelece a quantidade de registradores a serem lidos, bem como o tipo e o tamanho de cada seção do dado, enquanto o segundo vetor identifica a etiqueta de cada dado.

Para dar continuidade ao processo de criação da função que transformaria em texto e, assim, prosseguir com a etapa seguinte em C, é necessário determinar a quantidade de memória a ser utilizada. Para poder testar a conversão, primeiro foi empregado um método manual de identificação, no qual se supôs quanto cada tipo ocuparia na memória como texto. Posteriormente, foi realizado um laço no vetor de tipos, somando o espaço ocupado por cada dado e o tamanho máximo da etiqueta.

Posteriormente, a variável foi alocada para *string* e executado outro laço para montar a *string* usando “snprintf”. Após a implementação de alguns ajustes, o processo foi bem-sucedido, resultando em um texto coerente com todos os dados convertidos corretamente. Contudo, supor o tamanho de memória ocupado por cada tipo constitui um risco, visto que um cálculo impreciso pode resultar na corrupção da memória do microcontrolador. Diante do exposto, é imprescindível o desenvolvimento de um processo para a verificação precisa do tamanho.

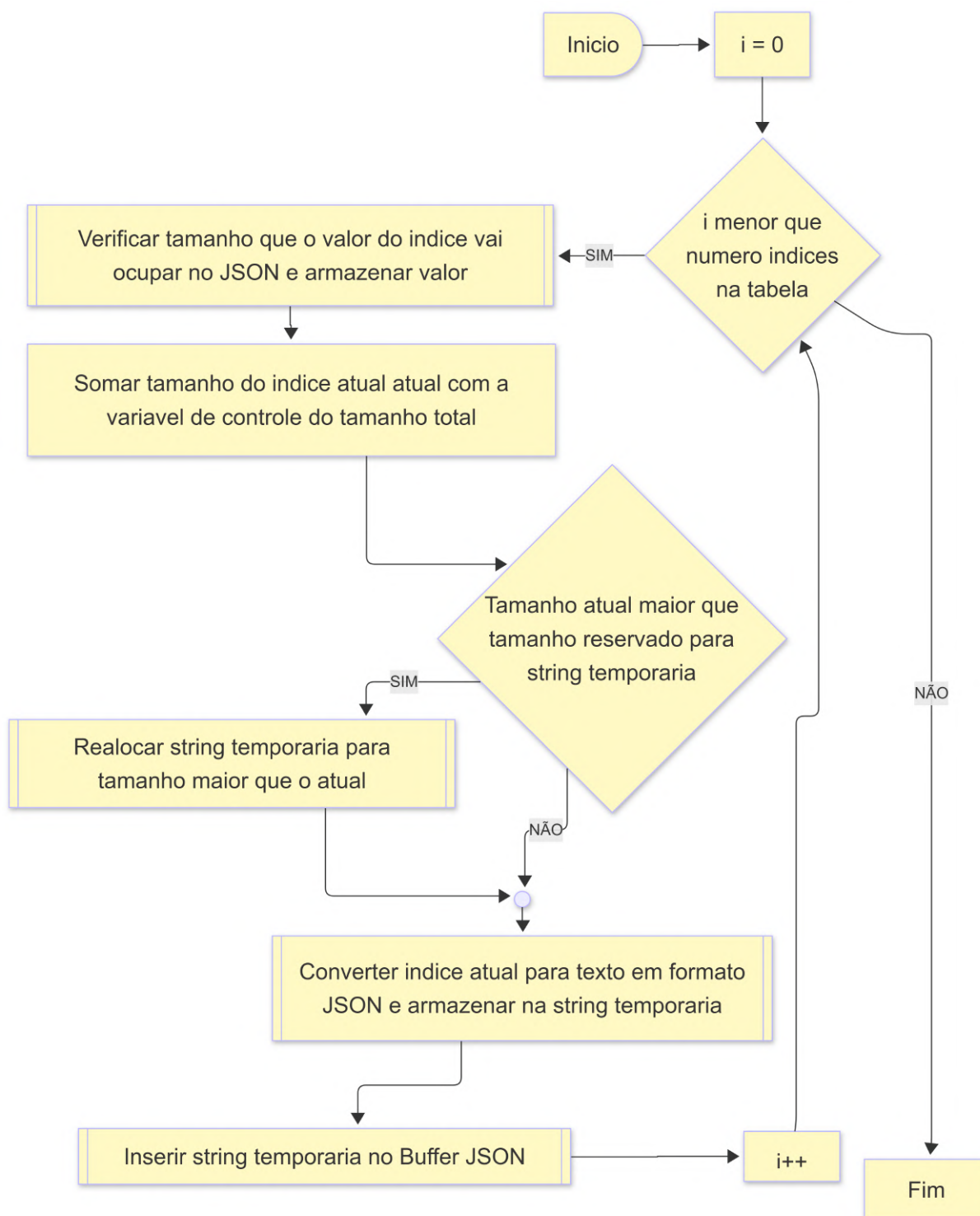
Para identificar o espaço de memória necessário para alocá-lo, foi utilizada a própria função “snprintf”, que, mesmo quando não há espaço suficiente, retorna o tamanho necessário. Dessa forma, primeiramente foi necessário executar um ciclo com uma variável com apenas um espaço alocado, somar o retorno de cada função e, posteriormente, outro ciclo após a realocação da memória com a soma dos valores retornados. Dessa forma, a *string* do dado tratado foi gerada. No entanto, logo após a apresentação da *string*, foi observado um erro na função seguinte de acesso de memória. Após uma investigação minuciosa, foram observados dois principais equívocos. O primeiro ponto a ser considerado é que, no ciclo de identificação de tamanho, foi utilizado o endereço da variável mais o acumulado de tamanho das interações anteriores e a função. Apesar de evitar sobrescrever a memória, a função sempre considera que o primeiro endereço está disponível. Em segundo lugar, na função “snprintf”, há um argumento que informa até quanto espaço será ocupado. No entanto, esse argumento foi novamente presumido, ao invés de medido.

Figura 20 – Conversão de dados em JSON



Fonte: Autoria própria

Figura 21 – Converter tabela para JSON



Fonte: Autoria própria

Com o intuito de solucionar o problema em pauta, foi utilizado o valor “NULL” como endereço e o valor 0 como espaço. Dessa forma, o comando “snprintf” retorna o tamanho da memória que seria ocupado, sem que haja uma tentativa de gravação. Dessa maneira, o problema imediato de acesso à memória foi resolvido. Entretanto, após um tempo, foi exibido um erro informando que não foi possível alocar, devido à falta de liberação de memória pelo comando “free()”. Após a correção, percebeu-se que seria mais adequado ter uma variável local com o *buffer* do texto, auxiliada por uma variável de tamanho que manteria alocado o espaço utilizado de memória para o texto. A determinação em questão foi estabelecida em virtude da similaridade entre os dados em termos de quantidade de memória ocupada. Somente foi necessária a realocação do *buffer* no caso de o tamanho deste ser inferior ao do texto a ser gerado. Para mitigar a quantidade de realocações, o tamanho do *buffer* é aumentado em passos e é sempre mantido alocado para o maior dado já lido.

Após uma análise mais aprofundada das tabelas completas, foi possível identificar que elas estão divididas em tabelas menores. Nesse caso, os dois primeiros parâmetros são o ID e o tamanho da tabela. Ademais, é possível ter tabelas com o mesmo ID e tamanho diferente. Nesse caso, o de menor tamanho é truncado. Dessa forma, as variáveis foram modificadas para a tabela inicial vista na Tabela 1 e duas outras, a 10X, disponibilizada na Tabela 3, que é a tabela com inteiro e fator de escala, e 11X, disponibilizada na Tabela 2, que é a tabela com ponto flutuante. O objetivo do estudo foi criar um vetor contendo todos os modelos em forma de uma estrutura para cada tabela. A estrutura em questão abarca o tipo, a etiqueta, o fator de escala (quando aplicável), a unidade e o número de registradores. Cada modelo, que também pode ser considerado uma estrutura, contém a tabela, o número de valores que a compõem e o número total de registradores. Abaixo como ficou a tabela 1 no código após a transformação feita em Python das tabelas em Excel:

```
const table_sunspec table_1[9] = {
    {.type = SUNSPEC_TYPE_UINT32,.label = "SID",.scale = "",.units = "",.size = 2},
    {.type = SUNSPEC_TYPE_UINT16,.label = "ID",.scale = "",.units = "",.size = 1},
    {.type = SUNSPEC_TYPE_UINT16,.label = "L",.scale = "",.units = "Registers",.size = 1},
    {.type = SUNSPEC_TYPE_STRING16,.label = "Mn",.scale = "",.units = "",.size = 16},
    {.type = SUNSPEC_TYPE_STRING16,.label = "Md",.scale = "",.units = "",.size = 16},
    {.type = SUNSPEC_TYPE_STRING8,.label = "Opt",.scale = "",.units = "",.size = 8},
    {.type = SUNSPEC_TYPE_STRING8,.label = "Vr",.scale = "",.units = "",.size = 8},
    {.type = SUNSPEC_TYPE_STRING16,.label = "SN",.scale = "",.units = "",.size = 16},
    {.type = SUNSPEC_TYPE_UINT16,.label = "DA",.scale = "",.units = "",.size = 1},};
```

Com esse mesmo código é possível gerar as constantes para qualquer tabela do padrão da SunSpec Alliance que tenha o cabeçalho ilustrado na figura 22

incluindo as novas tabelas 7XX para quando os inversores forem mais atuais.

Figura 22 – Cabeçalho Excel

	A	B	C	D	E	F	G	H	I	J	K
1	Smart Meter Register Map with Float AC-Meter Model										
2											
3	Start	End	Size	R/W	Function codes	Name	Description	Type	Units	Scale Factor	Range of values
4	40001	40002	2	R	0x03	SID	Well-known value. Uniquely identifies this as a SunSpec Modbus Map	uint32			0x53756e53 ('SunS')

Fonte: Autoria própria

3.2 Thingsboard

Para a visualização dos dados recebidos do inversor, foi utilizado o Thingsboard em um computador conectado à rede local. O envio de dados é realizado por padrão via MQTT em formato JSON. Para tanto, foi utilizado um *container* com a imagem base do Thingsboard, já configurada e disponibilizada pela ThingsBoard ([s.d.]b).

Assim foi configurado o *container* com podman, pois se trata de uma alternativa ao docker que já é utilizada na distribuição de Linux configurada no computador assim tendo uma melhor compatibilidade. Para que o *container* funcione corretamente, foi necessário gerar duas pastas para armazenamento de *log* e de dados, configurar as devidas permissões de acesso às pastas e garantir que o *container* seja criado usando o *namespace* correto para a conexão do MQTT e outros recursos, como o acesso pelo navegador. Também foi preciso definir as portas da rede que o *container* pode acessar. Para criar e executar o *container* em modo interativo, permitindo a visualização no terminal do *log* que também é gravado na pasta configurada foi usado o comando abaixo:

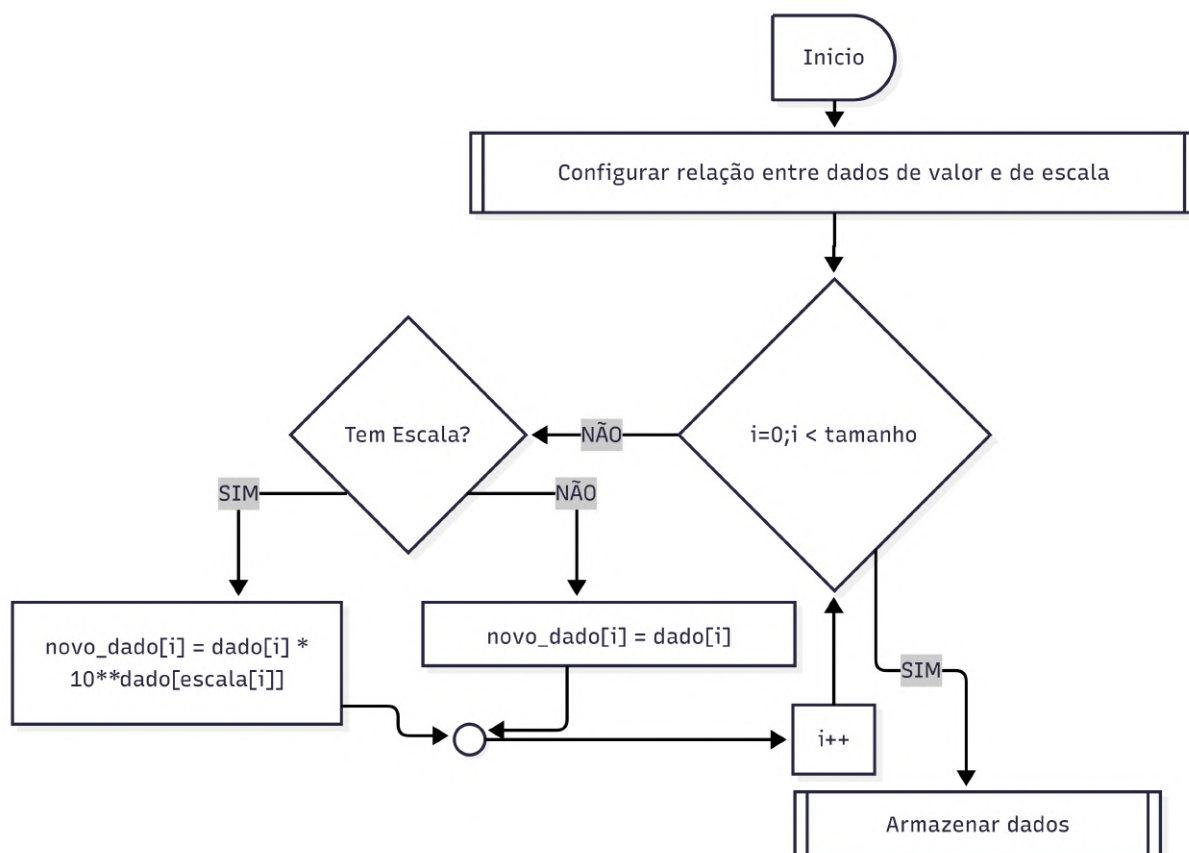
```
podman run --rm -it -p 9090:9090 -p 1883:1883 -p 7070:7070 -p
5683-5688:5683-5688/udp -v $(pwd)/.mytb-data:/data:z -v $(pwd)/.mytb-logs:/var/
log/thingsboard:z --name mytb --usersns keep-id:uid=799,gid=799 thingsboard/tb-
postgres
```

Para que o ESP32-C6 conseguisse se comunicar, foi utilizado como referência o exemplo de comunicação com MQTT disponibilizado pela Espressif. Ademais, foi analisada a possibilidade de publicação a partir de outra função. Para avaliar a operação do código, foi utilizado instalado e configurado no computador um servidor simplificado de MQTT, o Mosquitto, por ser de fácil instalação e por ser um programa de código aberto. Dessa forma, foi iniciado o servidor e um cliente na máquina, com o objetivo de observar o comportamento inicial do código-exemplo e verificar seu funcionamento.

Após a realização dos testes, foi possível constatar que o código de exemplo estabelecia a conexão, enviava as mensagens de teste e se desconectava. Considerando que, para um envio constante de dados com uma frequência na ordem de segundos, o processo de conexão e desconexão se mostrava inviável, a função de desconexão foi removida. Para integrar o código de aquisição, foram necessárias algumas modificações, de modo a permitir o envio das mensagens após a leitura e conversão de cada dado.

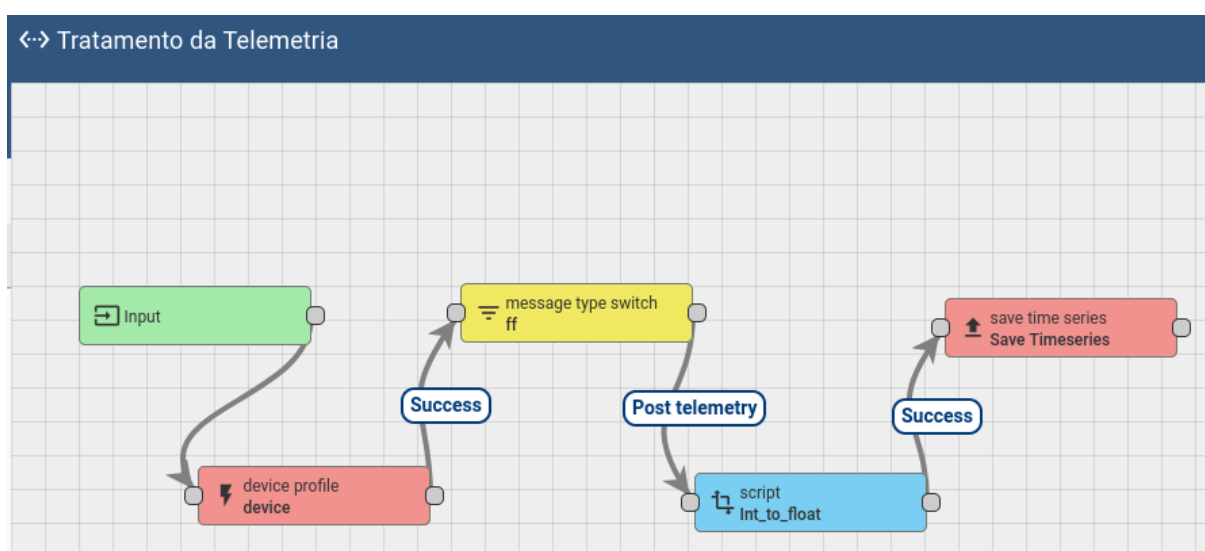
Para tanto, o código de envio de mensagens precisou fornecer o endereço para a função de aquisição. Duas formas de envio da mensagem foram testadas: uma delas consistia em usar diretamente a função de publicação, após o tratamento do dado, enquanto a outra envolvia a geração de um evento para ser tratado em outra tarefa. A publicação direta foi a primeira a apresentar resultados positivos e foi utilizada em parte dos testes de desenvolvimento. No entanto, o uso da chamada de evento seria mais adequado, uma vez que permitiria o funcionamento do envio e da aquisição de forma paralela. Dessa forma, caso a conexão esteja lenta, o algoritmo de aquisição não precisaria esperar o envio completo da mensagem antes de prosseguir para uma próxima leitura. Dessa forma, foi realizada uma análise detalhada do código e da documentação do exemplo, com o objetivo de compreender como seria viável gerar um evento próprio que enviasse o JSON. Constatou-se que a dificuldade inicial estava na declaração do evento. Após a correção dessa etapa, o envio por meio de evento foi bem-sucedido.

A função de tratamento dos dados foi desenvolvida em JSON, porém alguns ajustes foram realizados para que as funções de tratamento do Thingsboard fossem executadas. A etapa que demandou maior esforço foi a conversão automática de inteiro com escala para ponto flutuante. Para tanto, primeiramente foi utilizada a função *Calculated Fields* do Thingsboard, que contava com uma estrutura contendo etiqueta, valor e escala, além de um ciclo que verificava se havia escala. Caso a escala estivesse presente, a etiqueta correspondente seria buscada e aplicada. Após a execução dos testes, inclusive com o dado completo, os dados foram enviados, contudo a função não estava operando. Após uma análise detalhada, foi constatado que o *Calculated Fields* apresenta uma limitação de tamanho. Foram investigadas novas formas de aplicação da conversão. Ao executar o *script* diretamente com a mensagem em JSON recebida, após adaptar o dado e a função, foi possível concluir que, nas cadeias de regras, a conversão pode ser executada. Foi observado que os dados submetidos foram devidamente tratados por meio do método de tratamento aplicado. Para entender como o *script* opera na Figura 23 é possível observar um fluxograma de operação. Conforme demonstrado na Figura 24, é possível observar que o *script* atua antes dos dados serem armazenados no servidor pela cadeia de regras.

Figura 23 – Fluxograma do *script*

Fonte: Autoria própria

Figura 24 – Aba de cadeia de regras no ThingsBoard



Fonte: Autoria própria

Após uma análise mais aprofundada do envio, observou-se que o dado enviado não necessitava conter o fator de escala, uma vez que as tabelas já continham um fator de escala fixo. Diante do exposto, foi utilizada uma constante no *script* do

Thingsboard para conectar as etiquetas e seus fatores de escala. Dessa forma, o dado enviado foi simplificado, de modo que primeiro foi inserida a etiqueta contendo a tabela utilizada e, posteriormente, as etiquetas e os valores dos dados.

Para o desenvolvimento do *dashboard*, foi necessário analisar as duas tabelas utilizadas, a fim de identificar os dados que poderiam ser adquiridos. Posteriormente, foi criado um gráfico para cada conjunto de dados. Por exemplo, as três tensões de fase foram agrupadas no mesmo gráfico, enquanto as três tensões de linha foram dispostas em outro gráfico conjunto. A frequência, por sua vez, exigiu um gráfico exclusivo. Foram elaborados gráficos para todos os dados possíveis, inclusive os que já se sabia que o modelo do inversor não informaria. Nesse caso, sempre é retornado NaN (um valor nulo) no caso de ponto flutuante ou 0xFFFF no caso de inteiro com escalar, por questão de compatibilidade com outros modelos que podem informar e por questão de teste. Se nesses gráficos obtiver um valor diferente dos mencionados, sabe-se que ocorreu algum erro e todo dado daquele instante não se torna confiável.

Com o intuito de aprimorar a visualização dos gráficos, foi empregada a resolução da ANEEL (2021) para estabelecer os limites de frequência e fator de potência. Posteriormente, os gráficos foram coloridos de acordo com esses limites, utilizando-se a gradação verde, amarelo e vermelho. Dessa forma, o verde representa o cumprimento completo dos limites, o amarelo indica a permanência dentro das tolerâncias estabelecidas, que não devem exceder um tempo definido, e o vermelho e tons no meio indicam tolerâncias de tempo menores até a proibição completa.

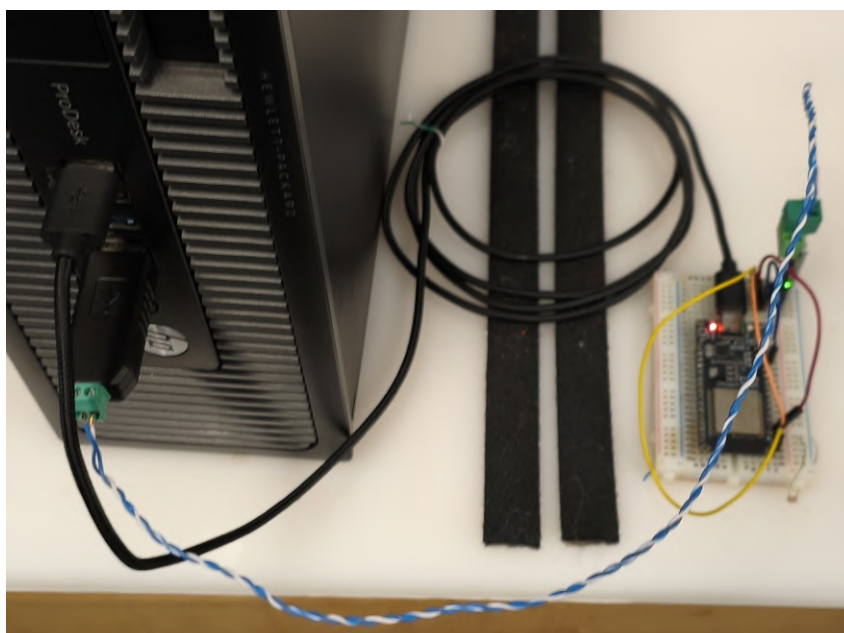
3.3 Inversor Fronius

Para estabelecer a conexão com o inversor, mostrou-se imprescindível utilizar um conversor RS485 em vez do FTDI, que era empregado anteriormente. Para garantir a segurança antes de conectar no inversor, foi realizado um teste com o conversor conectado ao ESP32-C6, operando como *master*, e o computador, operando como simulador de *slave*, no caso, o inversor, com a conexão física demonstrada na figura 25. Por conta do conversor utilizado ter um *auto_enable* não foi necessário fazer nenhuma alteração no código para esse teste.

Após a realização do teste no simulador, cujo resultado foi bem-sucedido, foi efetuada a conexão física com o inversor, em conformidade com as instruções contidas no guia do Fronius.

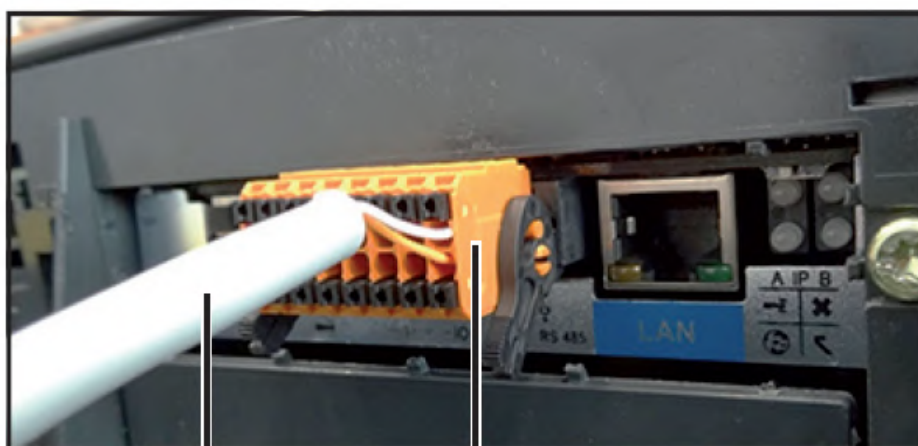
A Figura 26 ilustra a localização das conexões entre os fios, demonstrando onde cada fio deve ser conectado para realizar a conexão de maneira adequada. A Figura 27 apresenta um relato visual do processo de conexão aplicado neste trabalho.

Figura 25 – Conexão ESP32 e Computador via RS485



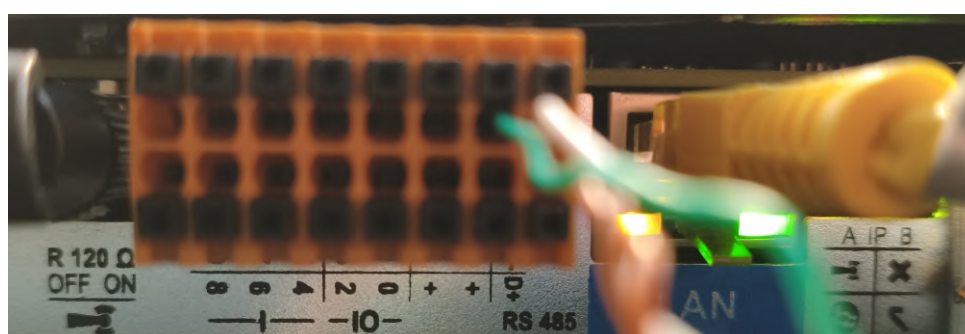
Fonte: Autoria própria

Figura 26 – Exemplo de conexão no manual



Fonte: Fronius ([s.d.], p.4)

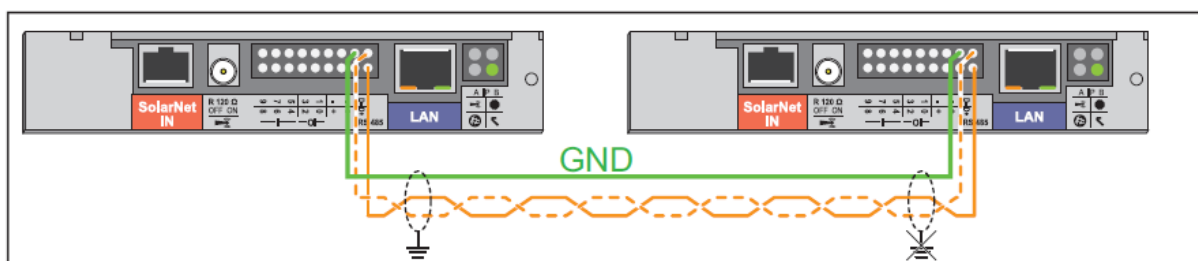
Figura 27 – Conexão física realizada



Fonte: Autoria própria

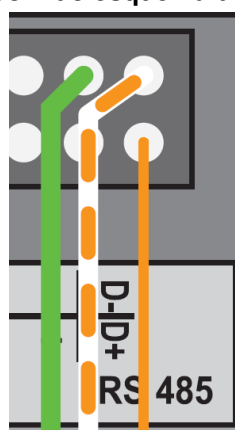
Para determinar as conexões dos fios na placa, foi utilizado o diagrama esquemático apresentado na figura 28, com zoom na figura 29, para identificar os terminais D+ e D- do RS485.

Figura 28 – Esquema da conexão física



Fonte: Fronius Datamanager 2.0 Modbus RTU Quickstart Guide(FRONIUS, [s.d.], p.5)

Figura 29 – Zoom do esquema da conexão física



Fonte: Fronius Datamanager 2.0 Modbus RTU Quickstart Guide(FRONIUS, [s.d.], p.5)

Após a conexão do ESP32-C6, já com o código sendo testado, foi observado o seguinte resultado:

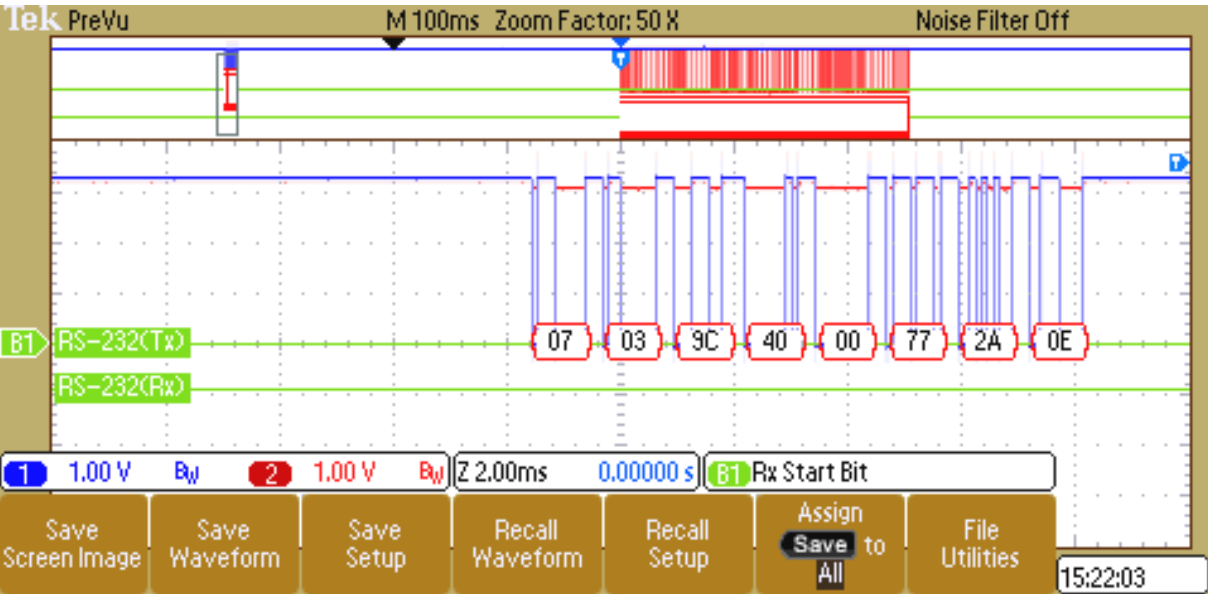
```
I (9956) MASTER_TEST: Characteristic #0(ID) (0x) value = 0.000000 (0x3c006f)4  
read successful.
```

```
I (9956) MASTER_TEST: ID = 111 | L = 60
```

Conforme observado, o dispositivo identificou corretamente o ID da tabela. Dessa forma, foi constatado que a conexão física está operando corretamente. Entretanto, após a identificação da tabela, com a subsequente execução do algoritmo e a leitura completa da tabela, foram apresentados os erros `ESP_ERR_INVALID_RESPONSE` ou `ESP_ERR_TIMEOUT`. Para investigar o erro, foi realizada uma medição por meio de um osciloscópio.

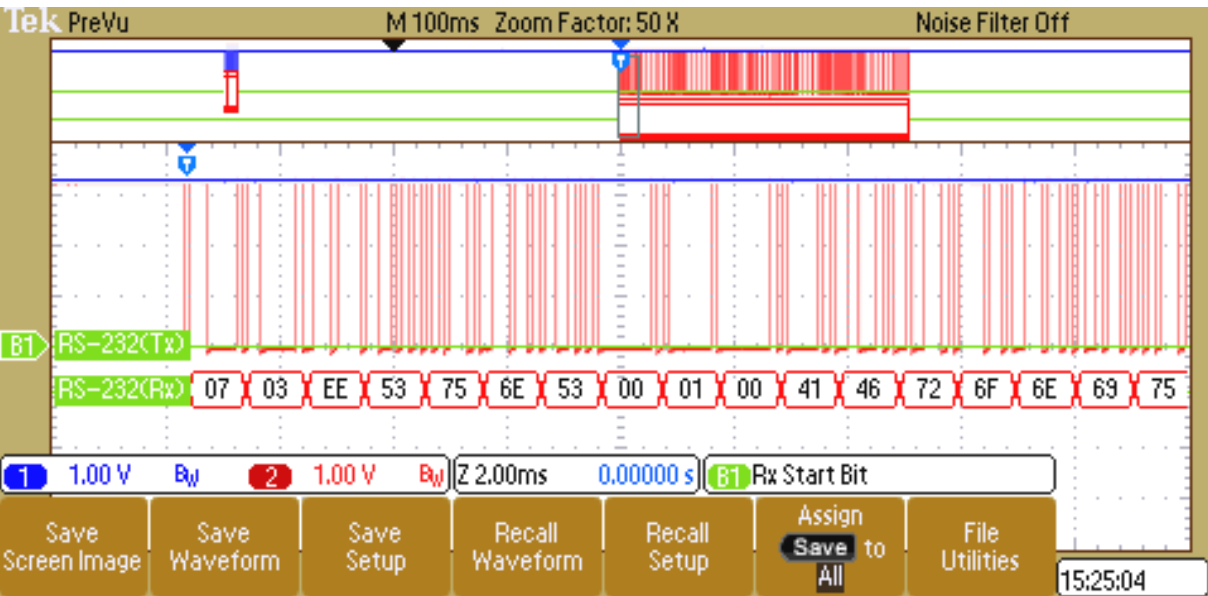
Conforme demonstrado na Figura 30, o ESP32-C6 realizou a requisição conforme o esperado para a leitura da Tabela 111. Na Figura 31, é possível observar o início da resposta, a qual também segue de acordo com o protocolo.

Figura 30 – Requisição feita pelo ESP



Fonte: Autoria própria

Figura 31 – Resposta dada pelo inversor



Fonte: Autoria própria

Após a análise da comunicação entre as partes, foi observado que a requisição e a resposta não apresentavam erros aparentes. No entanto, o tempo entre as mensagens parecia longo. Diante disso, foi gerada uma tabela pelo osciloscópio contendo o tempo detalhado da comunicação, que pode ser observado na tabela 4.

Com base nos dados obtidos pelo osciloscópio, foi identificado um atraso de aproximadamente 350 milissegundos entre a solicitação e a resposta. Após a consulta das configurações da biblioteca do protocolo ModBus, foi identificado o parâmetro CONFIG_MB_MASTER_TIMEOUT_MS_RESPOND, configurado em 400 milissegundos.

Tabela 4 – Detalhes da comunicação

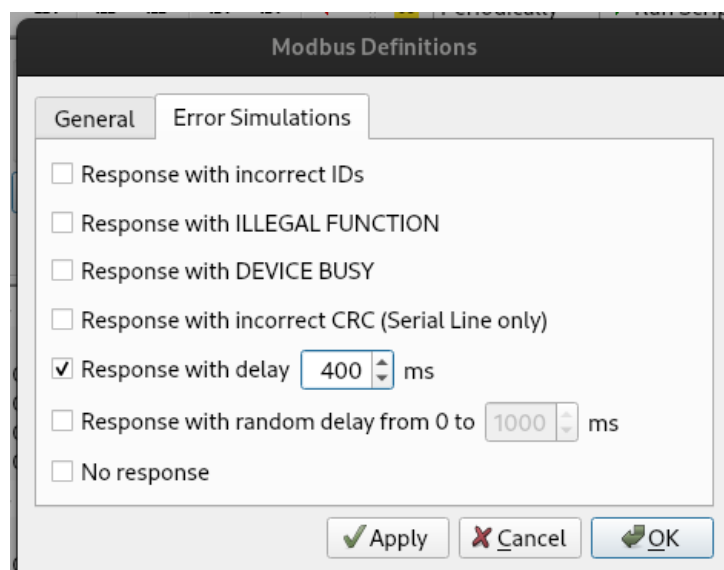
Time	Tx	Rx
-347,17E-03	07	
-345,92E-03	03	
-344,65E-03	9C	
-343,41E-03	40	
-342,16E-03	00	
-340,91E-03	77	
-339,66E-03	2A	
-338,41E-03	0E	
3,20E-06		07
83,20E-06		03
123,20E-06		EE
163,20E-06		53
203,20E-06		75
243,20E-06		6E
283,20E-06		53
344,00E-06		00
384,00E-06		01
424,00E-06		00

Fonte: Autoria própria

Dessa forma, a leitura do identificador da tabela é realizada de maneira precisa, e a solicitação maior recebe o *timeout* ou *invalid response* (quando lê um trecho antes de acontecer o *timeout*). Constatou-se, durante a aquisição do osciloscópio, que a resposta para uma requisição de 119 registradores demanda aproximadamente 250 milissegundos. Fazendo-se a soma das duas leituras, chega-se a um tempo mínimo de 600 milissegundos.

Dessa forma, com o objetivo de assegurar a leitura, o *timeout* foi configurado para 1 segundo e submetido a testes antes da conexão com o inversor. Para simular o atraso do inversor, utilizou-se a configuração do OModSim, conforme demonstrado na Figura 32. Após a conclusão dos testes, que foram bem-sucedidos, o dispositivo foi conectado ao inversor, e a comunicação ocorreu conforme o esperado.

Após a análise dos dados, foi possível constatar que o inversor configurado em inteiro com escalar apresentava coerência com as informações exibidas nos *dashboards*. No entanto, ao testar o inversor em *float*, os valores obtidos apresenta-

Figura 32 – Configuração usada no OModSim

Fonte: Autoria própria

ram discrepâncias. Ao examinar o texto de descrição recebido, notou-se que as letras estavam embaralhadas em cada par. Considerando que os registradores contêm dois caracteres, foi percebido que havia um problema de *little endian* e *big endian*, que são formas de ler *bytes*. Supondo um registrador exemplo ABCD, sendo “A” o mais significativo e “D” o menos, seria lido de maneira distinta, pois o *little endian* considera os algarismos mais significativos à direita de cada *byte*, lendo o registrador exemplo como BADC, enquanto o *big endian* considera o mais significativo à esquerda. Assim, o mesmo registrador seria lido como ABCD.

Após a alteração do protocolo para a leitura adequada da resposta do ESP32-C6, foi identificada a mensagem de parâmetro inválido. Dessa forma, foi necessário realizar um processamento dos dados para ordenar a saída corretamente. Contudo, o processo revelou-se mais longo que o projetado, em virtude de uma confusão na utilização da função “cast”. Para realizar a ordenação, foi necessário empregar um *cast* como inteiro de 8 bits. Ao realizar o *recast* para ponto flutuante, o inteiro foi apenas arredondado. Portanto, com o intuito de organizar corretamente, foi desenvolvido um “*union*” de bits com *float*, que permite a ordenação adequada, garantindo, consequentemente, a coerência dos valores.

Após a implementação das correções, o algoritmo foi submetido a testes para avaliar o seu comportamento em caso de troca da configuração do inversor para outra tabela. Constatou-se que, após o primeiro teste, o algoritmo não confirmava se a tabela lida era a correta, embora essa informação estivesse presente nos dados. Diante disso, foi inserida no processo de conversão para JSON uma checagem da tabela, de modo que, em caso de mudança, o algoritmo se adaptasse sem a necessi-

dade de reiniciar o equipamento.

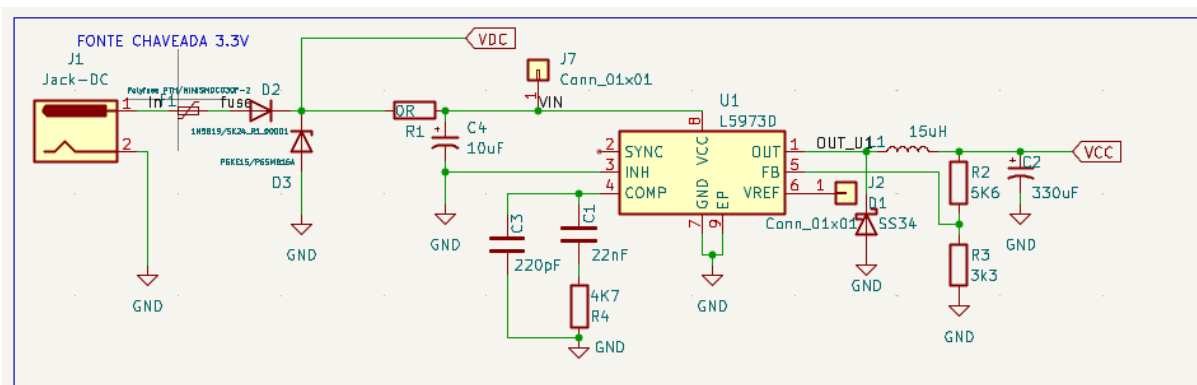
3.4 Hardware

Para se obter um protótipo funcional em uma peça única, que garanta a operação do microcontrolador e do algoritmo, foi necessário projetar uma placa de circuito impresso, diminuindo, assim, as incertezas causadas pela matriz de contato e pela conexão entre módulos. Para esse fim, foi utilizado o software KiCad, pois ele é gratuito, possui uma grande variedade de bibliotecas e integra muito bem a criação do esquema e a do layout em PCB. Por esses motivos, esse software foi amplamente utilizado no decorrer do curso, o que me proporcionou conhecimento prévio sobre sua utilização, facilitando a elaboração do layout.

3.4.1 Esquemático

Para o desenvolvimento dessa placa, ela foi projetada em três partes: a alimentação, apresentada na Figura 33, o conversor RS485, apresentado na Figura 34, e as conexões com o microcontrolador e seu circuito mínimo, apresentadas na Figura 35.

Figura 33 – Esquemático da fonte chaveada 3.3 Volts

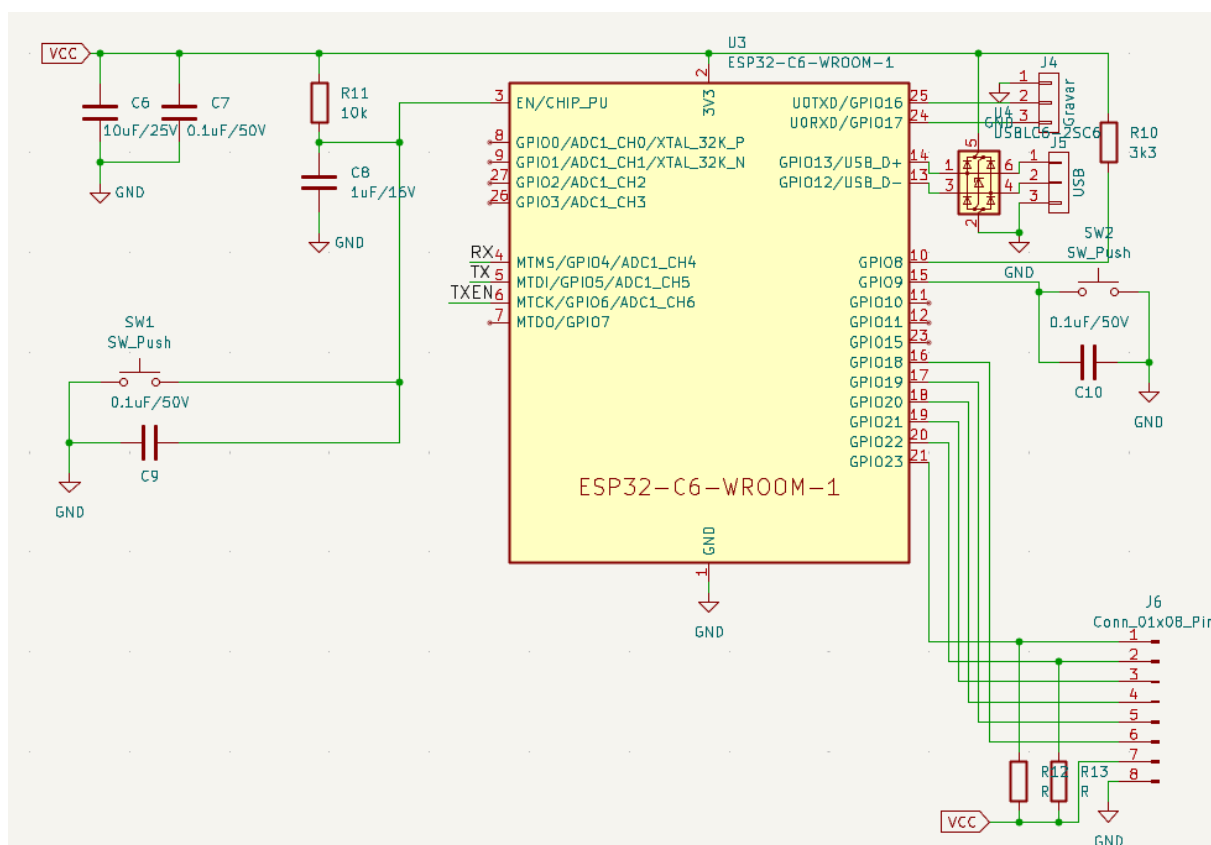


Fonte: Autoria própria

Para a alimentação, foi utilizado um projeto de fonte chaveada de 3,3 V, que já havia sido empregado em outros projetos do laboratório. O projeto em questão foi submetido a testes e conta com componentes disponíveis no laboratório. O emprego em diferentes projetos deve-se, sobretudo, à sua aptidão em diminuir a perda de energia quando comparada a uma fonte linear simples, o que é pertinente para a questão da temperatura, sendo consistente com a utilização de um microcontrolador de baixo consumo.

Para a realização da conversão, foi utilizado o componente 75LS176, que se encontrava de fácil acesso. No entanto, esse componente não era o ideal, pois

Figura 35 – Esquemático do ESP32-C6 e ligações base

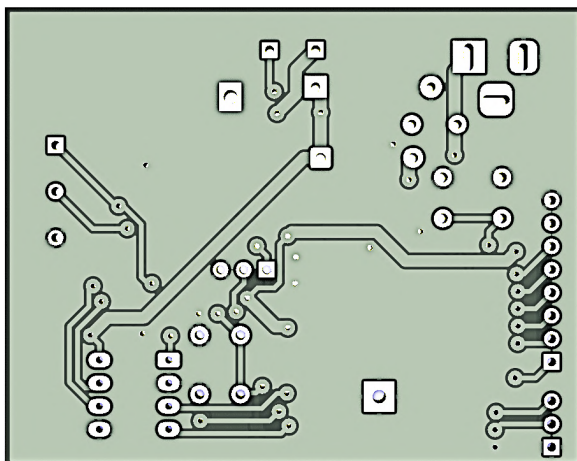


Fonte: Autoria própria

3.4.2 Layout

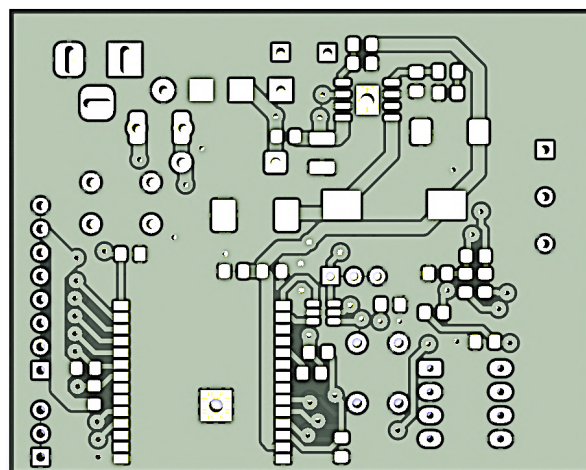
Conforme demonstrado na Figura 36, é possível observar o layout da camada inferior. A Figura 37 mostra o layout da camada superior. Por fim, a Figura 38 apresenta o layout completo.

Figura 36 – Bottom Layer



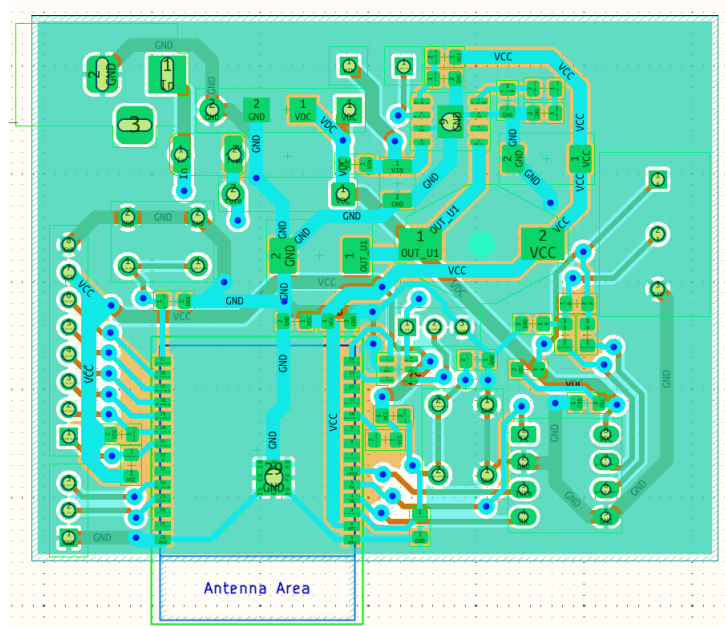
Fonte: Autoria própria

Figura 37 – Top Layer



Fonte: Autoria própria

Figura 38 – Layout KiCAD



Fonte: Autoria própria

Segundo as diretrizes estabelecidas para a fresadora, um detalhe significativo foi considerado no layout: a fresa disponibilizada não possui a funcionalidade de vias metalizadas. Diante do exposto, foi necessário criar uma via ao lado de cada ilha de componentes PTH para conectar a camada superior, onde estão localizados os componentes SMDs. Ademais, para que haja um funcionamento adequado da fonte de alimentação, é imprescindível que os componentes estejam próximos e que as trilhas que operam em alta frequência sejam curtas, ao passo que as trilhas de alimentação devem ser mais largas que as de sinal.

3.4.3 Confecção e Testes

Após a finalização do layout, os arquivos necessários para a produção das trilhas da placa por meio da fresadora foram gerados. A fresadora foi então iniciada e configurada com base nos arquivos apresentados, conforme pode ser visualizado na Figura 39 a fresadora em operação para a criação das trilhas na placa.

A Figura 40 mostra a placa após o processo de fresagem. A continuidade entre as trilhas foi testada para verificar se houve falha no processo e se restou cobre entre as trilhas. Os testes foram positivos, não encontrando-se nenhum ponto de curto. Conforme demonstrado, a placa apresenta uma configuração distinta do layout, com três furos localizados nos cantos. Esses furos, classificados como furos fiduciais, são essenciais para a orientação do dispositivo durante o processo de fresagem. A disposição desses furos pode ocorrer na parte externa da placa de cobre. No entanto, devido à ausência de furos de sustentação na placa, esses furos são utilizados para

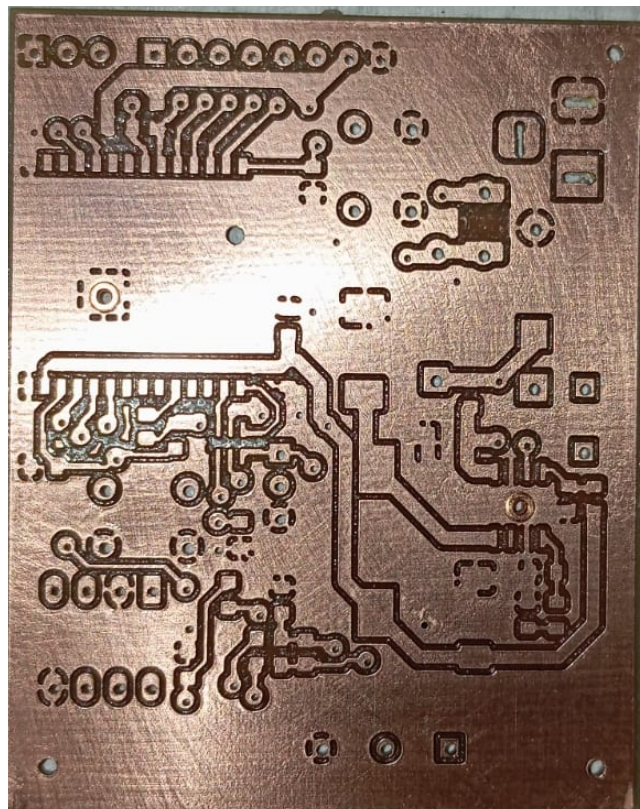
esse propósito específico.

Figura 39 – Confeção na fresa



Fonte: Autoria própria

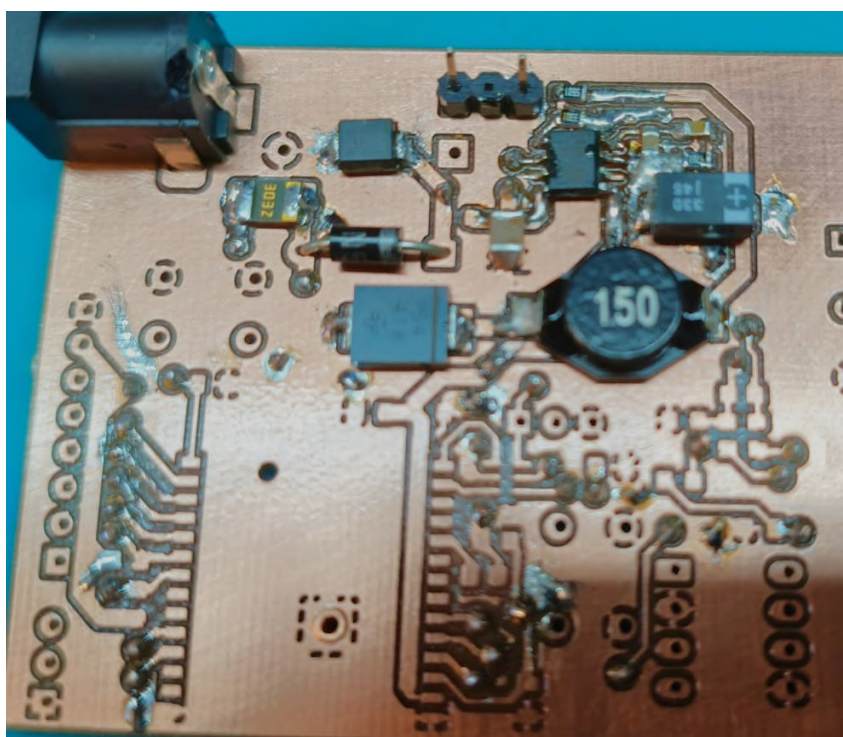
Figura 40 – Fresagem finalizada



Fonte: Autoria própria

A fabricação da placa foi iniciada com a soldagem de todas as vias, seguida pela realização dos testes de continuidade em cada uma e do teste de curto nas trilhas próximas, com ênfase na malha de terra. Após a soldagem de todos os componentes referentes à fonte chaveada, exceto o resistor de 0R, foram realizados os mesmos testes das vias. Posteriormente, o circuito foi alimentado com uma fonte de 5 volts e a tensão foi verificada até o resistor de 0R. Dessa maneira, o componente pode ser soldado para testar se a fonte chaveada forneceu os 3,3 volts previstos. O resultado dessa solda pode ser observado na Figura 41.

Figura 41 – Fonte 3.3V soldada



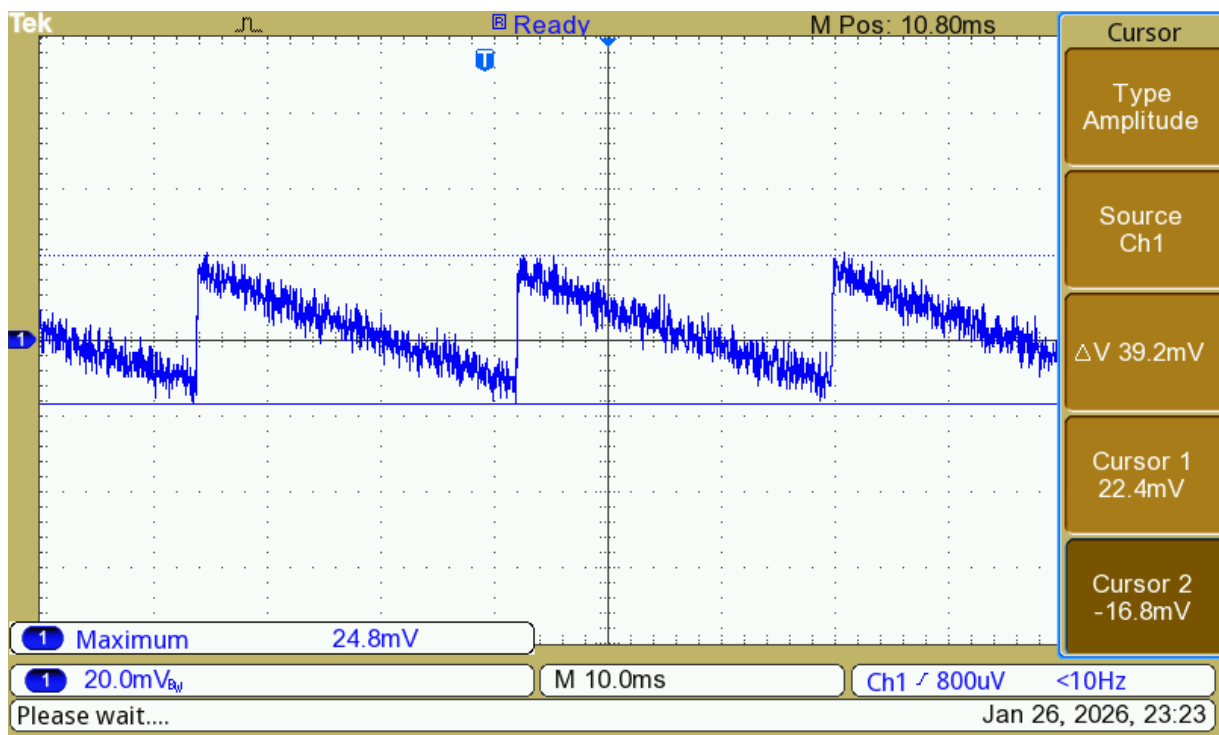
Fonte: Autoria própria

O teste inicial foi realizado com o uso de um multímetro, que mediu uma tensão de 3,377 V. Esse valor é ligeiramente superior ao projetado, porém está dentro dos limites de operação do ESP32-C6. A mensuração do *ripple* do circuito, definido como a oscilação na tensão de alimentação, foi realizada por meio de um segundo teste, utilizando-se um osciloscópio. O resultado dessa medição é apresentado na Figura 42. Dessa forma, foi constatado que o *ripple* encontrava-se em aproximadamente 1% a vazio, indicando um valor satisfatório.

Posteriormente, foi soldado o circuito do RS485, conforme demonstrado na figura 43, seguindo os mesmos procedimentos de teste de continuidade após cada componente.

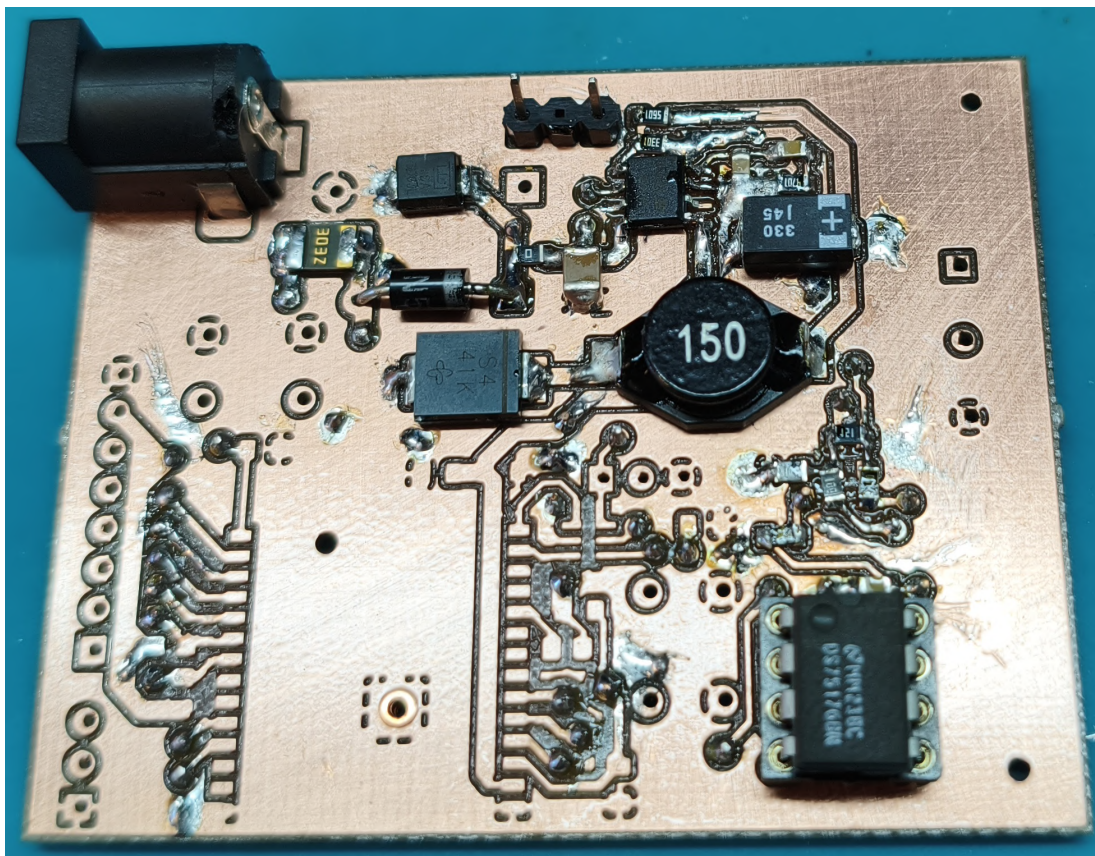
Além disso, foi realizado um teste do circuito utilizando o CuteCom no computador e um adaptador USB para simular o envio de dados do inversor para o ESP32-

Figura 42 – Ripple da fonte



Fonte: Autoria própria

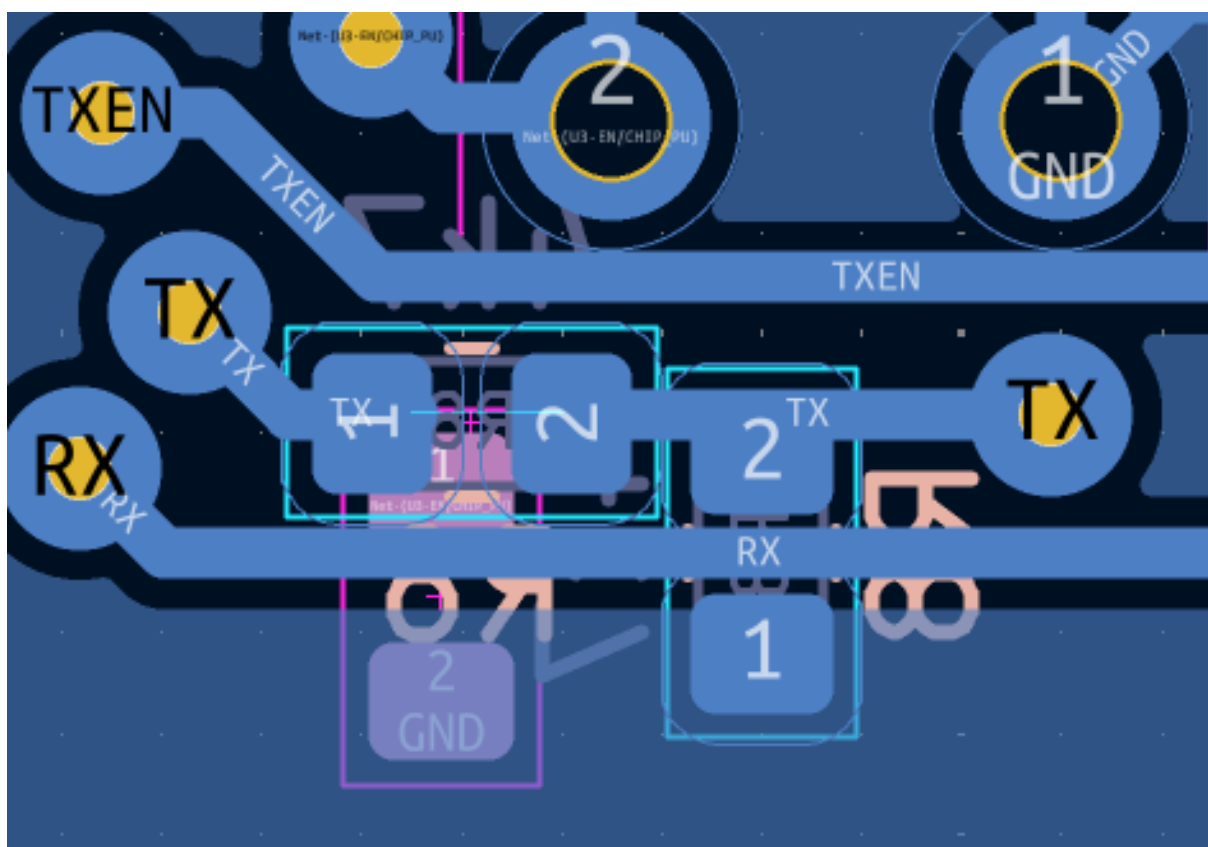
Figura 43 – RS485 soldado



Fonte: Autoria própria

C6. Foi observado, por meio da análise dos dados obtidos com o uso de um osciloscópio, que a tensão de saída do circuito integrado de conversão excede os limites operacionais dos pinos de entrada/saída do dispositivo ESP32-C6. Dessa forma, foi realizada uma adaptação na placa para incluir um divisor resistivo na trilha do TX do conversor, conforme ilustrado na Figura 44.

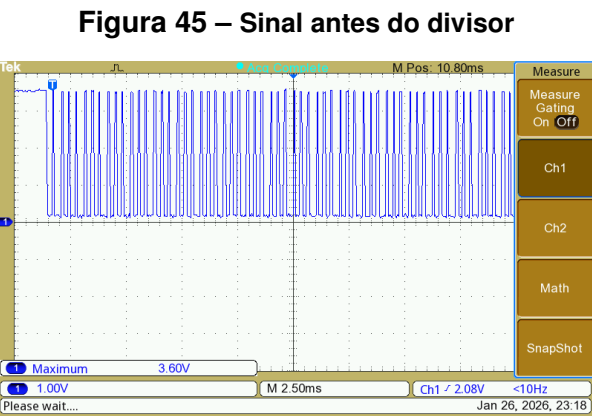
Figura 44 – Adaptação



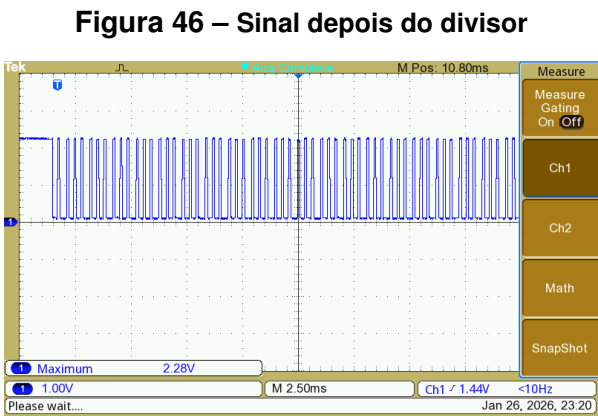
Fonte: Autoria própria

A fim de averiguar a eficiência da adaptação, foi executado o mesmo procedimento de simulação do envio, com o osciloscópio mensurando antes do divisor resistivo. O resultado desse procedimento encontra-se disponível na figura 45. Outro procedimento foi realizado, com o osciloscópio mensurando após o divisor resistivo, e seu resultado pode ser visualizado na figura 46. Para verificar se a adaptação funcionou corretamente foi realizado o mesmo procedimento de simulação do envio com o osciloscópio medindo antes do divisor resistivo que pode ser visualizado na figura 45 e outro com o osciloscópio medindo após o divisor resistivo que pode ser visualizado na figura 46.

Após a validação do circuito conversor, foi iniciado o processo de soldagem do ESP32-C6 e dos circuitos essenciais, tais como os capacitores de acoplamento, os pinos do UART para gravação, os botões de inicialização e reinicialização. Para tanto, foi utilizado o mesmo adaptador FTDI empregado nos testes iniciais com simu-



Fonte: Autoria própria



Fonte: Autoria própria

lador, a fim de ler a UART e gravar o ESP32-C6. Os testes realizados demonstraram que o ESP32-C6 estava ligado, porém não era possível gravar. De acordo com os dados apresentados na Tabela 5, disponível no *datasheet* do ESP32-C6, o pino GPIO8 deve estar em um estado lógico 1 para ativar o modo de download, necessário para a gravação.

Tabela 5 – Controle do modo de inicialização do chip

Boot Mode	GPIO8	GPIO9
SPI boot mode	Any value	1
Joint download boot mode ²	1	0

¹ **Bold** marks the default value and configuration.

² Joint Download Boot mode supports the following download methods:

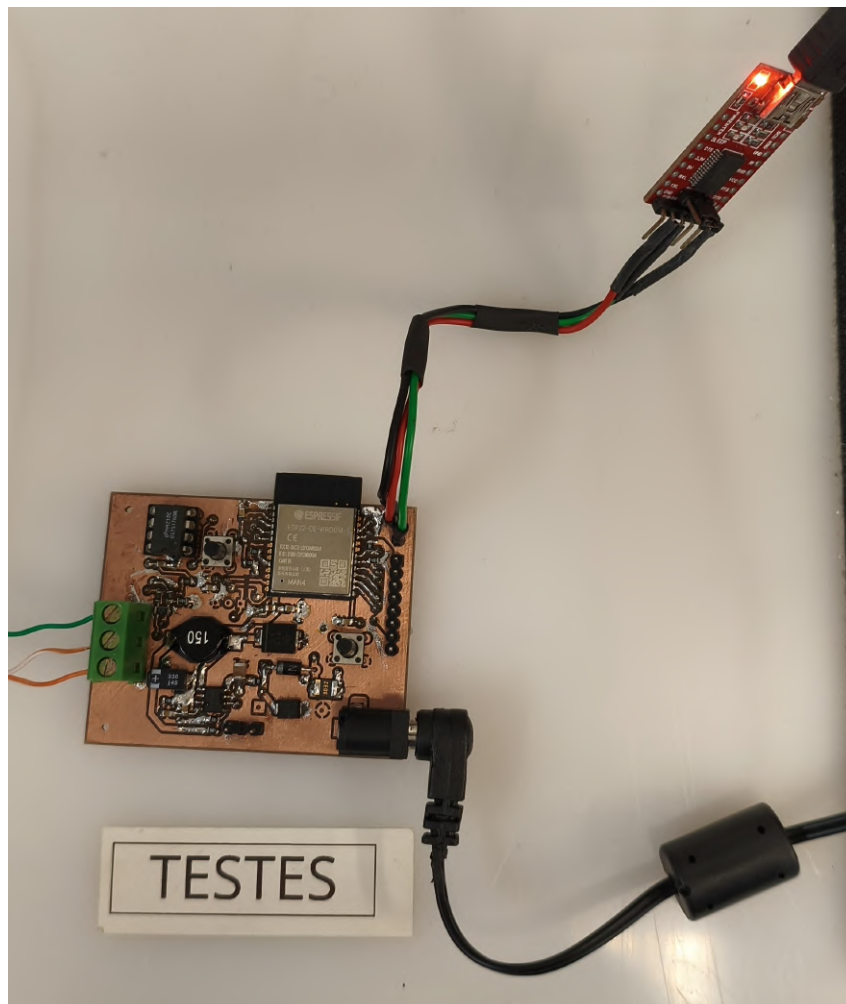
- USB-Serial-JTAG Download Boot
- UART Download Boot
- SDIO Download Boot

Adaptado de: Espressif ([2023?]b, p.13)

Portanto, o resistor de 3k3, conforme indicado no diagrama, foi conectado em *pull-up*, permitindo que a gravação ocorresse conforme o esperado e a saída no monitor apresentasse o resultado previsto. Posteriormente, foram soldados os pinos destinados à comunicação via ModBus e o barramento pino para SPI e I2C. No entanto, os pinos destinados à conexão USB não foram submetidos ao processo de soldagem, uma vez que sua implementação está prevista apenas em caso de falha na UART. Posteriormente, foi realizado o teste com o simulador do ModBus, que interpretou corretamente o sinal após a configuração do pino de RTS, pois o circuito de conversão utilizado é mais simples que o empregado nos testes com o kit de desenvolvimento e requer TX *enable*. Após a realização bem-sucedida do teste, foi realizada a conexão da placa ao inversor, e os dados reais foram analisados. Foi constatado que a placa apresenta o mesmo comportamento do kit de desenvolvimento em rela-

ção à comunicação ModBus. Conforme demonstrado na Figura 47, o circuito completo soldado encontra-se conectado à fonte, ao computador e ao inversor.

Figura 47 – Circuito completo



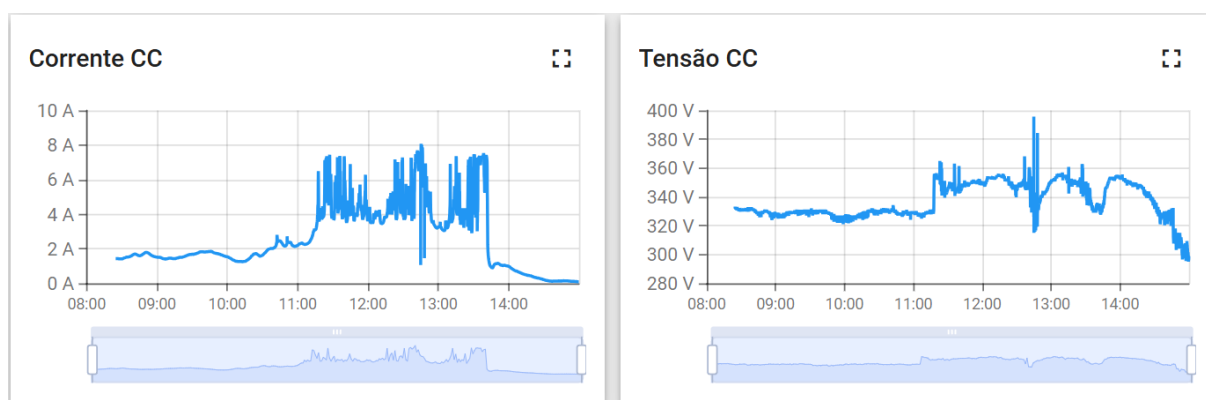
Fonte: Autoria própria

4 RESULTADOS E DISCUSSÕES

Para a realização da análise do protótipo em uso contínuo, foi necessário ligá-lo sob supervisão por um período de sete horas, a fim de obter uma série de dados contínuos nos *dashboards*.

Os gráficos representados na figura 48 correspondem à medição em corrente contínua de corrente e tensão. Esse é o modo pelo qual os painéis fotovoltaicos geram energia diretamente. Posteriormente, essa energia é processada pelos inversores e transformada em corrente alternada, que é o modo disponível nas tomadas. O gráfico da esquerda mostra a corrente gerada e consumida, enquanto o da direita mostra a tensão fornecida pelos painéis.

Figura 48 – CC

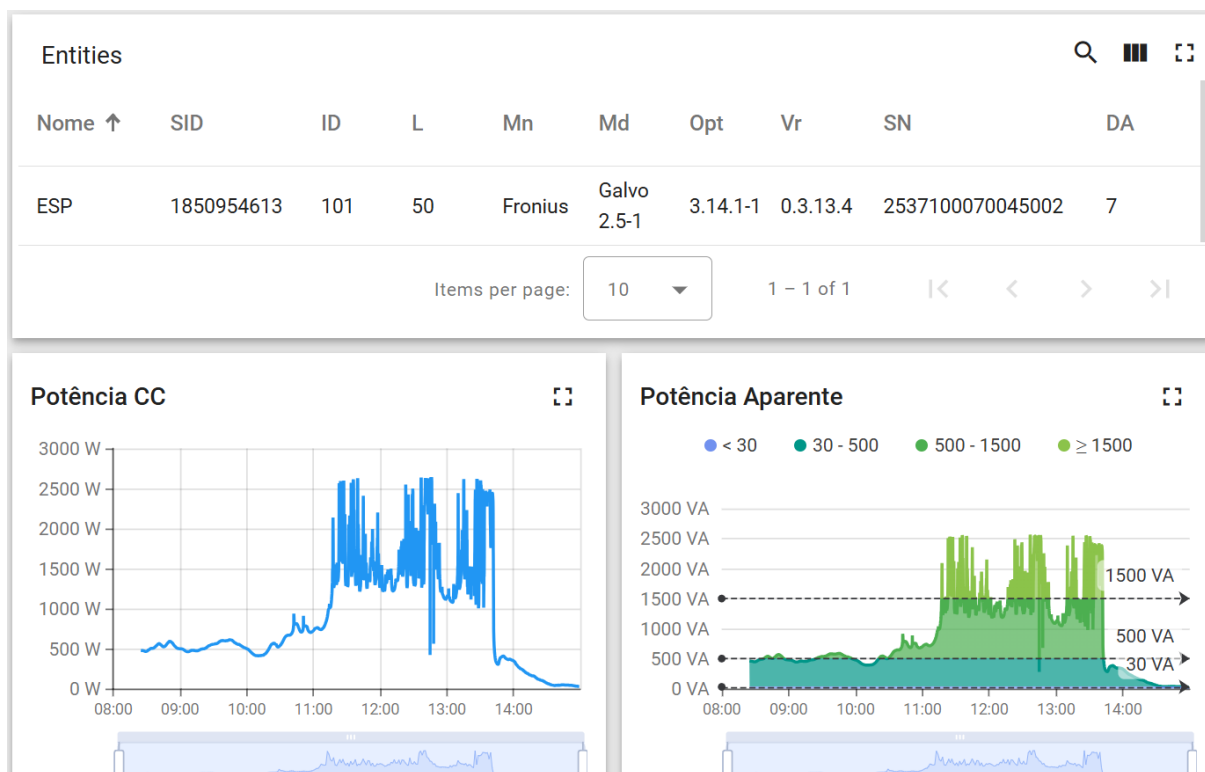


Fonte: Autoria própria

Conforme demonstrado na Figura 49, no topo da página estão dispostas as informações lidas a partir dos dados contidos na primeira tabela, comum a todos os inversores. Nessa tabela, é possível visualizar o SID do inversor, o ID da tabela que está sendo convertida, o seu comprimento no item “L”, o fabricante no item “Mn”, o modelo específico no item “Md”, a versão caracterizada no item “Vr”, o número de série no item “SN” e, por fim, o endereço do dispositivo no item “DA”. Os dados da potência aparente em corrente alternada (CA) e da potência em corrente contínua (CC) são apresentados nos gráficos da mesma imagem, à direita e à esquerda, respectivamente. A partir dessas informações, é possível calcular as perdas de transformação.

Conforme demonstrado na Figura 50, são apresentados dois gráficos: o da corrente alternada total à esquerda e o da corrente por fase à esquerda. No entanto, apenas a fase A está presente, pois o inversor em questão é monofásico. No caso do medidor monofásico, tanto o gráfico 1 quanto o gráfico 2 apresentam valores teóricos idênticos. No entanto, é possível estimar o consumo de energia. No dia em que essas informações foram coletadas, o dia estava parcialmente nublado pela manhã, o que gerou os picos e os vales no gráfico, de acordo com a cobertura das nuvens sobre os

Figura 49 – Potências



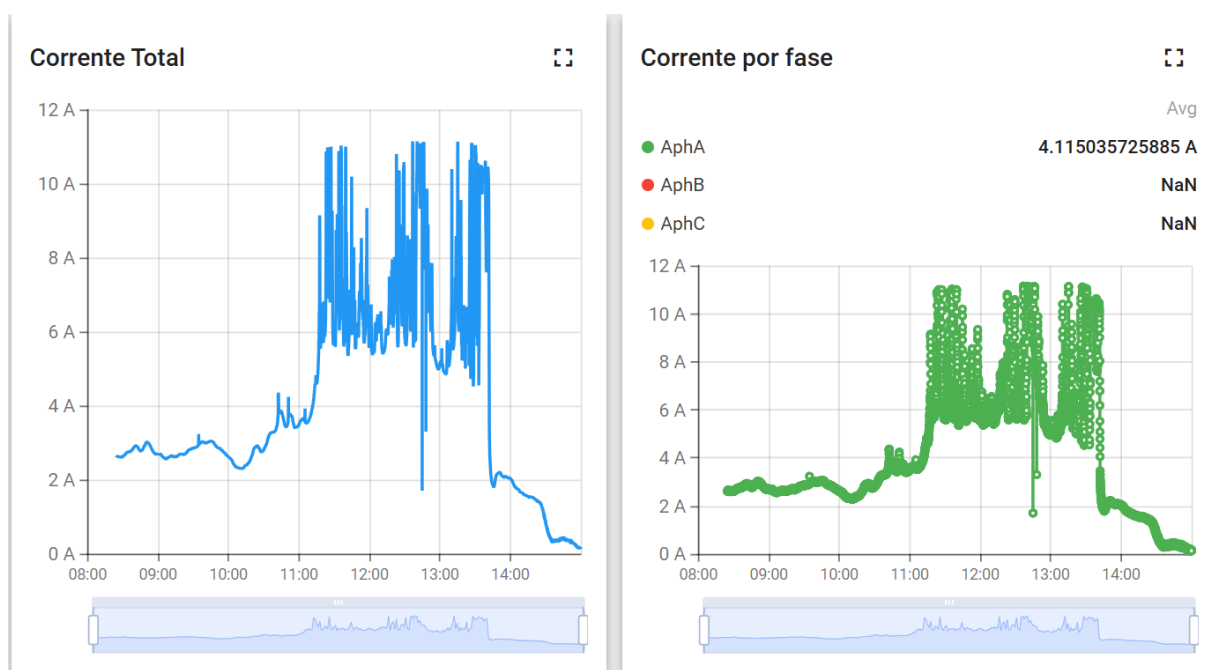
Fonte: Autoria própria

painéis. Durante a tarde, choveu intensamente, reduzindo significativamente a geração de energia nesse período.

Conforme demonstrado na seção do *dashboard* exibida na Figura 51, dois gráficos são apresentados. O primeiro, localizado à esquerda, representa a tensão de linha ou a tensão entre fases. No entanto, como a configuração é monofásica, não há dados disponíveis. Ainda assim, o gráfico é mantido, pois em modelos trifásicos ele auxilia na verificação da geração de energia. O segundo gráfico, localizado à direita, representa a tensão por fase. Por meio dele, é possível analisar a qualidade da energia gerada e verificar se está dentro dos limites de operação previstos pela ANEEL.

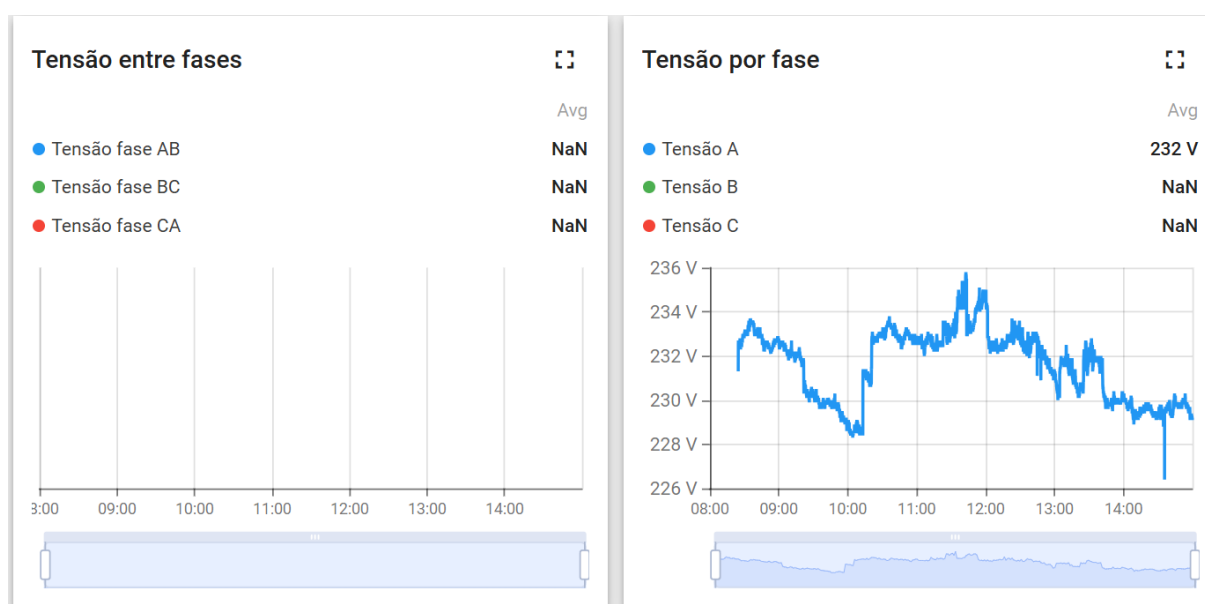
Conforme demonstrado no gráfico da esquerda da Figura 52, a temperatura não é fornecida pelo inversor em questão, contudo, está dentro do protocolo e pode ser fornecida por outros inversores que utilizem a mesma tabela, sendo esses dados úteis para saber em que condições o inversor está operando. Na direita, observa-se a energia acumulada que, para esse curto período e nível de geração, não apresentou grandes variações. Em períodos de tempo maiores ou com níveis de geração mais elevados, entretanto, é possível realizar uma análise mais macro sobre a geração. Ademais, em caso de falha na comunicação ou perda de dados, a energia acumulada permite o cálculo da geração no intervalo sem a necessidade de dados. Essa possibilidade é ilustrada na figura 53, na qual apenas foram adquiridos dados nos dias 29 de

Figura 50 – Corrente



Fonte: Autoria própria

Figura 51 – Tensão CA

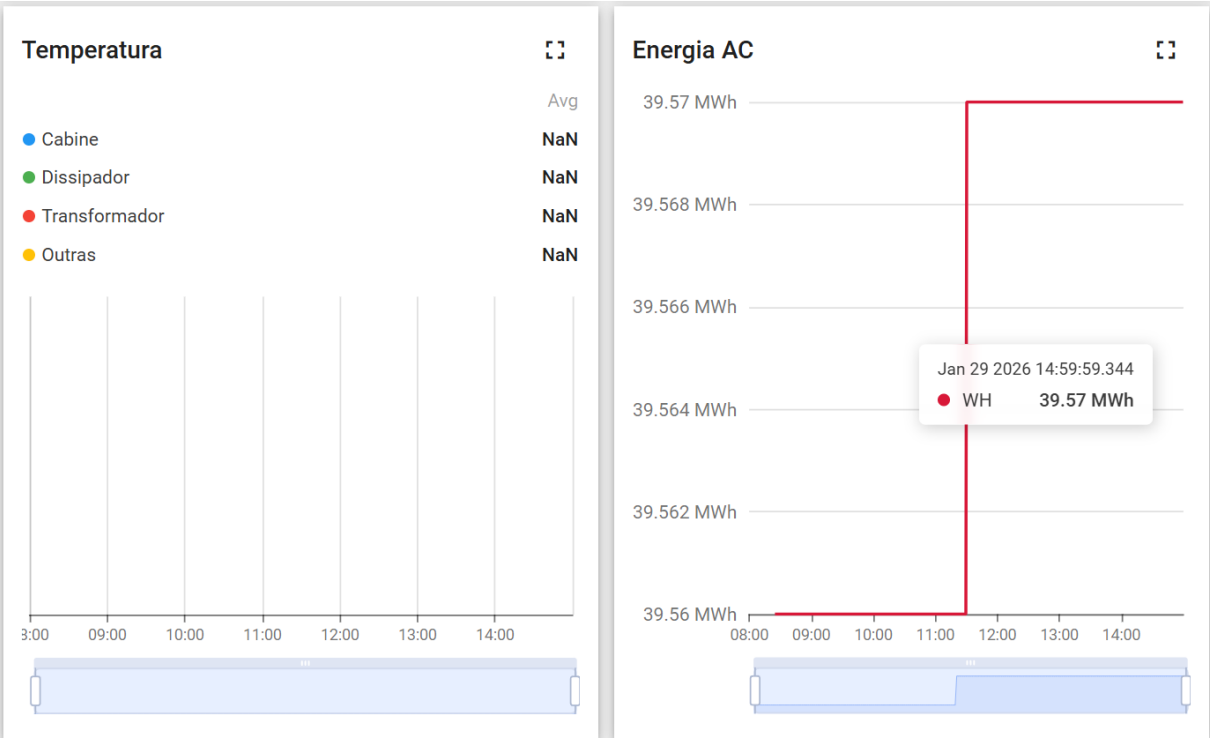


Fonte: Autoria própria

janeiro e 20 de fevereiro. Ainda assim, é possível determinar o total de energia gerada nesse intervalo de tempo.

A seção do *dashboard* exibida na figura 54 contém, à esquerda, o gráfico da frequência da tensão, o qual é de suma importância, uma vez que, segundo as normas da ANEEL (2021), deve permanecer em uma faixa bastante restrita. Ademais, há um período de tempo extremamente curto em que é possível que a frequência da

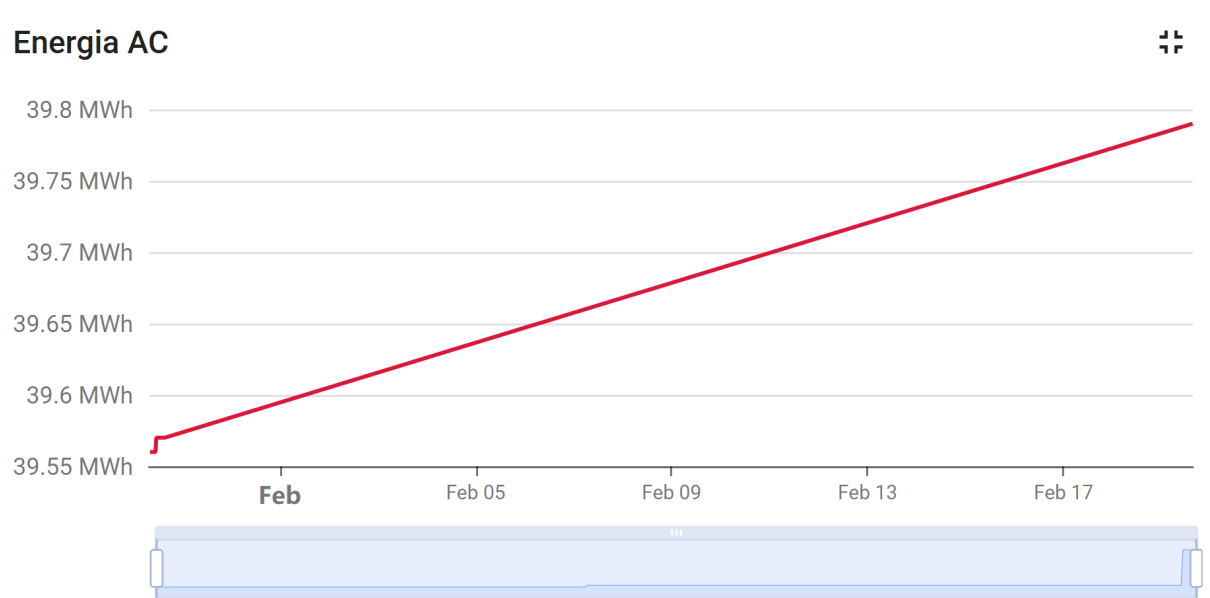
Figura 52 – Energia acumulada e Temperatura



Fonte: Autoria própria

tensão ultrapasse a faixa, em virtude da sincronização da geração. O gráfico à direita representa o fator de potência, que possibilita compreender o comportamento da carga de modo geral e determinar se a energia está com o fator de potência dentro do esperado. Para uma análise mais clara, o fator de potência foi submetido a um

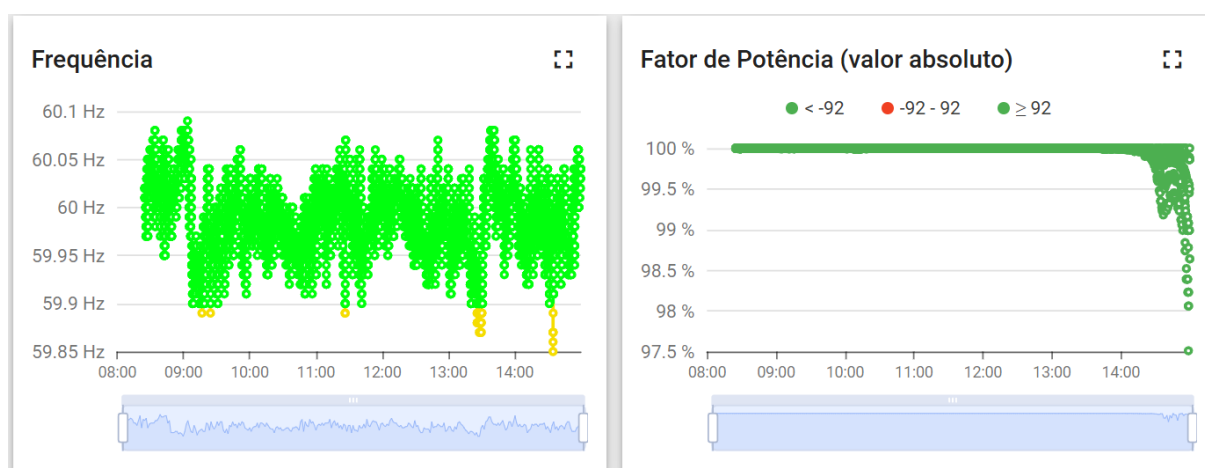
Figura 53 – Energia acumulada entre dias



Fonte: Autoria própria

tratamento dos dados e convertido para valor absoluto, uma vez que o equipamento opera com a potencia reativa baixa oscilando entre capacitiva e indutiva sempre operando com alto fator de potência porem alternando o sinal. Assim, se visualizado em períodos maiores onde há a necessidade de agrupar os valores por média, o fator de potencia parece baixo mesmo sempre estando próximo de 1.

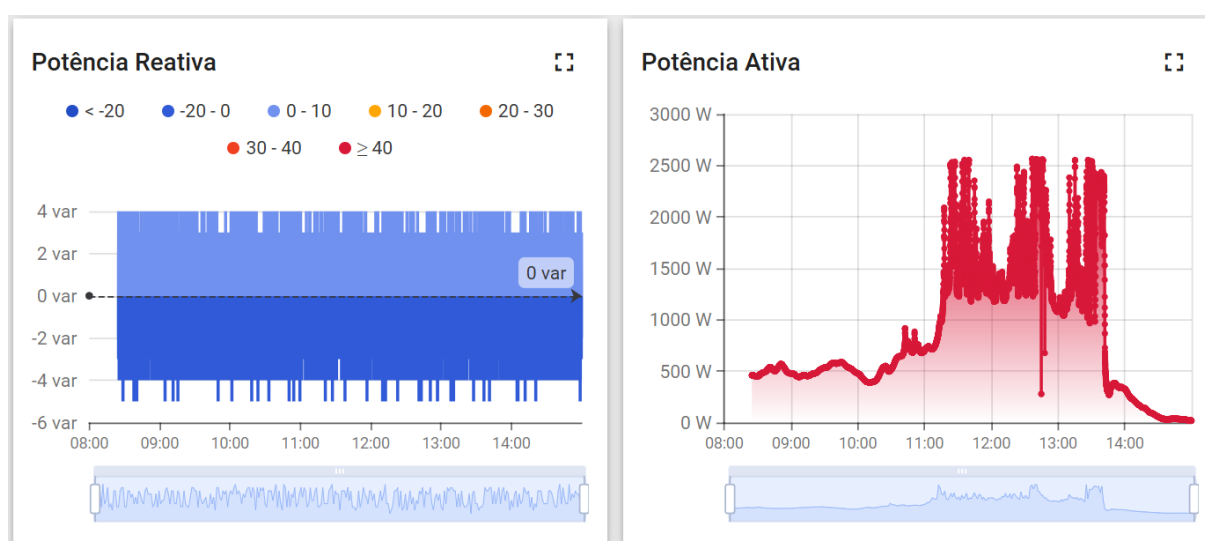
Figura 54 – Qualidade da energia



Fonte: Autoria própria

A Figura 55 apresenta a potência reativa à esquerda, demonstrando que, no intervalo entre -5 e 4 VAR, ela se aproxima de 0, conforme mencionado anteriormente. Em contrapartida, a potência ativa, à direita, opera em um espectro que varia entre 500 e 2500 W durante o período de maior energia.

Figura 55 – Potencia Ativa e Reativa

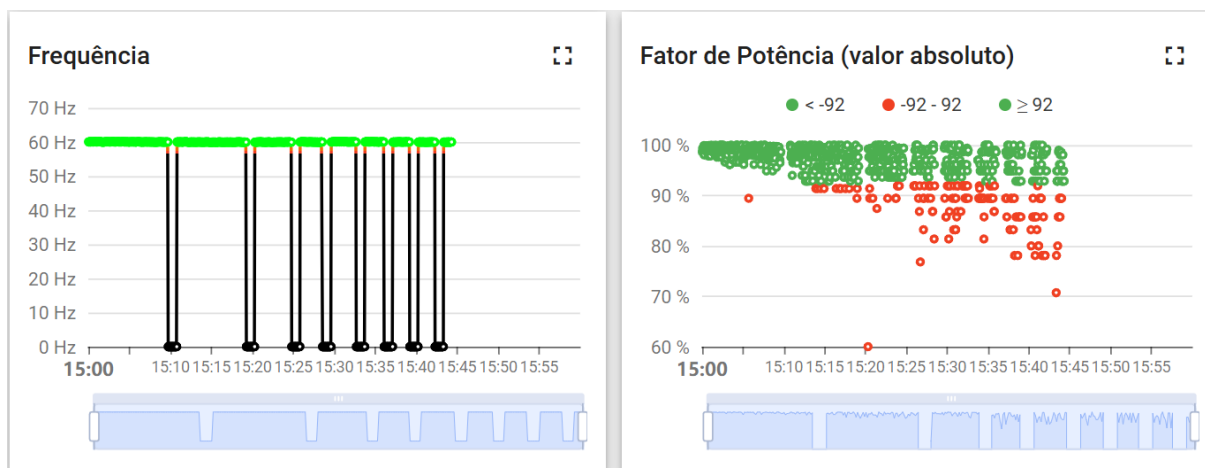


Fonte: Autoria própria

Dessa forma, é possível visualizar nas Figuras 56, 57 e 58 que o método de aquisição opera mesmo nos períodos em que o inversor interrompe o envio de

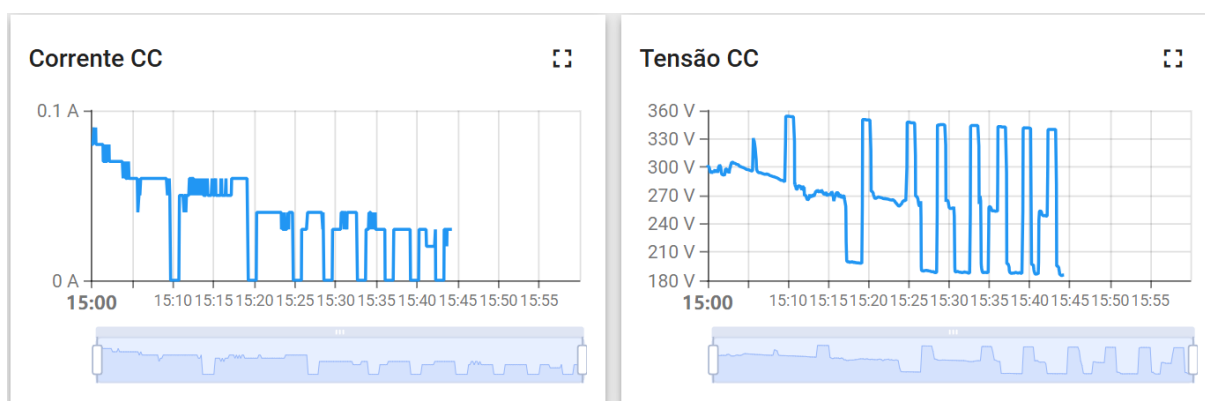
energia. Inicialmente, aparenta que o inversor acumula energia, mas, por se tratar de um inversor *on-grid*, ele não possui bateria. Na prática, a partir de um certo limite, o inversor define que não compensa converter a energia e corta a geração. Dessa forma, por estar sem carga, os painéis conseguem aumentar a tensão e, após o inversor perceber o aumento, volta a operar. Então, a tensão cai por conta da carga e o ciclo se repete.

Figura 56 – Operação com baixa energia gerada



Fonte: Autoria própria

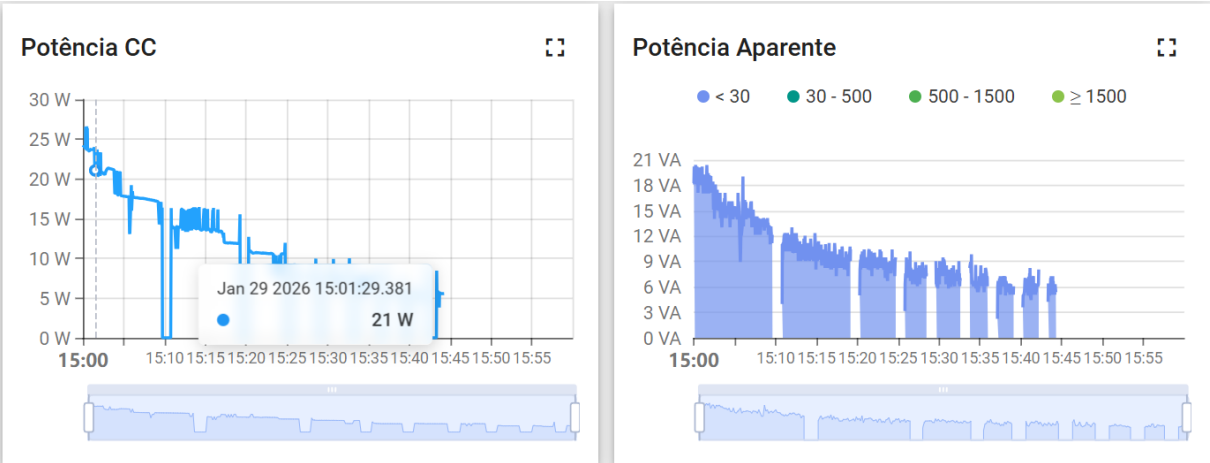
Figura 57 – Operação com baixa energia gerada



Fonte: Autoria própria

Considerando que a tabela que apresenta o maior número de registradores, ou seja, a de ponto flutuante, possui 130 registradores e o limite da comunicação ModBus é de 125 por requisição, conclui-se que a mensagem deve ser dividida em duas. De acordo com os dados obtidos, foi possível apurar que o *delay* do inversor é de 350 milissegundos. Ademais, o tempo para envio de 130 registradores a 9600 de taxa de transmissão é de aproximadamente 270 milissegundos. Dessa forma, o tempo total é de 970 milissegundos. Para uma frequência de aquisição de 1 Hz, foi definida a mesma frequência de aquisição total para ambas as tabelas, tanto para aquelas que precisam ser separadas quanto para aquelas que não precisam.

Figura 58 – Operação com baixa energia gerada 2



Fonte: Autoria própria

5 CONSIDERAÇÕES FINAIS

Este estudo por fim obteve êxito em relação aos objetivos propostos. Foi desenvolvido um método de aquisição de dados e o ambiente de testes foi utilizado para simular a conexão do ESP32-C6 com o inversor. Posteriormente, a comunicação foi aplicada diretamente com o inversor. A placa de circuito impresso foi projetada e confeccionada para melhor confiabilidade da comunicação. O servidor foi configurado com o ThingsBoard para receber os dados via MQTT e gerar os gráficos para visualização da aquisição de dados em tempo real. Esses gráficos estão disponíveis em um *dashboard* que os organiza. Ademais, tal resultado possibilita uma expansão simplificada para futuras empreitadas.

Durante o desenvolvimento do trabalho, o principal desafio enfrentado foi a alocação de memória com segurança, especialmente durante a montagem da *string*. Foi necessária atenção especial para finalizar as variáveis temporárias e garantir o espaço antes da inserção na variável final. Outro ponto a ser considerado é que, nas primeiras leituras, foram utilizadas funções para manipulação de strings. No entanto, os dados frequentemente contêm o caractere 0x00, que, para strings, significa o fim dela. Portanto, para resolver o problema, foi necessário varrer o vetor usando os limitadores da própria requisição de dado, visto que o tamanho é conhecido.

5.1 Sugestões para trabalhos futuros

- Aprimorar do algoritmo de reconfiguração do Wi-Fi, melhorando o desempenho em locais com rede instável;
- Desenvolver de um método para armazenamento de dados para casos em que as quedas de conexão são prolongadas;
- Importar tabelas 7XX permitindo a utilização em equipamentos mais modernos;
- Desenvolver funções para o controle do inversor, com o objetivo de, por exemplo, impedir que a geração supere a demanda contratada. Esse último aspecto é dependente de um algoritmo que identifica todas as tabelas ou da utilização das tabelas 7XX.

REFERÊNCIAS

ANEEL. *Resolução Normativa ANEEL N 956 DE 07/12/2021 - Federal - LegisWeb*. 2021. Disponível em: <https://www.legisweb.com.br/legislacao/?id=424435>. Acesso em: 15 jun. 2026. 48, 66

ANEEL. *Energia solar protagoniza crescimento da matriz elétrica em novembro*. 2025. Disponível em: <https://www.gov.br/aneel/pt-br/assuntos/noticias/2025/energia-solar-protagoniza-crescimento-da-matriz-eletrica-em-novembro>. Acesso em: 10 mar. 2026. 14

BIRD, L. *et al.* Wind and solar energy curtailment: A review of international experience. v. 65, p. 577–586, 2016. ISSN 1364-0321. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1364032116303161>. Acesso em: 17 out. 2025. 14

David Patterson; Andrew Waterman. *Guia prático RISC-V Atlas de uma Arquitetura Aberta*. 1. ed. Strawberry Canyon LLC, 2017. Disponível em: <http://www.riscbook.com/portuguese/>. Acesso em: 13 mar. 2026. 27

ENERGÊS. *Guia Completo do Inversor Solar*. 2020. Disponível em: <https://energes.com.br/inversor-solar/>. Acesso em: 25 mar. 2026. 18, 19

ESPRESSIF. *ESP-IDF*. 2020. Disponível em: <https://www.espressif.com/en/products/sdks/esp-idf>. Acesso em: 15 jun. 2026. 28

ESPRESSIF. *ESP32-C6 Series - Datasheet*. <https://www.espressif.com>, 2023a. Versão 1.5. Disponível em: https://documentation.espressif.com/esp32-c6/_datasheet_en.pdf. Acesso em: 14 mai. 2026. 26

ESPRESSIF. *ESP32-C6-DevKitC-1 - Schematics*. <https://www.espressif.com>, 2024. Versão 1.4. Disponível em: https://dl.espressif.com/dl/schematics/esp32-c6-devkitc-1-schematics_v1.4.pdf. Acesso em: 14 mai. 2026. 55

ESPRESSIF. *ESP32-C6-WROOM-1 ESP32-C6-WROOM-1U - Datasheet*. <https://www.espressif.com>, [2023?]b. Versão 1.4. Disponível em: https://documentation.espressif.com/esp32-c6-wroom-1_wroom-1u_datasheet_en.pdf. Acesso em: 14 mai. 2026. 62

Espressif. *About - ESP32 - — ESP-IDF Programming Guide v6.0.2 documentation*. [s.d.]. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/about.html>. Acesso em: 2 jul. 2026. 28

FRONIUS. *Fronius GEN24 - Modbus TCP and RTU Operating instructions*. [s.d.]. Disponível em: <https://manuals.fronius.com/html/4204102649/en-US.html>. Acesso em: 13 maio 2026. 20, 21, 22, 24, 49, 50

Intel. *O que é Wi-Fi 6?* [202–]. Disponível em: <https://www.intel.com/content/www/br/pt/gaming/resources/wifi-6.html>. Acesso em: 20 mar. 2026. 27

JSY-MK LTD., S. J. T. C. *Differences between RS485, TTL and RS232*. 2024. Disponível em: <https://www.jsypowermeter.com/news/differences-between-rs485-ttl-and-rs232/>. Acesso em: 13 maio 2026. 36

LabSmart. *PGEN – Portal de Gerenciamento de Energia de Instituições Públicas*. [202–]. Disponível em: <https://pgen.app/>. Acesso em: 31 mar. 2026. 15, 16, 17

LEE, J. *ESP-IDF Basics - Lecture 1*. 2025. Section: workshops. Disponível em: <https://developer.espressif.com/workshops/esp-idf-basic/lecture-1/>. Acesso em: 3 jul. 2026. 28, 29

SUNSPEC ALLIANCE. *About*. [s.d.]. Disponível em: <https://sunspec.org/about/>. Acesso em: 8 abr. 2026. 19

ThingsBoard. *Monolithic Architecture*. [s.d.]a. Disponível em: <https://thingsboard.io/docs/reference/architecture/monolithic/>. Acesso em: 14 mai. 2026. 24

ThingsBoard. *thingsboard/tb-postgres - Docker Image*. [s.d.]b. Disponível em: <https://hub.docker.com/r/thingsboard/tb-postgres>. Acesso em: 3 jul. 2026. 45

TRIVEDI, S. *Wi-Fi 6 OFDMA: Resource unit (RU) allocations and mappings*. 2020. Disponível em: <https://blogs.cisco.com/networking/wi-fi-6-ofdma-resource-unit-ru-allocations-and-mappings>. Acesso em: 15 jun. 2026. 27

Witte Software. *Modbus Protocol*. [s.d.]. Disponível em: <https://www.modbustools.com/modbus.html>. Acesso em: 24 mar. 2026. 23

YUAN, M. Getting to know MQTT. 2021. Disponível em: <https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 13 maio 2026. 25