UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Maycon Michel Krüger

Desenvolvimento de Smart Meter para Uso Residencial

Florianópolis
2026

Maycon Michel Krüger

Desenvolvimento de Smart Meter para Uso Residencial

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Engenheiro Eletricista.
Orientador: Prof. Eduardo Augusto Bezerra, Ph.D.

Florianópolis
2026

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Krüger, Maycon
Desenvolvimento de Smart Meter para Uso Residencial / Maycon Krüger ; orientador, Eduardo Bezerra, 2026.
65 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2026.

Inclui referências.

1. Engenharia Elétrica. 2. Smart meter. 3. Sistemas embarcados. 4. Medição de energia. 5. Controle de cargas. I. Bezerra, Eduardo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Maycon Michel Krüger

Título: Desenvolvimento de Smart Meter para Uso Residencial

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 9 de julho de 2026.

Documento assinado digitalmente
ROBERTO FRANCISCO COELHO
Data: 14/07/2026 20:06:04-0300
CPF: ***.034.249-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Roberto Francisco Coelho, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 15/07/2026 12:33:18-0300
CPF: ***.851.577-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Augusto Bezerra, Ph.D.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
RODRIGO VINICIUS MENDONCA PEREIRA
Data: 15/07/2026 11:10:10-0300
CPF: ***.663.779-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Rodrigo Vinícius Mendonça Pereira, Dr.
Membro da Banca
Universidade Federal de Santa Catarina

Documento assinado digitalmente
RICARDO DE ARAUJO ELIAS
Data: 14/07/2026 20:14:13-0300
CPF: ***.386.669-**
Verifique as assinaturas em https://v.ufsc.br

Eng. Eletric. Ricardo de Araujo Elias, Dr.
Membro da Banca
Celesc Distribuição S.A.

AGRADECIMENTOS

Agradeço à minha família, amigos, colegas de curso, professores e todos que, de alguma forma, contribuíram para a realização deste trabalho.

RESUMO

O acesso a informações detalhadas sobre o consumo de energia elétrica é um instrumento eficaz para a promoção da eficiência energética residencial. Este trabalho apresenta o desenvolvimento de um medidor inteligente (*smart meter*) residencial monofásico de custo acessível e código aberto, baseado no microcontrolador ESP32 operando sob o sistema operacional de tempo real NuttX.

As grandezas elétricas (tensão eficaz, corrente eficaz, potência ativa, potência reativa, potência aparente, fator de potência, frequência e energia acumulada) são adquiridas pelo sensor JSY-MK-194G via protocolo Modbus RTU sobre interface UART TTL. O *firmware*, desenvolvido integralmente sobre o NuttX, explora sua conformidade com o padrão POSIX para a implementação de *drivers* e da lógica de aplicação em tempo real, e incorpora o controle de seis relés via MQTT para o acionamento remoto de cargas elétricas.

A transmissão dos dados ao servidor é realizada via Wi-Fi com protocolo MQTT. Para garantir a resiliência da comunicação, as medições são persistidas localmente em memória flash (SPIFFS) e retransmitidas automaticamente após a reconexão. Os dados são armazenados em uma infraestrutura local executada em Raspberry Pi com InfluxDB e visualizados em tempo real por meio do Grafana.

O sistema é validado em instalação residencial real ao longo de 15 dias de operação contínua. A energia acumulada medida apresentou erro de $+1{,}08\%$ em relação ao medidor oficial da distribuidora. A disponibilidade da comunicação MQTT atingiu $97{,}25\%$.

Palavras-chave: Smart meter. Sistemas embarcados. NuttX. ESP32. MQTT. Modbus RTU. Medição de energia. Controle de cargas.

ABSTRACT

Detailed access to residential electricity consumption data is an effective instrument for promoting energy efficiency. This work presents the development of an affordable, open-source single-phase residential smart meter based on the ESP32 microcontroller running the NuttX real-time operating system.

Electrical quantities (RMS voltage, RMS current, active power, reactive power, apparent power, power factor, frequency, and accumulated energy) are acquired through the JSY-MK-194G sensor via Modbus RTU over a UART TTL interface. The firmware, developed entirely on NuttX, leverages its POSIX compliance for driver implementation and real-time application logic, and incorporates control of six relays via MQTT for remote switching of electrical loads.

Data transmission to the server is performed over Wi-Fi using the MQTT protocol. To ensure communication resilience, measurements are buffered locally in SPIFFS flash memory and retransmitted automatically upon reconnection. Data are stored in a local infrastructure running on a Raspberry Pi with InfluxDB and visualized in real time through Grafana.

The system is validated in a real residential installation over 15 days of continuous operation. Accumulated energy measurements showed an error of $+1.08\%$ relative to the utility company's official meter. MQTT communication availability reached 97.25%.

Keywords: Smart meter. Embedded systems. NuttX. ESP32. MQTT. Modbus RTU. Energy metering. Load control.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	Analog-to-Digital Converter (Conversor Analógico-Digital)
ANEEL	Agência Nacional de Energia Elétrica
ANSI	American National Standards Institute
API	Application Programming Interface (Interface de Programação de Aplicações)
CA	Corrente Alternada
CPU	Central Processing Unit (Unidade Central de Processamento)
CRC	Cyclic Redundancy Check (Verificação de Redundância Cíclica)
DAC	Digital-to-Analog Converter (Conversor Digital-Analógico)
DHCP	Dynamic Host Configuration Protocol
ESP32	Espressif Systems ESP32
FC	Function Code (código de função Modbus)
FP	Fator de Potência
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INMETRO	Instituto Nacional de Metrologia, Qualidade e Tecnologia
IoT	Internet of Things (Internet das Coisas)
IP	Internet Protocol
JSON	JavaScript Object Notation
LPWAN	Low-Power Wide-Area Network
MAC	Medium Access Control
MQTT	Message Queuing Telemetry Transport
NSH	NuttShell (shell interativo do NuttX)
OTA	Over-the-Air (atualização remota de firmware)
PLC	Power Line Communication
POSIX	Portable Operating System Interface
PRODIST	Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional
PWM	Pulse Width Modulation (Modulação por Largura de Pulso)
QoS	Quality of Service (Qualidade de Serviço)
RAM	Random Access Memory (Memória de Acesso Aleatório)
REST	Representational State Transfer
RF	Radio Frequency (Radiofrequência)
RMS	Root Mean Square (Valor Eficaz)
ROM	Read-Only Memory (Memória Somente de Leitura)

RS-485	Recommended Standard 485
RTC	Real-Time Clock (Relógio de Tempo Real)
RTOS	Real-Time Operating System (Sistema Operacional de Tempo Real)
RTU	Remote Terminal Unit
SDK	Software Development Kit
SoC	System on a Chip
SPI	Serial Peripheral Interface
SPIFFS	SPI Flash File System
SRAM	Static Random Access Memory (Memória de Acesso Aleatório Estática)
SSID	Service Set Identifier (identificador de rede Wi-Fi)
TC	Transformador de Corrente
TCC	Trabalho de Conclusão de Curso
TCP	Transmission Control Protocol
TLS	Transport Layer Security (Segurança da Camada de Transporte)
TP	Transformador de Potencial
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver-Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
VFS	Virtual File System
WLAN	Wireless Local Area Network
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3
YAML	YAML Ain’t Markup Language

LISTA DE SÍMBOLOS

V_{rms}	Tensão eficaz (V)
I_{rms}	Corrente eficaz (A)
P	Potência ativa (W)
Q	Potência reativa (VAr)
S	Potência aparente (VA)
ϕ	Ângulo de defasagem entre tensão e corrente (rad)
f	Frequência (Hz)
E	Energia elétrica (kWh)
T	Período (s)

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

O acesso a informações detalhadas sobre o consumo de energia elétrica é um dos instrumentos mais eficazes para a promoção da eficiência energética residencial. No Brasil, o setor residencial respondeu por aproximadamente 31,7% do consumo total de energia elétrica do país em 2025, totalizando 179,6 TWh (EPE, 2026), tornando este segmento um alvo prioritário para políticas de conservação de energia. Entretanto, os medidores convencionais instalados nas residências brasileiras fornecem apenas a leitura acumulada de energia ativa, sem possibilitar o monitoramento em tempo real de grandezas como tensão, corrente, fator de potência e frequência.

A modernização do parque de medidores residenciais é uma tendência consolidada internacionalmente, tendo como pilares o monitoramento de múltiplas grandezas elétricas em tempo real, a comunicação de dados por redes digitais e a integração com redes inteligentes (*smart grids*).

Paralelamente ao avanço regulatório, a proliferação de microcontroladores de baixo custo com conectividade sem fio integrada (em especial o ESP32 da Espressif Systems) e de sensores de medição de energia de precisão acessíveis, como os baseados em circuitos integrados de medição de potência, viabilizou o desenvolvimento de dispositivos de medição inteligente (*smart meters*) fora do ecossistema de grandes fabricantes de equipamentos elétricos.

No campo dos sistemas operacionais embarcados, o NuttX RTOS (Real-Time Operating System) destaca-se por oferecer conformidade com o padrão POSIX (o mesmo utilizado em sistemas Linux) em hardware de recursos restritos, como microcontroladores de 32 bits. Isso permite a reutilização de conhecimentos e abstrações de programação do mundo *desktop* no desenvolvimento de aplicações embarcadas, ao mesmo tempo em que oferece garantias de temporização determinística essenciais em sistemas de medição.

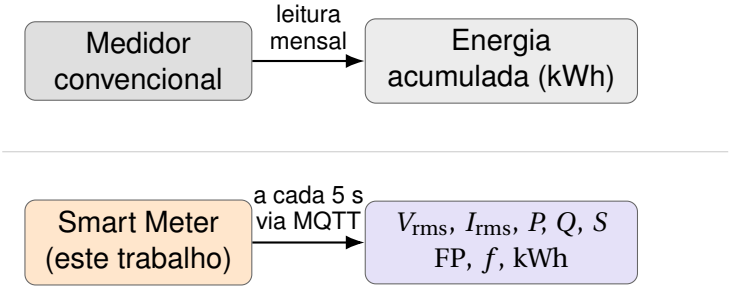
Para a transmissão dos dados coletados, o protocolo MQTT (*Message Queuing Telemetry Transport*) consolidou-se como o padrão predominante para aplicações de Internet das Coisas (IoT) com requisitos de baixo consumo de banda e latência aceitável. O Wi-Fi 802.11, meio físico primário de comunicação, oferece banda e alcance suficientes no ambiente residencial; para situações em que a conectividade esteja temporariamente indisponível, o sistema persiste as medições localmente em memória flash e as retransmite automaticamente após a reconexão.

O presente trabalho integra esses elementos no desenvolvimento de um *smart meter* residencial monofásico de baixo custo e código aberto: sensor de medição de energia JSY-MK-194G, microcontrolador ESP32, sistema operacional NuttX e protocolo MQTT. O código-fonte está disponível em `https://github.com/mayconkruger/`

`nuttx-apps` (*branch* `tcc`).

A Figura 1 ilustra a diferença entre o medidor convencional e o sistema proposto em termos de grandezas disponibilizadas ao usuário.

Figura 1 – Comparação entre o medidor convencional e o sistema de monitoramento proposto



Fonte: elaborado pelo autor.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um sistema de medição inteligente (*smart meter*) para circuitos elétricos residenciais monofásicos, capaz de adquirir grandezas elétricas em tempo real e transmiti-las a um servidor por meio de Wi-Fi com mensageria MQTT, executando sobre o sistema operacional de tempo real NuttX em hardware baseado no microcontrolador ESP32.

1.2.2 Objetivos Específicos

1. Projetar e implementar a aquisição das grandezas elétricas tensão eficaz (V_{rms}), corrente eficaz (I_{rms}), potência ativa (P), potência aparente (S), potência reativa (Q), fator de potência (*FP*), frequência (f) e energia acumulada (*kWh*) do sensor JSY-MK-194G, via protocolo Modbus RTU sobre interface UART TTL;

2. Integrar os recursos de comunicação e controle da plataforma KC868-A6 ao NuttX, utilizando as interfaces UART, I^2C e Wi-Fi disponíveis no sistema operacional para implementar a comunicação com o sensor de medição, o acionamento dos relés e a conectividade de rede;

3. Desenvolver o mecanismo de transmissão e tolerância a falhas de comunicação, empregando Wi-Fi e MQTT para o envio das medições ao servidor e armazenamento local em memória flash (SPIFFS) para retransmissão automática em caso de indisponibilidade de rede;

4. Integrar o *smart meter* a uma infraestrutura local de armazenamento e visualização de dados, baseada em Raspberry Pi e Docker Compose, permitindo o monitoramento contínuo e a análise histórica das grandezas medidas;

5. Avaliar experimentalmente o funcionamento do sistema em ambiente residencial real, comparando as medições de energia obtidas com o medidor oficial da distribuidora e analisando a confiabilidade da comunicação ao longo do período de operação.

1.3 JUSTIFICATIVA

A relevância deste trabalho fundamenta-se em dois eixos principais:

Eficiência energética: Estudos demonstram que consumidores residenciais com acesso a informações detalhadas sobre seu perfil de consumo reduzem o gasto energético entre 5% e 15% em média (Darby, 2006). O monitoramento contínuo de grandezas como fator de potência e corrente de pico também possibilita a identificação precoce de equipamentos defeituosos ou de baixo desempenho, contribuindo para a redução de perdas e o aumento da vida útil das instalações.

Sistemas embarcados e IoT: O desenvolvimento de um *smart meter* sobre um RTOS de conformidade POSIX em hardware de prateleira constitui um exercício prático abrangente das disciplinas de sistemas embarcados: desenvolvimento de *drivers*, escalonamento de tarefas em tempo real, protocolos de comunicação serial e sem fio, e integração com serviços de coleta e visualização de dados. A escolha do NuttX, em particular, expõe o desenvolvedor a conceitos avançados de programação de sistemas que extrapolam a maioria das implementações baseadas em *bare-metal* ou em bibliotecas de alto nível como o Arduino.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em quatro capítulos, conforme descrito a seguir.

O Capítulo 2 apresenta o referencial teórico necessário à compreensão do sistema desenvolvido, abordando: fundamentos da medição de energia elétrica em circuitos monofásicos; conceitos de *smart meters*; sistemas operacionais de tempo real com ênfase no NuttX; o protocolo MQTT; e uma revisão dos trabalhos relacionados.

O Capítulo 3 descreve o processo de desenvolvimento do sistema, contemplando a seleção e caracterização dos componentes de hardware, a configuração do ambiente NuttX, o desenvolvimento dos *drivers* de baixo nível, a implementação da camada de aplicação e a integração com a infraestrutura local de coleta e visualização.

O Capítulo 4 apresenta os resultados experimentais obtidos em instalação residencial real, com análise quantitativa da precisão das medições de energia e da

confiabilidade das comunicações, seguida das conclusões gerais do trabalho e das sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta os fundamentos teóricos que embasam o desenvolvimento do *smart meter* proposto neste trabalho. São abordados os conceitos de medição de energia elétrica em circuitos monofásicos, a arquitetura de medidores inteligentes, o sistema operacional de tempo real NuttX, a tecnologia de comunicação sem fio Wi-Fi, o protocolo MQTT, e uma revisão dos trabalhos relacionados.

2.1 MEDIÇÃO DE ENERGIA ELÉTRICA EM CIRCUITOS MONOFÁSICOS

A medição precisa das grandezas elétricas de um circuito de corrente alternada (CA) monofásico é a base de qualquer sistema de monitoramento de energia. As principais grandezas de interesse são descritas a seguir.

2.1.1 Tensão e Corrente Eficazes (RMS)

Em um circuito CA, tanto a tensão quanto a corrente variam sinusoidalmente no tempo. O valor eficaz (*Root Mean Square*, RMS) de uma grandeza periódica $x(t)$ de período T é definido como:

$$X_{\text{rms}} = \sqrt{\frac{1}{T}\int_0^T x^2(t)\,dt} \tag{1}$$

Para uma tensão senoidal pura $v(t) = V_p \cos(\omega t)$, o valor RMS é $V_{\text{rms}} = V_p/\sqrt{2}$. Na rede elétrica brasileira, a tensão nominal de distribuição residencial monofásica é de 127 V ou 220 V (RMS), com frequência nominal de 60 Hz. O valor RMS de tensão e corrente é a grandeza diretamente utilizada para o cálculo de potência e para a faturação de energia.

2.1.2 Potências Elétricas: Ativa, Reativa e Aparente

Em circuitos CA com cargas reativas (indutivas ou capacitivas), definem-se três grandezas de potência distintas (Chapman, 2013):

Potência ativa (P), expressa em watts (W), representa o trabalho efetivo realizado pela carga (calor, luz, movimento mecânico) e é a grandeza faturada pelas distribuidoras:

$$P = V_{\text{rms}} \cdot I_{\text{rms}} \cdot \cos\phi \tag{2}$$

onde ϕ é o ângulo de defasagem entre tensão e corrente.

Potência reativa (Q), expressa em volt-ampère reativo (VAr), representa a energia armazenada e devolvida por elementos reativos (indutores e capacitores), sem realizar trabalho útil:

$$Q = V_{\text{rms}} \cdot I_{\text{rms}} \cdot \sin\phi \tag{3}$$

Potência aparente (S), expressa em volt-ampère (VA), é a magnitude do vetor complexo formado por P e Q:

$$S = V_{\text{rms}} \cdot I_{\text{rms}} = \sqrt{P^2 + Q^2} \tag{4}$$

O triângulo de potências expresso pelas equações acima evidencia que cargas com baixo fator de potência exigem maior corrente da rede para a mesma potência ativa entregue, aumentando as perdas na distribuição.

2.1.3 Fator de Potência

O fator de potência (*FP*) é a razão entre potência ativa e aparente:

$$FP = \frac{P}{S} = \cos\phi \tag{5}$$

O *FP* varia entre 0 e 1 (ou entre -1 e $+1$ quando se considera o sinal). Para cargas puramente resistivas, $FP = 1$; para cargas puramente reativas, $FP = 0$. No contexto residencial, cargas como motores de compressores de ar-condicionado e refrigeradores apresentam fator de potência tipicamente entre 0,7 e 0,9 (Chapman, 2013). Valores baixos de *FP* implicam aumento da corrente demandada da rede, podendo ocasionar sobrecarga de condutores e transformadores.

2.1.4 Energia Elétrica Acumulada

A energia elétrica ativa acumulada, expressa em quilowatt-hora (kWh), é a grandeza utilizada para a faturação do consumo residencial. É obtida pela integração da potência ativa ao longo do tempo:

$$E = \int_{t_1}^{t_2} P(t)\, dt \tag{6}$$

Em implementações práticas de medição digital, essa integral é aproximada por somas de Riemann sobre intervalos de amostragem Δt:

$$E \approx \sum_{k} P_k \cdot \Delta t \tag{7}$$

2.1.5 Técnicas de Medição de Grandezas Elétricas CA

Diversas abordagens são utilizadas para a medição de corrente e tensão em circuitos CA de potência (Balbinot; Brusamarello, 2010):

Resistores shunt: A corrente é medida pela queda de tensão em um resistor de baixo valor inserido em série com a carga. Apresenta alta precisão e linearidade, porém introduz perdas por condução e está em contato galvânico com o circuito monitorado.

Transformadores de corrente (TC): Oferecem isolamento galvânico e são amplamente utilizados em medidores de energia comerciais. O TC de núcleo fechado requer interrupção do circuito para instalação, enquanto o TC de núcleo dividido (*split-core*) permite instalação sem desligamento.

Transformadores de potencial (TP): São o equivalente dos TCs para a medição de tensão. Reduzem a tensão da rede a um nível seguro e compatível com o circuito de aquisição, mantendo o isolamento galvânico. Em sistemas de baixa tensão residencial, divisores resistivos são frequentemente empregados como alternativa mais simples, porém sem isolamento galvânico.

Sensores de efeito Hall: Medem o campo magnético gerado pela corrente, oferecendo isolamento galvânico e resposta a sinais CC e CA. Apresentam maior custo e complexidade de sinal de saída.

Circuitos integrados de medição de energia: Soluções como o ADE7758 (Analog Devices), CS5490 (Cirrus Logic) e os circuitos embarcados no sensor JSY-MK-194G integram todos os blocos de condicionamento de sinal, conversão analógico-digital (ADC) de alta resolução e processamento digital, entregando as grandezas já calculadas via interface de comunicação digital (SPI, UART ou RS-485). Essa abordagem simplifica o hardware da aplicação e melhora a repetibilidade das medições.

2.2 MEDIDORES INTELIGENTES (*SMART METERS*)

2.2.1 Definição e Evolução

Um medidor inteligente, ou *smart meter*, é um dispositivo eletrônico capaz de registrar o consumo de energia elétrica em intervalos configuráveis e comunicar essas informações por meio de uma rede de comunicação (Gungor et al., 2011). Diferentemente dos medidores eletromecânicos convencionais, que utilizam um disco de alumínio girando sob a influência de campos eletromagnéticos, os *smart meters* empregam circuitos eletrônicos de medição digital e interfaces de comunicação padronizadas.

A evolução dos medidores residenciais pode ser sintetizada em três gerações (Depuru; Wang; Devabhaktuni, 2011):

1. **Medidores eletromecânicos (Geração 1):** Operam por indução eletromagnética. Medem apenas energia ativa acumulada. Leitura manual periódica.

2. **Medidores eletrônicos estáticos (Geração 2):** Utilizam circuitos eletrônicos para medição, com display digital. Podem registrar múltiplos postos tarifários. Leitura ainda predominantemente manual ou por *modem* de baixa velocidade.

3. **Medidores inteligentes (Geração 3, *Smart Meters*):** Comunicação bidirecional em tempo real, medição de múltiplas grandezas (tensão, corrente, potência ativa e reativa, fator de potência, qualidade de energia), suporte a tarifação dinâmica, detecção de fraudes e integração com redes *smart grid*.

2.2.2 Regulamentação no Brasil

No Brasil, as regras de prestação do serviço público de distribuição de energia elétrica são estabelecidas pela Resolução Normativa ANEEL nº 1.000/2021 (ANEEL, 2021). A norma técnica IEC 62053-21 (IEC, 2020) especifica os requisitos de desempenho metrológico para medidores estáticos de energia elétrica ativa em corrente alternada, definindo as classes de exatidão 0,5, 1 e 2 (com erros máximos de 0,5%, 1,0% e 2,0%, respectivamente).

O INMETRO regulamenta os requisitos metrológicos para aprovação de modelo e verificação de medidores de energia elétrica ativa por meio da Portaria nº 493/2021 (INMETRO, 2021), que define os critérios de exatidão e ensaios de conformidade aplicáveis aos equipamentos utilizados em transações comerciais.

2.3 SENSOR DE MEDIÇÃO JSY-MK-194G

O JSY-MK-194G é um módulo de medição de energia elétrica monofásica bidirecional fabricado pela Shenzhen Jiansiyan Technology Co., Ltd., projetado para circuitos de corrente alternada com tensão de entrada de 1 a 300 V CA (50/60 Hz) e faixa de corrente de 20 mA a 100 A. O módulo possui dois canais de corrente independentes (CT1 e CT2), cada um com seu próprio transformador de corrente do tipo *split-core*, compartilhando um único canal de tensão medido por um transformador de potencial (TP) interno. Internamente, o sinal de cada TC e do TP é condicionado e digitalizado por um chip de medição dedicado com conversor analógico-digital de 24 bits, que calcula as grandezas e as disponibiliza ao sistema hospedeiro via interface UART TTL com protocolo Modbus RTU.

O módulo apresenta exatidão de $\pm 1{,}0\%$ para tensão, corrente, potência ativa e energia (Shenzhen Jiansiyan Technology Co., Ltd., 2023), taxa de atualização padrão de 330 ms, isolação dielétrica de 3000 VDC entre o circuito monitorado e a alimentação auxiliar, e deriva de temperatura de $\leq$100 ppm/°C. A alimentação auxiliar aceita 3,3 V a 5 V CC com consumo de 10 mA. As dimensões físicas do módulo são 65×48×28 mm.

2.3.1 Grandezas Medidas e Registradores Modbus

O JSY-MK-194G disponibiliza as seguintes grandezas via leitura de registradores Modbus (função 0x03, *Read Holding Registers*), separadas por canal:

- Tensão eficaz (V_{rms}): compartilhada entre os dois canais;

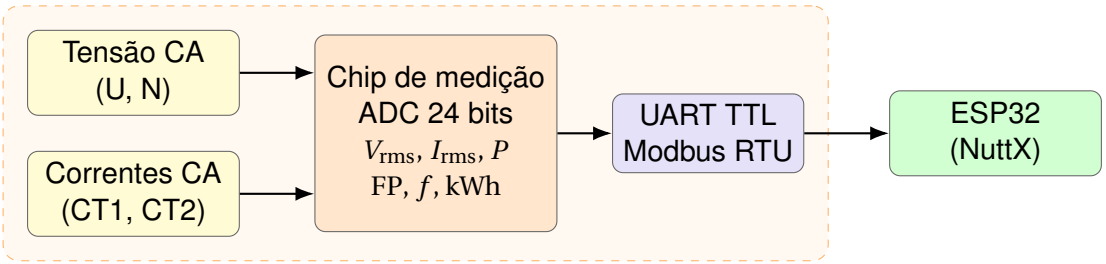
- Corrente eficaz (I_{rms}): independente por canal;
- Potência ativa (P): independente por canal;
- Fator de potência (*FP*): independente por canal;
- Frequência (f): resolução de 0,01 Hz, compartilhada;
- Energia acumulada (kWh): separada em energia importada (positiva) e exportada (negativa) por canal;
- Sentido de fluxo de potência: registrador por canal indicando importação (0x00) ou exportação (0x01).

Potência aparente (S) e potência reativa (Q) não são fornecidas diretamente pelo sensor: são calculadas pelo *firmware* a partir de $S = V \cdot I$ e $Q = \sqrt{S^2 - P^2}$.

A interface UART TTL opera com as configurações padrão de 4800 bps, 8 bits de dados, sem paridade e 1 bit de parada (8N1). O endereço Modbus padrão de fábrica é 0x01, podendo ser configurado de 1 a 255. A taxa de atualização dos registradores é de 330 ms por padrão, que representa a frequência de *saída* dos valores calculados nos registradores Modbus, e não a frequência de amostragem interna do ADC. O chip de medição emprega conversão sigma-delta de 24 bits e opera internamente a frequências da ordem de quilohertz para capturar fielmente os ciclos da rede CA em 60 Hz; contudo, a frequência exata de amostragem interna não é divulgada pelo fabricante no *datasheet* do módulo (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

A Figura 2 apresenta o diagrama funcional do módulo, desde as entradas analógicas de tensão e corrente até a saída digital via UART.

Figura 2 – Diagrama funcional do módulo JSY-MK-194G



Fonte: elaborado pelo autor.

2.3.2 Protocolo Modbus RTU

O Modbus RTU (*Remote Terminal Unit*) é um protocolo de comunicação serial mestre-escravo amplamente utilizado em automação industrial e instrumentação. No

modo RTU, os dados são transmitidos em formato binário compacto, com cada mensagem delimitada por intervalos de silêncio no barramento (mínimo 3,5 tempos de caractere).

Uma requisição de leitura de registradores (função 0x03) possui a seguinte estrutura:

Tabela 1 – Estrutura de uma requisição Modbus RTU, função 0x03

End.	**FC**	**Ini. Hi**	**Ini. Lo**	**Qtd. Hi**	**Qtd. Lo**	**CRC Lo**	**CRC Hi**
1 byte	0x03	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

Fonte: elaborado pelo autor.

Os 2 bytes de CRC-16, transmitidos ao final, são calculados sobre todos os bytes anteriores e garantem a integridade dos dados.

2.4 PLATAFORMA DE HARDWARE KC868-A6

A plataforma KC868-A6, desenvolvida pela empresa KINCONY, é uma placa de desenvolvimento baseada no módulo ESP32 da Espressif Systems. O ESP32 é um *System-on-Chip* (SoC) de duplo núcleo Xtensa LX6 a 240 MHz, com 520 KB de SRAM, Wi-Fi 802.11 b/g/n integrado, Bluetooth 4.2 e ampla variedade de periféricos.

A placa KC868-A6 contém os seguintes circuitos relevantes para este projeto, identificados no esquemático:

- **Interface SPI:** Barramento SPI disponível em conector de expansão, conectado ao controlador SPI do ESP32 (HSPI ou VSPI);
- **Seis relés de comutação:** A placa dispõe de seis relés eletromecânicos de uso geral, cada um acionado por um transistor NPN dedicado. As bases dos transistores são controladas pelos pinos de saída de um expansor de I/O PCF8574 (Texas Instruments, 2024) conectado ao barramento I^2C do ESP32. A lógica de acionamento é ativa em nível baixo (*active-LOW*): o transistor conduz e energiza a bobina do relé quando o bit correspondente no PCF8574 está em nível lógico 0;
- **Barramento I^2C:** Compartilhado entre o expansor de saídas PCF8574 (endereço `0x24`), um segundo PCF8574 de entradas digitais (endereço `0x22`) e um RTC DS1307 (endereço `0x68`);
- **Conversor USB-UART com AutoProg:** Circuito de programação automática via DTR/RTS do conversor USB-serial, dispensando a necessidade de pressionar botão BOOT/EN durante a gravação;
- **Alimentação:** Regulador de tensão 5 V e 3,3 V com proteção de inversão de polaridade.

2.5 SISTEMAS OPERACIONAIS DE TEMPO REAL (RTOS)

2.5.1 Conceitos Fundamentais

Um sistema operacional de tempo real (RTOS) é um sistema operacional projetado para atender a restrições temporais determinísticas, garantindo que tarefas críticas sejam executadas dentro de prazos (*deadlines*) estabelecidos (Liu, 2000). Distingue-se dos sistemas operacionais de uso geral (GPOS, *General Purpose Operating Systems*) pelo compromisso com a previsibilidade temporal em detrimento do *throughput* médio.

Os conceitos centrais de um RTOS incluem:

Tarefas (*tasks*): Unidades de execução independentes, cada uma com seu próprio contexto de CPU (registradores, pilha) e estado (pronta, executando, bloqueada, suspensa). Em sistemas POSIX, são mapeadas para *threads*.

Escalonador (*scheduler*): Algoritmo que decide qual tarefa ocupa a CPU a cada instante. O escalonamento preemptivo com prioridades fixas (*Rate Monotonic Scheduling*, RMS) é o mais utilizado em RTOS hard real-time, enquanto o escalonamento por *deadline* mais próximo (*Earliest Deadline First*, EDF) é teoricamente ótimo para sistemas de utilização variável (Liu; Layland, 1973).

Mecanismos de sincronização e comunicação: Semáforos, mutexes, filas de mensagens (*message queues*) e variáveis de condição são utilizados para coordenar o acesso a recursos compartilhados e para a troca de dados entre tarefas.

Inversão de prioridade: Problema que ocorre quando uma tarefa de alta prioridade aguarda um recurso ocupado por uma de baixa prioridade, enquanto tarefas de prioridade intermediária executam livremente. Protocolos como *Priority Inheritance* e *Priority Ceiling* mitigam esse problema.

2.5.2 NuttX RTOS

O NuttX é um RTOS de código aberto licenciado sob Apache License 2.0, com foco em conformidade com os padrões POSIX e ANSI em microcontroladores de 8 a 64 bits (Apache Software Foundation, 2024). O projeto é mantido pela Apache Software Foundation desde 2019.

As principais características do NuttX relevantes para este trabalho são:

Conformidade POSIX: O NuttX implementa um subconjunto substancial da API POSIX, incluindo *pthreads*, semáforos POSIX, filas de mensagens POSIX (`mqueue`), sinais e o sistema de arquivos VFS (*Virtual File System*). Isso permite que *drivers* e aplicações sejam desenvolvidos com as mesmas chamadas de sistema utilizadas em Linux, facilitando a portabilidade e desenvolvimento.

Modelo de *driver* orientado a arquivo: Assim como no Linux, todos os *drivers* de dispositivo no NuttX são acessados por meio de chamadas de sistema padrão (`open`, `read`, `write`, `ioctl`, `close`) sobre nós no sistema de arquivos `/dev`. Isso impõe uma

interface uniforme que simplifica o desenvolvimento de aplicações portáveis.

Arquitetura monolítica plana (*flat build*): No modo de construção padrão, o núcleo e as aplicações compartilham o mesmo espaço de endereçamento, o que elimina a sobrecarga de mudança de contexto entre espaço de usuário e kernel presente em sistemas com MMU. Também existe um modo protegido com separação de privilégios, disponível em processadores com MPU.

Suporte ao ESP32: A Espressif Systems mantém um *port* oficial do NuttX para a família ESP32 (Espressif Systems, 2024), com suporte a Wi-Fi, Bluetooth e todos os periféricos de hardware (UART, SPI, I2C, ADC, DAC, PWM).

Sistema de configuração Kconfig: Herdado do kernel Linux, o sistema Kconfig permite a configuração modular de todas as funcionalidades do sistema operacional antes da compilação, gerando um binário mínimo ajustado às necessidades da aplicação.

2.5.3 Comparação com Outros RTOS

A Tabela 2 apresenta uma comparação entre os principais RTOS de código aberto considerados para este projeto:

Tabela 2 – Comparação entre RTOS de código aberto considerados para ESP32

Característica	**NuttX**	**FreeRTOS**	**Zephyr**
Conformidade POSIX	Alta	Parcial	Parcial
Suporte ESP32	Oficial	Oficial (ESP-IDF)	Experimental
Modelo de *driver*	VFS/`/dev`	Ad-hoc	Device Tree
Licença	Apache 2.0	MIT	Apache 2.0
Footprint mínimo	~50 KB	~5 KB	~8 KB
Wi-Fi ESP32 nativo	Sim	Sim	Não

Fonte: elaborado pelo autor com base em (Apache Software Foundation, 2024; Espressif Systems, 2024; Linux Foundation, 2024).

O FreeRTOS, embora seja o RTOS mais amplamente utilizado em microcontroladores e tenha suporte oficial da Espressif via ESP-IDF, não oferece conformidade POSIX nem o modelo de *driver* baseado em VFS. O Zephyr, apesar de sua crescente adoção, ainda não possui suporte estável para Wi-Fi no ESP32. O NuttX foi escolhido por combinar suporte ESP32 maduro, conformidade POSIX e o modelo de *driver* orientado a arquivo, que alinha-se com o objetivo pedagógico e técnico deste trabalho.

2.6 COMUNICAÇÃO SEM FIO

2.6.1 Wi-Fi 802.11

O padrão IEEE 802.11 define as especificações da camada física e de controle de acesso ao meio (MAC) para redes locais sem fio (WLAN). O ESP32 suporta os

modos b (1–11 Mbps a 2,4 GHz), g (até 54 Mbps a 2,4 GHz) e n (até 150 Mbps a 2,4 GHz em fluxo único) (Espressif Systems, 2023).

No contexto residencial, o Wi-Fi é o meio de comunicação primário escolhido por três razões: (i) infraestrutura já existente na maioria das residências brasileiras; (ii) largura de banda mais que suficiente para os payloads de telemetria gerados pelo *smart meter* (< 1 kB por amostra); e (iii) suporte nativo maduro no NuttX para a família ESP32.

2.7 PROTOCOLO MQTT

O MQTT (*Message Queuing Telemetry Transport*) é um protocolo de mensageria leve, baseado no modelo *publish/subscribe*, projetado para ambientes com largura de banda reduzida, alta latência ou conectividade instável (Banks et al., 2019). Foi padronizado pela OASIS como MQTT 3.1.1 em outubro de 2014 e MQTT 5.0 em março de 2019.

2.7.1 Modelo *Publish/Subscribe*

No modelo *publish/subscribe*, os produtores de dados (*publishers*) e consumidores (*subscribers*) são desacoplados espacial e temporalmente por meio de um intermediário denominado *broker*. O *publisher* envia mensagens a um tópico (*topic*), sem conhecer quais clientes irão recebê-las. Os *subscribers* declaram interesse em tópicos específicos, podendo utilizar curingas (+ para um nível, # para múltiplos níveis).

Neste projeto, a estrutura de tópicos adotada é:

```
smartmeter/<device_id>/telemetry   → telemetria JSON (uplink periódico)
smartmeter/<device_id>/relay/set   → comando de relé JSON (downlink)
```

2.7.2 Qualidade de Serviço (QoS)

O MQTT define três níveis de qualidade de serviço (Banks et al., 2019):

- **QoS 0 (*At most once*):** A mensagem é entregue no máximo uma vez, sem confirmação. Adequado para telemetria periódica onde a perda ocasional de amostras é aceitável;
- **QoS 1 (*At least once*):** A mensagem é entregue pelo menos uma vez, com confirmação (`PUBACK`). Pode resultar em duplicatas em caso de retransmissão;
- **QoS 2 (*Exactly once*):** Garante exatamente uma entrega, por meio de um protocolo de *handshake* de quatro vias. Maior sobrecarga de comunicação.

Para a telemetria de energia, QoS 1 é o nível mais adequado: garante que nenhuma amostra seja perdida sem a sobrecarga do QoS 2, enquanto o sistema de armazenamento no servidor pode descartar duplicatas com base em *timestamp*.

2.7.3 Formato de Payload

O MQTT não define o formato do *payload* das mensagens, deixando essa escolha para a aplicação. Neste projeto, utiliza-se JSON (*JavaScript Object Notation*) por sua legibilidade, ampla compatibilidade com ferramentas de IoT e suporte nativo em plataformas de armazenamento e visualização de dados. Um exemplo de *payload* de telemetria é apresentado a seguir:

```
{
  "device_id": "smartmeter_01",
  "ts": 1746000000,
  "voltage_v": 220.5,
  "current_a": 3.42,
  "active_power_w": 720,
  "reactive_power_var": 210,
  "apparent_power_va": 754,
  "power_factor": 0.955,
  "frequency_hz": 60.01,
  "energy_kwh": 125.43
}
```

2.8 TRABALHOS RELACIONADOS

Diversos trabalhos na literatura abordam o desenvolvimento de sistemas de monitoramento de energia elétrica baseados em microcontroladores e comunicação sem fio. Esta seção apresenta os mais relevantes ao contexto deste projeto e suas limitações em relação à proposta aqui desenvolvida.

Depuru, Wang e Devabhaktuni (2011) apresentam uma revisão abrangente dos desafios, vantagens e estado da arte dos medidores inteligentes para redes elétricas, discutindo tecnologias de comunicação (ZigBee, PLC, RF) e requisitos de segurança. O trabalho evidencia a relevância do tema, mas não propõe uma implementação específica nem aborda o uso de RTOS ou armazenamento local de dados.

Silva (2018) desenvolveram um sistema de monitoramento de energia residencial baseado em ESP8266 com transmissão MQTT, utilizando o sensor PZEM-004T. O trabalho validou a integração MQTT-ESP com *brokers* públicos, contudo operou em modo *bare-metal* sobre o SDK da Espressif, sem as garantias de temporização de um RTOS.

Enriko, Abidin e Noor (2021) propuseram um sistema de medição inteligente com comunicação LoRaWAN voltado à eletrificação rural, validando o uso de redes LPWAN de longo alcance para a telemetria de energia em regiões de baixa cobertura celular. Apesar de demonstrar a viabilidade da tecnologia LoRaWAN para esse cenário, o trabalho não abordou a persistência local de dados em caso de indisponibilidade do canal de comunicação, nem utilizou um RTOS com conformidade POSIX.

Em contraste com os trabalhos citados, o presente projeto propõe a integração de: (i) medição precisa via sensor dedicado com interface Modbus; (ii) RTOS de conformidade POSIX (NuttX) sobre ESP32; (iii) armazenamento local em memória *flash* (SPIFFS) com retransmissão automática para garantia de entrega mesmo durante quedas de conectividade Wi-Fi. Essa combinação representa um avanço em relação ao estado da arte identificado, particularmente no aspecto da robustez de comunicação e da qualidade do ambiente de desenvolvimento embarcado.

A Tabela 3 resume as principais características dos trabalhos citados em relação à proposta deste TCC:

Tabela 3 – Comparação entre trabalhos relacionados e este projeto

Característica	**Depuru, Wang e Devabhaktuni (2011)**	**Silva (2018)**	**Enriko, Abidin e Noor (2021)**	**Este trabalho**
Microcontrolador	N/E	ESP8266	N/E	ESP32
RTOS	Não	Não	Não	NuttX (POSIX)
Sensor de medição	N/E	PZEM-004T	N/E	JSY-MK-194G
Protocolo medição	N/E	UART	N/E	Modbus RTU/UART TTL
Comunicação	ZigBee/PLC/RF	Wi-Fi/MQTT	LoRaWAN	Wi-Fi/MQTT
Armazenamento local	Não	Não	Não	Flash NV

Fonte: elaborado pelo autor.

3 DESENVOLVIMENTO

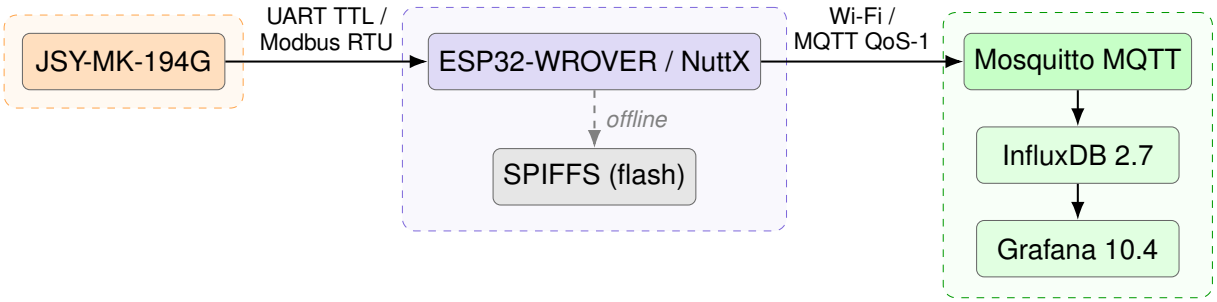
Este capítulo descreve o processo de desenvolvimento do *smart meter* residencial proposto, abrangendo a caracterização detalhada dos componentes de hardware, o diagrama de interconexão do sistema, a configuração do ambiente NuttX, o desenvolvimento dos *drivers* de baixo nível e da camada de aplicação, e a integração com a infraestrutura local de coleta e visualização.

3.1 ARQUITETURA DO SISTEMA

A Figura 3 ilustra a arquitetura geral do *smart meter* desenvolvido. O sistema é composto por três subsistemas funcionais principais:

- **Subsistema de aquisição:** O sensor JSY-MK-194G mede as grandezas elétricas do circuito monofásico e as disponibiliza ao ESP32 via protocolo Modbus RTU sobre UART TTL. O ESP32 atua como mestre Modbus, realizando consultas periódicas ao sensor.
- **Subsistema de processamento:** O NuttX RTOS executa no ESP32, orquestrando as tarefas de leitura do sensor, formatação dos dados e transmissão. Em caso de indisponibilidade da rede, as medições são persistidas localmente em flash e retransmitidas automaticamente.
- **Subsistema de comunicação:** O Wi-Fi integrado ao ESP32 envia os dados ao *broker* MQTT no servidor local (Raspberry Pi). Quando a conectividade está indisponível, as amostras são armazenadas em SPIFFS e reenviadas após a reconexão.

Figura 3 – Arquitetura geral do *smart meter*: subsistema de aquisição (laranja), processamento KC868-A6/NuttX (azul) e comunicação/servidor local (verde)



Fonte: elaborado pelo autor.

3.2 CARACTERIZAÇÃO DO HARDWARE

3.2.1 Plataforma KC868-A6: Análise do Esquemático

A análise do esquemático da placa KC868-A6 (versão 1.3) permitiu identificar as conexões relevantes para este projeto. O microcontrolador principal é o módulo ESP32 com 4 MB de flash SPI e 520 KB de SRAM. As interfaces de interesse e suas conexões ao ESP32 são descritas abaixo.

3.2.1.1 Interface UART1 (GPIO12/GPIO13)

A placa KC868-A6 disponibiliza a UART1 do ESP32 nos pinos GPIO12 (TX) e GPIO13 (RX). O sensor JSY-MK-194G opera com interface TTL e é conectado diretamente a esses pinos.

3.2.1.2 Circuito de Programação (AutoProg)

A placa KC868-A6 inclui um circuito *AutoProg* baseado em transistores BJT que converte os sinais DTR e RTS do conversor USB-UART nos sinais EN e GPIO0 do ESP32, implementando o protocolo de entrada em modo de *download* automaticamente, sem necessidade de pressionar o botão BOOT durante a programação. O conversor USB-UART utilizado é compatível com CH340 ou CP2102.

3.2.1.3 Alimentação

A KC868-A6 é alimentada por fonte externa de 12 VDC ou por USB-C (5 V). O esquemático revela reguladores lineares que fornecem:

- 3,3 V: para o ESP32, JSY-MK-194G e demais periféricos;
- 5 V: disponível nos terminais de expansão e relés.

A tensão de operação de 3,3 V é compatível com o JSY-MK-194G, dispensando conversores de nível lógico.

3.2.2 Sensor JSY-MK-194G: Especificações e Interface

3.2.2.1 Especificações Elétricas

A Tabela 4 apresenta as especificações elétricas do sensor JSY-MK-194G extraídas do manual do fabricante.

3.2.2.2 Pinagem e Conexão

O JSY-MK-194G expõe quatro pinos de interface elétrica e dois conectores para os transformadores de corrente:

Tabela 4 – Especificações elétricas do JSY-MK-194G

Parâmetro	**Valor**
Tensão de entrada (CA)	1 V – 300 V rms
Corrente de entrada	20 mA – 100 A rms
Faixa de frequência	45 Hz – 65 Hz
Exatidão (tensão, corrente)	±0,5% F.S.
Exatidão (potência, energia)	±1,0%
Resolução de tensão	0,01 V
Resolução de corrente	0,01 A
Resolução de potência	0,1 W
Resolução de energia	0,01 kWh
Tensão de alimentação	3,3 V ou 5 V DC
Interface de comunicação	UART TTL (3,3 V / 5 V)
Protocolo	Modbus RTU
Taxa de comunicação	4800 bps (padrão); 1200 a 38400 bps (configurável)
Formato de dados	8N1 (8 bits, sem paridade, 1 stop bit)
Endereço Modbus padrão	0x01
Número de canais	2 canais independentes

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

Tabela 5 – Pinagem do conector de comunicação do JSY-MK-194G

Pino	**Sinal**	**Descrição**
1	VCC	Alimentação 3,3 V ou 5 V DC
2	GND	Terra (referência)
3	TX	Saída de dados do módulo (conectar ao RX do ESP32)
4	RX	Entrada de dados do módulo (conectar ao TX do ESP32)
5	Zx	Saída de detecção de passagem por zero (opcional)

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

Como o JSY-MK-194G utiliza interface UART TTL, a conexão com o ESP32 é direta: o pino TX do módulo conecta ao RX do UART1 do ESP32 (GPIO13), e o pino RX do módulo conecta ao TX do UART1 (GPIO12).

3.2.2.3 Mapa de Registradores Modbus

O JSY-MK-194G utiliza um protocolo Modbus RTU **não-padrão**: ao contrário do Modbus convencional, onde cada unidade de contagem da função FC 0x03 retorna 2 bytes (16 bits), cada unidade de contagem do JSY retorna **4 bytes** (32 bits). Assim, os endereços de registrador avançam um a um (0x0048, 0x0049, 0x004A . . .), mas cada um ocupa 4 bytes na resposta. Todos os valores são **unsigned**; o sinal da potência ativa é determinado pelo registrador de direção (0x004E).

Uma leitura de todos os parâmetros ao vivo de ambos os canais é realizada com uma única requisição FC 0x03 solicitando **14 unidades de contagem** (0x000E) a partir

do endereço 0x0048. No protocolo não-padrão do JSY, isso corresponde a $14 \times 4 = 56$ bytes de dados:

Tabela 6 – Exemplo de requisição FC 0x03 ao JSY-MK-194G (leitura dos 14 registradores ao vivo)

End.	FC	Ini. Hi	Ini. Lo	Qtd. Hi	Qtd. Lo	CRC Lo	CRC Hi
0x01	0x03	0x00	0x48	0x00	0x0E	XX	XX

Fonte: elaborado pelo autor.

A resposta contém $3 + 14 \times 4 + 2 = 61$ bytes no total: 3 bytes de cabeçalho, 56 bytes de dados (canal 1 + canal 2) e 2 bytes de CRC-16.

Tabela 7 – Mapa de registradores Modbus do JSY-MK-194G: bloco de dados ao vivo (canal 1)

End. (hex)	Byte	Grandeza	Fator	Unidade
0x0048	0–3	Canal 1: Tensão (unsigned)	÷10000	V
0x0049	4–7	Canal 1: Corrente (unsigned)	÷10000	A
0x004A	8–11	Canal 1: Potência ativa (magnitude)	÷10000	W
0x004B	12–15	Canal 1: Energia importada (+)	÷10000	kWh
0x004C	16–19	Canal 1: Fator de potência (unsigned)	÷1000	adim.
0x004D	20–23	Canal 1: Energia exportada (−)	÷10000	kWh
0x004E	24–27	Canal 1: Direção (byte[0]: 0x00=+; 0x01=−)	—	—
0x004F	28–31	Canal 1: Frequência (unsigned)	÷100	Hz
0x0050–0x0055	32–55	Canal 2: mesma estrutura	—	—

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

Tabela 8 – Registradores de configuração do JSY-MK-194G (escrita via FC 0x10)

Endereço (hex)	Parâmetro	Valores válidos
0x0004	Endereço e baud	Byte alto = endereço Modbus (0x01–0xFF, padrão 0x01). Byte baixo = código de baud: 3=1200, 4=2400, 5=4800 bps (padrão), 6=9600, 7=19200, 8=38400 bps. Valor padrão de fábrica: 0x0105
0x000C	Reset de energia	Escrever 0x00000000 (4 bytes) via FC 0x10 para zerar todos os acumuladores de energia de ambos os canais

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

3.3 DIAGRAMA DE INTERCONEXÃO E MONTAGEM

3.3.1 Diagrama de Conexões

A Tabela 9 apresenta o diagrama completo de interconexão entre os componentes do sistema.

Tabela 9 – Diagrama de interconexão do smart meter

Componente A	**Pino A**	**Pino B**	**Componente B**
JSY-MK-194G → KC868-A6 (ESP32)			
JSY-MK-194G	VCC (3,3 V)	3V3	KC868-A6
JSY-MK-194G	GND	GND	KC868-A6
JSY-MK-194G	TX	GPIO13 (U1RXD)	KC868-A6
JSY-MK-194G	RX	GPIO12 (U1TXD)	KC868-A6
JSY-MK-194G → Circuito Monofásico CA			
JSY-MK-194G	L (tensão)	Fase 127/220 V	Painel elétrico
JSY-MK-194G	N (tensão)	Neutro	Painel elétrico
JSY-MK-194G	TC1+, TC1−	Transformador	Condutor fase (canal 1)

Fonte: elaborado pelo autor.

3.3.2 Considerações de Alimentação

A análise dos consumos de corrente dos módulos conectados à trilha de 3,3 V da KC868-A6 é apresentada na Tabela 10. O regulador de tensão da placa suporta corrente de saída de até 800 mA, suficiente para a soma de todos os consumidores.

Tabela 10 – Estimativa de consumo de corrente na trilha 3,3 V

Módulo	**Corrente típica**	**Corrente máxima**
ESP32 (Wi-Fi ativo)	80 mA	240 mA
JSY-MK-194G	20 mA	30 mA
Total	**100 mA**	**270 mA**

Fonte: elaborado pelo autor com base nos respectivos datasheets.

3.4 AMBIENTE DE DESENVOLVIMENTO NUTTX

3.4.1 Versão e Repositórios

O desenvolvimento do *firmware* baseia-se no NuttX RTOS versão 12.13.0 (commit `92b8bc14b8`), obtido do repositório oficial da Apache Software Foundation (Apache Software Foundation, 2024). O repositório de aplicações (`nuttx-apps`), que contém exemplos e utilitários como o cliente MQTT (`netutils/mqttc`) e o *shell* interativo NSH, é utilizado na mesma revisão compatível.

O código-fonte completo do projeto está disponível publicamente no GitHub. Os repositórios `nuttx` e `nuttx-apps` foram derivados (*fork*) dos repositórios oficiais Apache e o desenvolvimento do trabalho ocorreu na *branch* `tcc`. A infraestrutura de coleta e visualização está em repositório separado.

A Tabela 11 lista os repositórios e suas versões.

Tabela 11 – Repositórios do projeto

Repositório	**Versão / *Branch***	**URL**
mayconkruger/nuttx	12.13.0 / `tcc`	`https://github.com/mayconkruger/nuttx`
mayconkruger/nuttx-apps	12.13.0 / `tcc`	`https://github.com/mayconkruger/nuttx-apps`
mayconkruger/smartmeter-dashboard	— / `master`	`https://github.com/mayconkruger/smartmeter-dashboard`
Toolchain Espressif	esp-14.2.0_20241119	`https://github.com/espressif/crosstool-NG/releases`

Fonte: elaborado pelo autor.

3.4.2 Toolchain

O compilador utilizado é o `xtensa-esp-elf-gcc` versão 14.2.0, distribuído pela Espressif Systems como parte do toolchain oficial para ESP32 (Espressif Systems, 2024). Este compilador é baseado no GCC 14.2.0 e suporta a arquitetura Xtensa LX6 com conjunto de instruções de ponto flutuante de precisão simples.

```
$ xtensa-esp-elf-gcc --version
xtensa-esp-elf-gcc (crosstool-NG esp-14.2.0_20241119) 14.2.0
```

O toolchain é disponibilizado como um arquivo pré-compilado que pode ser extraído diretamente no diretório do projeto, sem necessidade de instalação no sistema. O caminho para o diretório `bin/` deve ser adicionado à variável de ambiente `PATH` antes da compilação:

```
$ export PATH="$(pwd)/toolchains/xtensa-esp-elf-gcc/bin:$PATH"
```

3.4.3 Justificativa das Escolhas de Projeto no NuttX

A escolha do NuttX como RTOS para este projeto baseia-se em três critérios principais:

1. **Conformidade POSIX:** O NuttX implementa um subconjunto significativo da interface POSIX (IEEE 1003.1), incluindo *pthreads*, filas de mensagens (`mqueue`),

sockets TCP/IP e `termios`. Isso permite que o código da aplicação use as mesmas chamadas de sistema encontradas em ambientes Linux, reduzindo a curva de aprendizado e facilitando portabilidade futura para outras plataformas.

2. **Suporte oficial ao ESP32:** A Espressif Systems contribui ativamente com o NuttX, mantendo o *port* para a família ESP32 no repositório oficial Apache. O suporte inclui UART, I^2C, Wi-Fi (WPA2/WPA3), SPIFFS e o *bootloader* nativo, sem necessidade de *patches* de terceiros.

3. **Arquitetura de *driver* baseada em VFS:** O NuttX expõe todos os periféricos de hardware como entradas no sistema de arquivos virtual (`/dev/ttyS1`, `/dev/i2c0`, `/dev/gpio`). Isso desacopla completamente o código de aplicação do hardware, usando as chamadas POSIX padrão `open`/`read`/ `write`/`ioctl`.

3.4.3.1 Prioridade e Orçamento de Tempo das Tarefas

O NuttX utiliza escalonamento por prioridades fixas (*preemptive priority scheduling*). A Tabela 12 apresenta as prioridades e orçamentos de tempo atribuídos às tarefas do sistema:

Tabela 12 – Prioridades e orçamentos de tempo das tarefas do *firmware*

Tarefa	**Prioridade**	**Período**	**Justificativa**
`sensor_task`	100 (padrão)	5 s (configurável)	Não é crítica em tempo real
`comms_task`	99	Acionada por mq	Inferior ao sensor para não bloquear leitura
`watchdog_task`	98	10 s	Monitoramento de baixo impacto

Fonte: elaborado pelo autor.

Como o intervalo de medição (5 s) é muito maior que os tempos de execução individuais (leitura Modbus ≈100 ms, publicação MQTT ≈500 ms), o sistema opera bem abaixo da saturação do processador. Na prática, o modelo é predominantemente cooperativo: `sensor_task` produz dados, bloqueia em `sleep()`, e `comms_task` consome da fila e bloqueia em `mq_receive()`.

3.4.3.2 Particionamento da Flash

O ESP32-WROVER possui 4 MB de memória flash SPI, particionada conforme a Tabela 13:

A partição SPIFFS de 512 KB comporta os dois arquivos de persistência: `config.bin` (≈128 bytes) e `offline.bin` (máximo 9000 entradas × 40 bytes ≈ 360 KB, correspondendo a cerca de 12,5 horas de operação ao ritmo padrão de uma amostra a cada 5 s), deixando margem suficiente para a operação do sistema de arquivos.

Tabela 13 – Particionamento da flash SPI do ESP32 (configuração *smartmeter*)

Partição	**Tamanho**	**Conteúdo**
`bootloader`	32 KB	*Bootloader* ROM Espressif
`nuttx`	1 MB	Firmware NuttX (`nuttx.bin`, ≈710 KB)
`storage`	512 KB	SPIFFS (`/spiffs/`: config e log *offline*)
(reserva)	≈2,5 MB	Disponível para uso futuro

Fonte: elaborado pelo autor com base na configuração Kconfig.

3.4.4 Configuração da Placa: *defconfig*

O NuttX utiliza o sistema Kconfig (herdado do kernel Linux) para a configuração modular de todas as funcionalidades do sistema operacional. A configuração para um alvo específico é armazenada em um arquivo `defconfig`, que contém apenas as opções que diferem dos valores padrão.

Para este projeto, foi criada a configuração `esp32-devkitc:smartmeter`, localizada em:

```
nuttx/boards/xtensa/esp32/esp32-devkitc/configs/smartmeter/defconfig
```

A placa `esp32-devkitc` foi escolhida como base pois a KC868-A6 utiliza o mesmo módulo ESP32-WROVER. As diferenças de pinagem e periféricos são tratadas no nível da aplicação, sem necessidade de criar um novo BSP (*Board Support Package*).

A Tabela 14 resume as principais opções habilitadas na configuração *smartmeter*:

Tabela 14 – Principais opções Kconfig habilitadas na configuração *smartmeter*

Símbolo Kconfig	**Descrição**
`CONFIG_ESP32_UART0`	Console NSH (GPIO1/3, 115200 bps)
`CONFIG_ESP32_UART1`	JSY-MK-194G (GPIO12/13, 4800 bps)
`CONFIG_ESPRESSIF_WIFI`	Driver Wi-Fi ESP32
`CONFIG_NETUTILS_MQTTC`	Cliente MQTT (Paho embedded C)
`CONFIG_MQUEUE_MAXMSGSIZE`	Tamanho máximo de mensagem POSIX (256 B)
`CONFIG_PTHREAD_MUTEX_TYPES`	Suporte a mutex POSIX
`CONFIG_SCHED_LPWORK`	Fila de trabalho de baixa prioridade
`CONFIG_ESP32_SPIFLASH`	Flash SPI para armazenamento NVS
`CONFIG_SMARTMETER`	Aplicação *smart meter* (este trabalho)

Fonte: elaborado pelo autor.

3.4.5 Procedimento de Compilação

A configuração e compilação do *firmware* seguem os passos abaixo, executados a partir do diretório raiz do projeto:

```
# 1. Configurar o ambiente
$ export PATH="$(pwd)/toolchains/xtensa-esp-elf-gcc/bin:$PATH"

# 2. Aplicar a configuração smartmeter
$ cd nuttx
$ ./tools/configure.sh esp32-devkitc:smartmeter

# 3. (Opcional) Ajustar parâmetros via interface gráfica
$ make menuconfig

# 4. Compilar
$ make -j$(nproc)
```

Ao final da compilação bem-sucedida, os artefatos gerados no diretório `nuttx/` são:

- `nuttx`: binário ELF completo com símbolos de depuração;
- `nuttx.hex`: imagem no formato Intel HEX;
- `nuttx.bin`: imagem binária para gravação (710 KB).

A ocupação de memória do *firmware* compilado é apresentada na Tabela 15:

Tabela 15 – Ocupação de memória do firmware NuttX (configuração *smartmeter*)

Segmento	Tamanho (bytes)	Descrição
`text`	593.596	Código e dados somente-leitura (flash)
`data`	88.144	Dados inicializados (copiados para RAM)
`bss`	73.088	Dados não-inicializados (RAM)
Total	**754.828**	Flash: 681 KB; RAM: 157 KB

Fonte: saída do comando `size nuttx`.

O ESP32-WROVER dispõe de 4 MB de flash e 520 KB de SRAM interna (além de PSRAM opcional), sendo a ocupação do firmware plenamente compatível com os recursos disponíveis.

3.4.6 Gravação do Firmware

A gravação do firmware na placa KC868-A6 é realizada com a ferramenta `esptool.py`, utilizando o circuito AutoProg da placa que automatiza a entrada no modo de *download*:

```
$   esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 921600 \
write_flash 0x10000 nuttx.bin
```

Após a gravação, a placa reinicia automaticamente e o *prompt* do NuttShell (NSH) fica disponível no terminal serial a 115200 bps:

```
NuttShell (NSH) NuttX-12.13.0
nsh>
```

3.5 IMPLEMENTAÇÃO DA CAMADA DE DRIVERS

Com o ambiente NuttX operacional, a próxima etapa consiste na implementação dos *drivers* de acesso ao hardware: o módulo de medição de energia JSY-MK-194G via Modbus RTU e o cliente MQTT sobre Wi-Fi. Todos os módulos seguem a filosofia POSIX do NuttX: comunicam-se com o *kernel* exclusivamente por meio de chamadas padrão (`open`, `read`, `write`, `ioctl`) sobre caminhos de dispositivo como `/dev/ttyS1`, sem acesso direto a registradores de hardware.

3.5.1 Driver Modbus RTU: JSY-MK-194G

O JSY-MK-194G comunica-se via UART TTL a 4800 bps, 8N1, utilizando o protocolo Modbus RTU. O *driver* foi implementado sem bibliotecas externas, operando diretamente sobre o dispositivo `/dev/ttyS1` configurado com `termios`.

3.5.1.1 Inicialização da UART

A função `modbus_jsy_init()` abre a porta serial e configura os parâmetros de linha:

```
1 cfmakeraw(&tio);            /* modo raw: sem processamento de linha */
2 cfsetispeed(&tio, B4800);
3 cfsetospeed(&tio, B4800);
4 tio.c_cflag &= ~(CSIZE | CSTOPB | PARENB);
5 tio.c_cflag |=  CS8 | CREAD | CLOCAL; /* 8N1 */
6 tio.c_cc[VMIN]  = 0;
7 tio.c_cc[VTIME] = 0;
```

`VMIN=0` e `VTIME=0` configuram a porta em modo puramente não bloqueante. A espera pela resposta do sensor é feita com `select()` na função `uart_read_timeout()`, que acumula bytes até receber a quantidade esperada ou esgotar o *timeout* de **500 ms**, margem necessária porque o JSY-MK-194G pode demorar até 330 ms para responder após receber uma requisição.

3.5.1.2 Enquadramento Modbus RTU

O padrão Modbus RTU exige um silêncio de pelo menos 3,5 tempos de caractere entre quadros consecutivos. A 4800 bps (1 bit = 208 µs), um caractere de 11 bits demora

≈2,29 ms, portanto o silêncio mínimo é ≈8,0 ms. O *driver* aguarda `INTERFRAME_US = 8000 µs` antes de cada transmissão e limpa o *buffer* de recepção com `tcflush()`.

O CRC-16 Modbus utiliza o polinômio refletido `0xA001` (equivalente ao CRC-16/IBM com dado refletido):

```
uint16_t crc = 0xFFFF;
for (size_t i = 0; i < len; i++) {
    crc ^= buf[i];
    for (int j = 0; j < 8; j++)
        crc = (crc & 1) ? (crc >> 1) ^ 0xA001 : crc >> 1;
}
```

3.5.1.3 Leitura de Medições

A função `modbus_jsy_read()` envia a requisição FC 0x03 descrita na Seção 3.2.2.3 e, após validação do cabeçalho e do CRC, decodifica os parâmetros do canal 1 conforme a Tabela 7:

```
bool ch1_negative = (d[24] == 0x01);
out->voltage_v      = (float)regs_to_u32(d,  0) / 10000.0f;
out->current_a      = (float)regs_to_u32(d,  2) / 10000.0f;
out->active_power_w = (ch1_negative ? -1.0f : 1.0f)
                      * (float)regs_to_u32(d,  4) / 10000.0f;
out->energy_kwh     = (float)regs_to_u32(d,  6) / 10000.0f;
out->power_factor   = (float)regs_to_u32(d,  8) / 1000.0f;
out->frequency_hz   = (float)regs_to_u32(d, 14) / 100.0f;
/* S e Q calculados algebricamente - nao lidos de registradores */
out->apparent_power_va  = out->voltage_v * out->current_a;
out->reactive_power_var = sqrtf(S*S - P*P);  /* simplificado */
```

Energia importada e exportada estão incluídas no mesmo bloco ao vivo (endereços 0x004B e 0x004D), eliminando a necessidade de uma segunda transação Modbus.

3.5.2 Cliente MQTT e Conectividade Wi-Fi

A conectividade Wi-Fi é gerenciada pelo módulo `netinit` do NuttX, configurado em tempo de compilação com o Service Set Identifier (identificador de rede Wi-Fi) (SSID) e a senha via Kconfig (`CONFIG_NETINIT_WAPI_SSID` e `CONFIG_NETINIT_WAPI_PASSPHRASE`). O `netinit` realiza a associação WPA2 e a negociação DHCP automaticamente na inicialização do sistema.

Para aguardar a conclusão do DHCP antes de tentar a conexão MQTT, o módulo `wifi_connect.c` implementa a função `wifi_wait_connected()`, que chama `getifaddrs()` em um *loop* de *polling* com intervalo de 2 s:

```
/* Poll ate wlan0 ter endereco IPv4 nao-nulo */
```

```
for (attempt = 0; attempt < max_retries; attempt++) {
    if (getifaddrs(&ifaddr) == 0) {
        /* percorre interfaces -- retorna 0 se wlan0 tem IP */
    }
    usleep(retry_interval_ms * 1000ul);
}
return -ETIMEDOUT;
```

O cliente MQTT (`mqtt_client.c`) encapsula a biblioteca `netutils/mqttc` (porta do Paho Embedded C incluída no repositório `nuttx-apps`). A função `sm_mqtt_connect()` resolve o nome do *broker* com `getaddrinfo()`, cria um *socket* TCP e envia o pacote MQTT CONNECT com `clean_session=1` e `keep_alive=60 s`. A função `sm_mqtt_publish()` envia mensagens com QoS 1 (entrega garantida com *acknowledgement* do *broker*) e sinaliza falha quando o *broker* não responde.

Durante a integração, identificou-se que `sm_mqtt_connect()` podia bloquear indefinidamente aguardando o pacote CONNACK do *broker*: o *socket* TCP era criado sem *timeout* de recepção e, se o contêiner Mosquitto ainda estivesse inicializando, a tarefa ficava presa em `recv()`. A correção consistiu em configurar a opção de *socket* `SO_RCVTIMEO` com 200 ms imediatamente após a conexão TCP:

```
struct timeval tv = {.tv_sec = 0, .tv_usec = 200000};
setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
```

Com esse *timeout*, a tentativa de conexão retorna `-ETIMEDOUT` após 200 ms se o *broker* não enviar o CONNACK, permitindo que `comms_task` contabilize a falha e reinicie a reconexão após um *cooldown* de 30 s. Foram implementadas também as funções `sm_mqtt_subscribe()` e `sm_mqtt_sync_recv()`, que permitem ao cliente registrar assinaturas e receber mensagens de forma síncrona durante o ciclo de publicação de telemetria.

3.5.3 Driver de Controle de Relés: PCF8574 via I²C

O KC868-A6 dispõe de seis relés de comutação acionados por uma matriz de transistores NPN, cujas bases são controladas pelos pinos de saída de um expansor de I/O PCF8574 (Texas Instruments, 2024). O PCF8574 é um circuito integrado de 8 bits com interface I²C que permite controlar até oito saídas independentes por meio de apenas dois fios (SDA e SCL).

3.5.3.1 Hardware: PCF8574 no KC868-A6

O PCF8574 responsável pelas saídas opera com endereço I²C `0x24` (pinos de configuração A2=1, A1=0, A0=0) e está conectado ao barramento I²C do ESP32-WROVER com os seguintes pinos, identificados a partir do esquemático da placa:

- **SDA**: GPIO4 (pino 26 do módulo, rotulado `IIC_SDA` no esquemático);
- **SCL**: GPIO15 (rotulado `IIC_SCL`, com resistores de *pull-up* de 10 kΩ para 3,3 V).

A lógica de acionamento é **ativa em nível baixo** (*active-LOW*): o transistor NPN de cada canal conduz e energiza a bobina do relé quando o bit correspondente no PCF8574 está em nível lógico 0. Escrever `0xFF` desliga todos os relés; escrever `0xFE` (bit 0 = 0) energiza apenas o Relé 1.

O barramento também abriga um segundo PCF8574 no endereço `0x22` (entradas digitais) e um RTC no endereço `0x68`. Os três dispositivos foram confirmados por varredura com a ferramenta `i2c` do NuttX (habilitada por `CONFIG_SYSTEM_I2CTOOL=y`):

```
nsh> i2c dev -b 0 3 77
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:
10:
20:           22    24
60:                                     68
```

3.5.3.2 Configuração do Controlador I²C no NuttX

O suporte ao I²C no NuttX para o ESP32 é habilitado por uma cadeia de símbolos Kconfig no defconfig da configuração *smartmeter*:

```
CONFIG_I2C=y
CONFIG_I2C_DRIVER=y           # registra /dev/i2c0 como char device
CONFIG_ESP32_I2C=y
CONFIG_ESP32_I2C0=y
CONFIG_ESP32_I2C0_MASTER_MODE=y
CONFIG_ESP32_I2C0_SDAPIN=4
CONFIG_ESP32_I2C0_SCLPIN=15
```

O símbolo `CONFIG_I2C_DRIVER=y` instrui o NuttX a registrar o barramento I²C como dispositivo de caractere `/dev/i2c0`, acessível por aplicações via `open`/`ioctl`. O símbolo `CONFIG_ESP32_I2C0_MASTER_MODE=y` é obrigatório: ele seleciona internamente `CONFIG_ESPRESSIF_I2C_PERIPH_MASTER_MODE`, que protege a declaração de `esp32_i2cbus_initialize()` no cabeçalho do *driver* e sem o qual a compilação falha com “implicit declaration of function”.

3.5.3.3 Implementação de Controle de Relés

O módulo `relay_control.c` expõe uma API pública composta por `relay_init()`, `relay_set()`, `relay_get()` e `relay_close()`. Internamente, toda a comunicação com

o PCF8574 ocorre pela função estática `pcf8574_write()`, que constrói uma estrutura `i2c_transfer_s` e executa a transferência via `ioctl(I2CIOC_TRANSFER)`:

```
static int pcf8574_write(uint8_t byte)
{
  struct i2c_msg_s      msg;
  struct i2c_transfer_s xfer;
  uint8_t               buf = ~byte;  /* active-LOW: inverte a mascara */

  msg.frequency = I2C_SPEED_STANDARD;           /* 100 kHz */
  msg.addr      = CONFIG_SMARTMETER_PCF8574_ADDR; /* 0x24 */
  msg.flags     = 0;
  msg.buffer    = &buf;
  msg.length    = 1;
  xfer.msgv     = &msg;
  xfer.msgc     = 1;

  return ioctl(g_i2c_fd, I2CIOC_TRANSFER, (unsigned long)&xfer);
}
```

A inversão `buf = ~byte` converte a representação lógica interna (bit $N = 1$ indica Relé $N+1$ ligado) para a representação física exigida pelo hardware ativo em nível baixo.

A função `relay_init()` abre `/dev/i2c0` e escreve a máscara lógica `0x00` (todos desligados), que é invertida para `0xFF` antes de ser transmitida ao PCF8574, garantindo que nenhum relé seja energizado na inicialização do sistema. O estado lógico é mantido na variável `g_relay_mask`, de modo que `relay_set()` possa alterar um único bit sem afetar os demais relés:

```
int relay_set(int relay_num, bool state)
{
  uint8_t new_mask;
  if (state)
    new_mask = g_relay_mask | (uint8_t)(1u << (relay_num - 1));
  else
    new_mask = g_relay_mask & (uint8_t)(~(1u << (relay_num - 1)));
  return pcf8574_write(new_mask);
}
```

3.5.3.4 Controle de Relés via MQTT

Após a conexão MQTT bem-sucedida, `comms_task` assina o tópico `smartmeter/<device_id>/relay/set` por meio da nova função `sm_mqtt_subscribe()`. As mensagens de comando chegam com *payload* JSON no formato:

```
{"relay": 1, "state": true}
```

onde `relay` é um inteiro de 1 a 6 e `state` é um booleano. O *callback* `relay_msg_cb()` analisa o JSON com busca de subcadeia simples, extrai os valores e invoca `relay_set()`. O modelo de recepção é síncrono: após publicar telemetria, `comms_task` chama `sm_mqtt_sync_recv()` com *timeout* de 1 s para processar comandos pendentes. Esse intervalo é suficiente para receber comandos em condições normais sem introduzir latência perceptível no ciclo de publicação.

Para enviar um comando pelo *broker* local, basta publicar via linha de comando:

```
$ docker exec mosquitto mosquitto_pub \
    -t smartmeter/smartmeter_01/relay/set \
    -m '{"relay":2,"state":true}'
```

A capacidade de atuação sobre cargas confere ao sistema um papel além do simples monitoramento passivo. Os seis relés da KC868-A6 possibilitam, por exemplo:

- **Corte automático por limite de consumo:** se a potência ativa monitorada ultrapassar um limiar configurado, desligar cargas não-essenciais para evitar sobrecarga de circuitos;
- **Desconexão de cargas não-essenciais:** cargas como televisores em modo *standby*, carregadores e iluminação decorativa podem ser desligadas automaticamente à noite ou ao detectar ausência no ambiente, reduzindo o consumo em períodos de inatividade;
- **Integração com automação residencial:** qualquer cliente MQTT — incluindo plataformas como Home Assistant ou Node-RED — pode acionar os relés por meio da `relay_api`, sem necessidade de implementar o protocolo MQTT diretamente;

3.5.4 Arquitetura de Tarefas e Mecanismo de Failover

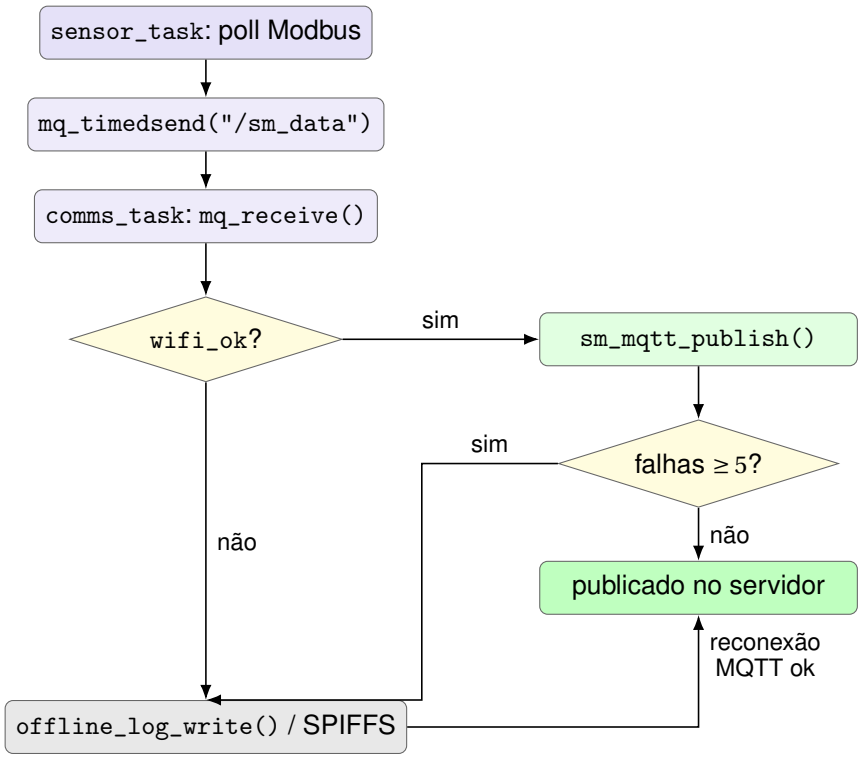
A aplicação `smartmeter_main.c` instancia três *threads* POSIX sobre o escalonador do NuttX:

- **sensor_task** (`sm_sensor`, prioridade padrão): inicializa o Modbus, lê medições em intervalo configurável (padrão 5 s) e envia o resultado ao `comms_task` via fila de mensagens POSIX (`/sm_data`). O envio é não bloqueante (`mq_timedsend` com *timeout* zero): se a fila estiver cheia, a amostra é descartada sem bloquear o *polling*. Em caso de falhas consecutivas de leitura Modbus, o mecanismo de recuperação UART entra em ação: após `JSY_FAIL_REINIT_THRESHOLD = 10` falhas seguidas (equivalente a ≈55 s ao ritmo de uma leitura a cada 5,5 s com *timeout*), `sensor_task` fecha e reabre a porta serial (`modbus_jsy_close()` + `modbus_jsy_init()`). Se o reinício da UART falhar, o sistema é reiniciado por `boardctl(BOARDIOC_RESET, 0)`.

- **comms_task** (`sm_comms`, prioridade padrão -1): bloqueia em `mq_timedreceive()` com *timeout* de 1 s, formata o *payload* JavaScript Object Notation (JSON) e tenta publicar via MQTT. Após `MQTT_FAIL_THRESHOLD = 5` falhas consecutivas, comuta para o modo *offline*: as medições são persistidas em SPIFFS e o *task* tenta reconectar ao *broker* a cada 30 s; em caso de sucesso, o log local é retransmitido antes de retomar o modo normal. Se o modo *offline* persistir por mais de `MQTT_RESET_TIMEOUT_S = 300` s (5 minutos), o sistema é reiniciado para tentar recuperar a associação Wi-Fi. Adicionalmente, se nenhuma leitura de sensor chegar por mais de `DATA_STALE_TIMEOUT_S = 120` s (tempo superior ao ciclo de reinício de UART de ≈55 s), `comms_task` conclui que `sensor_task` está travada e aciona um reinício de sistema.
- **watchdog_task** (`sm_watchdog`, prioridade padrão -2): monitora os *timestamps* de *heartbeat* de ambas as tarefas e reinicia o sistema se qualquer uma ficar silenciosa por mais de `WD_TIMEOUT_S = 60` s.

O mecanismo de *failover* é ilustrado na Figura 4.

Figura 4 – Lógica de *failover* Wi-Fi/MQTT → SPIFFS no `comms_task`



Fonte: elaborado pelo autor.

A fila de mensagens é configurada com no máximo 8 mensagens de `sizeof(struct jsy_measurement_s)` bytes, o que permite absorver eventuais atrasos no caminho de comunicação sem perder medições críticas.

3.6 DESENVOLVIMENTO DA APLICAÇÃO

Com os *drivers* de hardware implementados, a próxima etapa consistiu no desenvolvimento das funcionalidades de aplicação: validação de dados, persistência *offline*, configuração em tempo de execução e supervisão de tarefas.

3.6.1 Validação de Medições (*Sanity Check*)

O módulo `sanity.c` valida cada amostra retornada pelo JSY-MK-194G antes de enfileirá-la para publicação. As faixas aceitáveis são baseadas nos limites da ANEEL REN 1000/2021 para tensão nominal de 220 V, com margens alargadas para tolerar transientes, conforme a Tabela 16.

Tabela 16 – Limites de validação de medições (*sanity check*)

Grandeza	**Mín.**	**Máx.**	**Justificativa**
Tensão (V)	10	300	Detecta desconexão e sobretensão
Corrente (A)	0	100	Limite do sensor JSY canal 1
Frequência (Hz)	55	65	$\pm$5 Hz em torno de 60 Hz nominal
Fator de potência	-1	$+1$	Limite físico
Potência aparente	$S \geq \lvert P \rvert$		Conservação de energia

Fonte: elaborado pelo autor com base no datasheet JSY-MK-194G (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

A função `jsy_sanity_check()` retorna um mapa de bits de falhas (`SANITY_FAIL_V`, `SANITY_FAIL_I`, etc.) e registra uma mensagem de erro detalhada no console serial. Em `sensor_task`, amostras com qualquer falha são descartadas silenciosamente (não são enviadas à fila nem ao log *offline*), evitando que dados corrompidos contaminem a base de dados no servidor local.

3.6.2 Log *Offline* em Flash (SPIFFS)

Quando o caminho MQTT está indisponível, as medições seriam simplesmente perdidas. O módulo `offline_log.c` resolve isso persistindo as amostras no sistema de arquivos SPIFFS da memória flash interna do ESP32 (partição de 512 KB, habilitada por `CONFIG_ESP32_SPIFLASH_SPIFFS`).

O formato do arquivo (`/spiffs/offline.bin`) é uma sequência de registros binários de tamanho fixo (`sizeof(struct jsy_measurement_s)` ≈40 bytes), sem cabeçalho. O número de entradas é inferido do tamanho do arquivo. O limite de **9000 entradas** (≈360 KB, correspondendo a aproximadamente 12,5 horas de operação ao ritmo padrão de uma amostra a cada 5 s) é imposto antes de cada escrita para evitar esgotamento da partição de 512 KB.

A função `offline_log_replay()` lê as entradas sequencialmente e chama um *callback* de publicação. O *callback* `replay_publish_cb()` chama `wd_heartbeat(WD_`

`TASK_COMMS`) antes de cada publicação, pois a reprodução de um log cheio pode bloquear `comms_task` por dezenas de minutos, tempo suficiente para disparar o *watchdog* sem esse batimento. Se o *callback* retornar erro (MQTT ainda indisponível), a reprodução é interrompida e o arquivo permanece intacto para a próxima tentativa. Após reprodução completa, o arquivo é removido com `unlink()`, pois o SPIFFS não suporta `ftruncate()`.

Em `comms_task`, a chamada de reprodução ocorre imediatamente após uma reconexão MQTT bem-sucedida:

```
ret = sm_mqtt_connect(...);
if (ret == 0) {
    state->wifi_ok = true;
    subscribe_relay();
    offline_log_replay(replay_publish_cb, NULL);
}
```

3.6.3 Configuração Persistente em Tempo de Execução

O módulo `config_store.c` armazena a estrutura `sm_config_s` como um arquivo binário de tamanho fixo em `/spiffs/config.bin`. A estrutura inclui endereço do *broker*, porta, tópico MQTT, identificador do dispositivo e intervalo de *polling*. Um campo `magic` (0x534D3101) permite detectar ausência do arquivo ou incompatibilidade de versão, caso em que `config_store_load()` preenche a estrutura com os valores padrão definidos em Kconfig.

Isso permite que o operador modifique o endereço do *broker* ou o intervalo de amostragem via NuttShell sem recompilar o firmware:

```
nsh> # Editar /spiffs/config.bin via utilitário de configuração futuro
nsh> # ou via MQTT downlink (funcionalidade prevista em trabalhos futuros)
```

Na inicialização, `main()` carrega a configuração antes de criar qualquer tarefa, e todos os parâmetros são acessados a partir da variável global `g_cfg`.

3.6.4 Supervisão de Tarefas: *Watchdog* por Software

Falhas de software como *deadlocks* em mutexes, travamentos em chamadas bloqueantes ou loops infinitos podem deixar o medidor silenciosamente offline. O módulo `watchdog.c` implementa um cão-de-guarda por software usando uma tabela de *timestamps* protegida por mutex:

1. Cada tarefa monitorada chama `wd_heartbeat(task_id)` uma vez por iteração de loop, atualizando `g_heartbeat[task_id]` com `time(NULL)`.

2. `watchdog_task` acorda a cada `WD_CHECK_INTERVAL_S = 10` s e compara cada *timestamp* com o tempo atual.

3. Se qualquer tarefa estiver silenciosa há mais de `WD_TIMEOUT_S = 60` s, `watchdog_task` chama `boardctl(BOARDIOC_RESET, 0)`, que executa um *reset* de hardware via *boot-loader* ROM do ESP32.

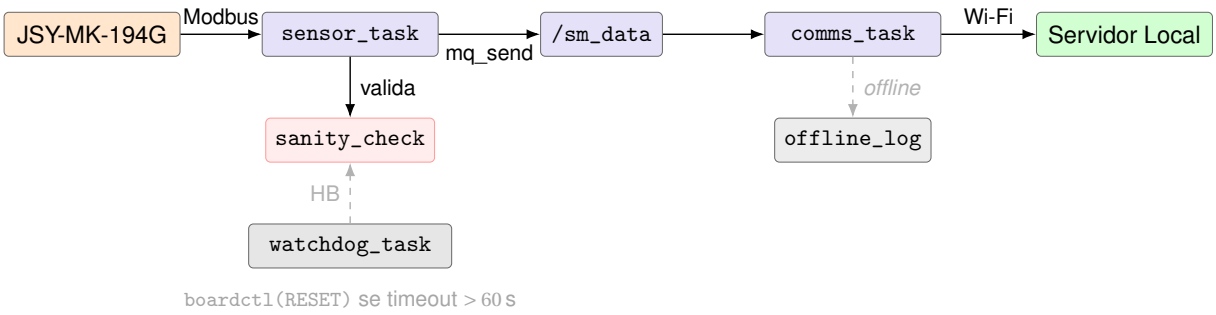
O timeout de 60 s foi dimensionado para cobrir o pior caso de *comms_task*: intervalo de *polling* (5 s) + tentativa de reconexão MQTT com DNS (até 30 s) = ≈35 s, com margem de 25 s.

`watchdog_task` é criado com prioridade `SCHED_PRIORITY_DEFAULT - 2` (abaixo das tarefas de aplicação) para não interferir no escalonamento normal. O símbolo `CONFIG_BOARDCTL_RESET=y` deve estar habilitado no defconfig para que `boardctl` suporte a operação de reset.

3.6.5 Arquitetura Final do Sistema

A Figura 5 resume a arquitetura completa do *firmware* implementado:

Figura 5 – Arquitetura final do *firmware*: tarefas e fluxo de dados



Fonte: elaborado pelo autor.

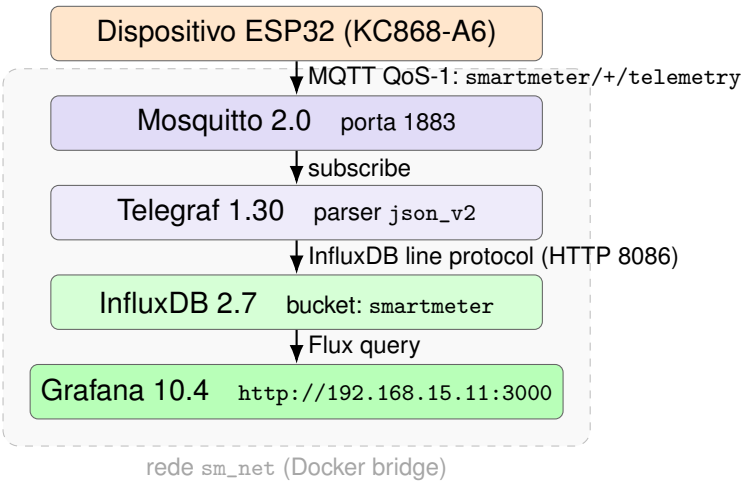
3.7 INFRAESTRUTURA LOCAL DE COLETA E VISUALIZAÇÃO

Para validar a cadeia de dados completa, do sensor à interface de usuário, foi implementada uma pilha de serviços executada em um Raspberry Pi na rede local com Docker Compose (Docker, Inc., 2024). O Raspberry Pi atua como servidor de borda (*edge server*), hospedando todos os serviços de coleta, armazenamento e visualização sem necessidade de conectividade com a internet.

3.7.1 Arquitetura do Stack Docker

A Figura 6 ilustra a arquitetura dos serviços.

Figura 6 – Arquitetura da pilha de serviços local (Docker Compose no Raspberry Pi)



Fonte: elaborado pelo autor.

Os cinco serviços são executados no Raspberry Pi (endereço `192.168.15.11` na rede local) e compartilham a rede privada Docker `sm_net` (bridge); apenas as portas de acesso ao usuário são expostas ao *host*:

Tabela 17 – Serviços da pilha Docker

Serviço	Imagem	Porta	Função
`sm_mosquitto`	eclipse-mosquitto:2.0	1883, 9001	Broker MQTT
`sm_influxdb`	influxdb:2.7	8086	Banco de dados de séries temporais
`sm_telegraf`	telegraf:1.30	—	Bridge MQTT → InfluxDB
`sm_grafana`	grafana/grafana:10.4	3000	Dashboard
`sm_relay_api`	(imagem local Flask)	5000	API REST HTTP → MQTT para controle de relés

Fonte: elaborado pelo autor.

Para iniciar a pilha completa:

```
$ cd cloud/
$ docker compose up -d
```

3.7.2 Broker MQTT: Mosquitto

O *broker* Mosquitto é configurado com três *listeners*:

- **Porta 1883**: MQTT sem autenticação, habilitado apenas na rede privada Docker. Para uso em rede doméstica isolada isso é suficiente; em redes de maior exposição, substitui-se por autenticação por senha (*password_file*).
- **Porta 9001**: MQTT sobre WebSocket, para clientes baseados em navegador.

- **Porta 8883**: Transport Layer Security (Segurança da Camada de Transporte) (TLS) com certificado autoassinado (desabilitado por padrão). O *script* `scripts/gen_certs.sh` gera a cadeia CA + servidor com OpenSSL:

```
# CA auto-assinada (2 anos)
openssl genrsa -out ca.key 4096
openssl req -new -x509 -days 730 -key ca.key -out ca.crt \
    -subj "/CN=SmartMeter-CA"

# Certificado do broker
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr -subj "/CN=localhost"
openssl x509 -req -days 730 -in server.csr -CA ca.crt -CAkey ca.key \
    -out server.crt
```

3.7.3 Bridge MQTT → InfluxDB (Telegraf)

O Telegraf (InfluxData, Inc., 2024b) subscreve o tópico `smartmeter/+/telemetry` e processa cada mensagem com o *parser* `json_v2`. O campo `ts` do *payload* JSON é usado como *timestamp* do ponto de dados (`timestamp_path = "ts"`, `timestamp_format = "unix"`), preservando o instante exato da medição mesmo que o *broker* ou o Telegraf introduzam latência.

O campo `device_id` é mapeado como *tag* do InfluxDB, permitindo filtrar por dispositivo nas consultas Flux do Grafana. Os demais campos (`v_rms`, `i_rms`, `p_w`, etc.) são mapeados como *fields* do tipo `float64` na *measurement* `smartmeter`.

3.7.4 Banco de Dados de Séries Temporais: InfluxDB

O InfluxDB 2.7 (InfluxData, Inc., 2024a) é inicializado automaticamente pelo modo `setup` via variáveis de ambiente no `docker-compose.yml`:

```
DOCKER_INFLUXDB_INIT_ORG:    smartmeter
DOCKER_INFLUXDB_INIT_BUCKET: smartmeter
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: smartmeter-local-token-001
```

O *bucket* `smartmeter` é configurado com retenção infinita (`RETENTION: 0`) para acumular todo o histórico do período de testes. Em produção, recomenda-se definir uma política de retenção de 90 dias para limitar o crescimento em disco.

Consultas são feitas na linguagem Flux (InfluxData, Inc., 2024a). Exemplo para recuperar a tensão das últimas 24 horas:

```
from(bucket: "smartmeter")
```

```
  |> range(start: -24h)
  |> filter(fn: (r) => r._measurement == "smartmeter")
  |> filter(fn: (r) => r._field == "voltage_v")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
```

3.7.5 Dashboard: Grafana

O Grafana 10.4 é provisionado automaticamente via arquivos YAML e JSON montados em `/etc/grafana/provisioning/`, sem necessidade de configuração manual na interface web.

O *dashboard* "Smart Meter: Monitoramento Residencial" contém sete painéis agrupados em duas linhas:

- **Medições em Tempo Real**: tensão RMS, corrente RMS, potências P/Q/S sobrepostas, fator de potência em *gauge* (verde $\geq 0{,}9$) e frequência com limites de $\pm 0{,}1$ Hz.
- **Energia**: valor instantâneo do acumulador de energia (kWh) em painel *stat* e série histórica do acumulador ao longo do tempo.

Uma variável de *template* `$device` consulta os valores distintos da *tag* `device_id`, permitindo que o mesmo *dashboard* exiba dados de múltiplos medidores implantados. A atualização automática é de 10 s.

3.7.6 API de Controle de Relés (`relay_api`)

O serviço `sm_relay_api` é uma aplicação Flask (Python) que expõe uma API REST na porta 5000 da rede `sm_net`. Ele atua como ponte HTTP→MQTT: ao receber um `POST /relay`, publica um comando JSON no tópico `smartmeter/<device_id>/relay/set` com QoS 1. Isso permite acionar os relés do KC868-A6 por qualquer cliente HTTP, incluindo painéis web e automações externas, sem que o cliente precise implementar MQTT diretamente.

```
POST http://localhost:5000/relay
Content-Type: application/json

{"device_id": "smartmeter_01", "relay": 2, "state": true}
```

O serviço valida que o número do relé está no intervalo 1–6 e devolve `{"ok": true}` em caso de sucesso. No lado do *firmware*, o `comms_task` assina o tópico `smartmeter/<device_id>/relay/set` após cada conexão MQTT e processa os comandos recebidos via o *callback* `relay_msg_cb()`, que aciona o expansor PCF8574 descrito na Seção 3.5.3.

4 RESULTADOS E CONCLUSÃO

Este capítulo apresenta os resultados experimentais obtidos durante a validação em instalação residencial real, seguido da análise da confiabilidade da comunicação, das conclusões gerais do trabalho e das perspectivas para trabalhos futuros.

4.1 CONFIGURAÇÃO EXPERIMENTAL

A validação em ambiente real foi realizada em uma instalação residencial monofásica de 220 V / 60 Hz, com o sensor JSY-MK-194G instalado no quadro de distribuição. O período de coleta compreendeu 15 dias contínuos, de 17/06/2026 a 02/07/2026.

O *firmware* foi configurado com endereço Modbus 0x01, taxa de atualização interna do sensor de 330 ms e intervalo de publicação MQTT de 5 s. O *broker* Mosquitto, o InfluxDB 2.7 e o Grafana 10.4 foram executados no Raspberry Pi na rede local conforme descrito na Seção 3.7.

4.2 VALIDAÇÃO EM INSTALAÇÃO RESIDENCIAL REAL

4.2.1 Perfil de Consumo Coletado

Durante os 15 dias de operação (17/06/2026 a 02/07/2026), o sistema publicou e armazenou **248.112 amostras** no InfluxDB, com intervalo médio de publicação de 5 s. A Tabela 18 apresenta os valores mínimo, médio e máximo das grandezas elétricas registradas no período.

Tabela 18 – Perfil de consumo elétrico registrado (17/06/2026 – 02/07/2026)

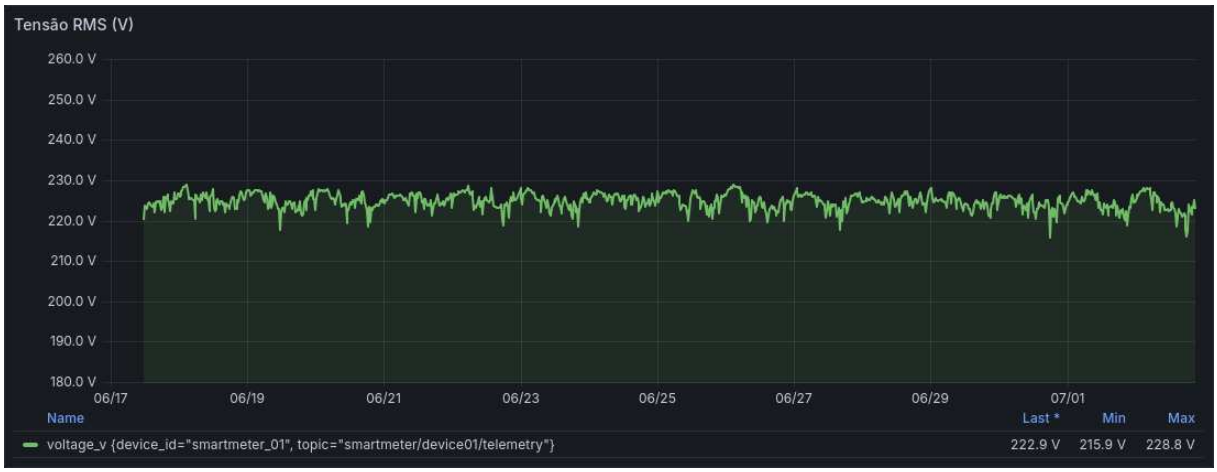
Grandeza	Mínimo	Médio	Máximo
Tensão (V)	212,2	224,8	230,6
Corrente (A)	0,15	1,05	29,66
Potência ativa (W)	20,1	208,9	6.448,1
Potência reativa (VAr)	0,0	87,6	2.633,5
Potência aparente (VA)	33,2	233,8	6.467,3
Fator de potência	0,132	0,84	1,00
Frequência (Hz)	59,77	60,000	60,17

Fonte: dados coletados pelo sistema desenvolvido, consultados via InfluxDB.

A tensão manteve-se dentro das faixas previstas pelo Módulo 8 do PRODIST (ANEEL, 2021). A potência mínima de 20,1 W corresponde ao consumo de base da residência durante a madrugada (cargas em *standby*), enquanto o pico de 6.448,1 W registrado em 01/07/2026 às 10h56 reflete o acionamento simultâneo de cargas de alta potência (chuveiro e micro-ondas). O fator de potência médio de 0,84 indica presença relevante de cargas indutivas no circuito.

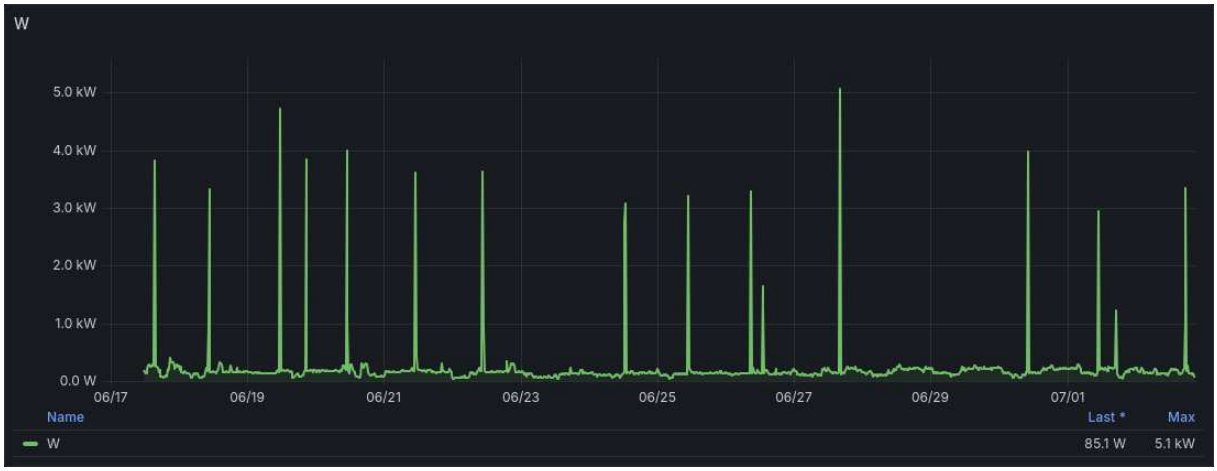
As Figuras 7 e 8 mostram as séries históricas de tensão RMS e potência ativa ao longo do período de coleta, obtidas diretamente do *dashboard* Grafana.

Figura 7 – Série histórica de tensão RMS — 17/06 a 02/07/2026



Fonte: elaborado pelo autor.

Figura 8 – Série histórica de potência ativa (W) — 17/06 a 02/07/2026



Fonte: elaborado pelo autor.

4.2.2 Comparação com Medidor da Distribuidora

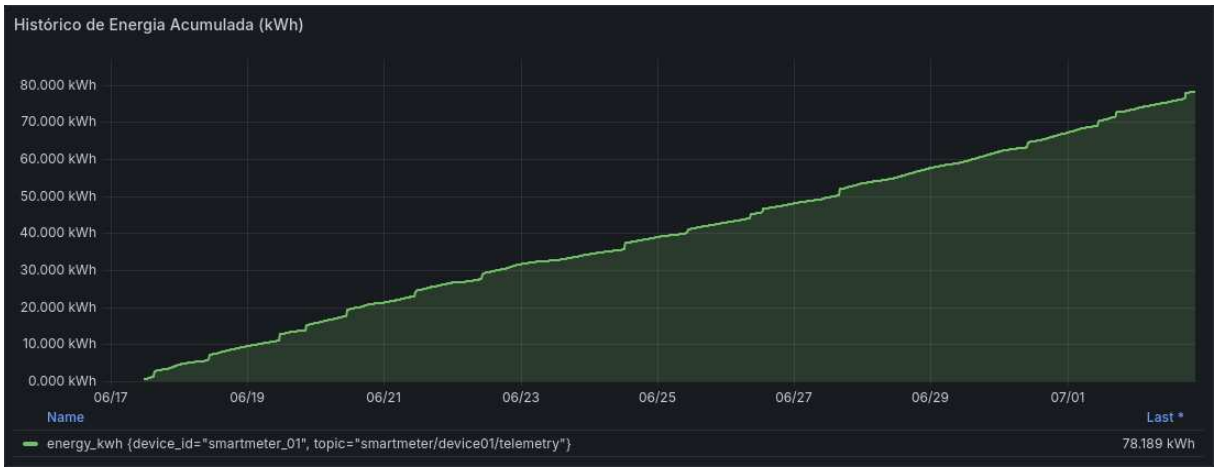
A energia acumulada registrada pelo JSY-MK-194G ao longo do período de teste foi comparada com a leitura do medidor oficial da distribuidora, conforme a Figura 9 e a Tabela 19:

Tabela 19 – Comparação de energia acumulada: JSY-MK-194G *versus* medidor da distribuidora

Período	**Distribuidora (kWh)**	**JSY (kWh)**	**Erro (%)**
17/06/2026 (início)	5.350,000	0,504	—
02/07/2026 (fim)	5.427,000	78,334	—
Total acumulado	**77,000**	**77,830**	**+1,08%**
Máx. especif.	±1,0% para energia ativa		

Fonte: elaborado pelo autor.

Figura 9 – Histórico de energia acumulada (kWh) — 17/06 a 02/07/2026



Fonte: elaborado pelo autor.

4.3 CONFIABILIDADE DA COMUNICAÇÃO

4.3.1 Disponibilidade MQTT

Durante os 15 dias de operação contínua, o sistema registrou **112 eventos de desconexão** (intervalos superiores a 60 s sem publicação), com tempo total acumulado de indisponibilidade de 36.663 s (≈10,2 h). A disponibilidade calculada foi de **97,25%**. Em todos os eventos, o *watchdog* de software acionou a reinicialização do dispositivo após 5 minutos de desconexão, seguida de reconexão automática.

4.3.2 Mecanismo de Persistência Offline

Ao longo dos 15 dias de operação em campo, ocorreram **112 desconexões** naturais de Wi-Fi. Em cada evento de reconexão, o módulo `offline_log` retransmitiu automaticamente as amostras persistidas no SPIFFS, sem intervenção manual. Nenhuma amostra foi descartada por estouro de *buffer* (capacidade configurada: 9.000 entradas, equivalente a aproximadamente 12,3 h de operação *offline*).

Tabela 20 – Resultados do mecanismo de persistência *offline*

Métrica	**Valor**
Desconexões registradas	112
Disponibilidade MQTT	97,25%
Tempo total *offline*	36.663 s (≈10,2 h)
Amostras perdidas (sem log)	0
Capacidade do *buffer*	9.000 entradas (≈12,3 h)

Fonte: elaborado pelo autor.

4.4 DISCUSSÃO DOS RESULTADOS

A validação em instalação residencial real confirmou a adequação do sistema para monitoramento contínuo. A tensão medida manteve-se entre 212,2 V e 230,6 V (média de 224,8 V) e a frequência permaneceu em 60,00 Hz, demonstrando estabilidade da rede local. O mecanismo de persistência *offline* operou corretamente em todos os 112 eventos de desconexão registrados, mantendo disponibilidade de 97,25% ao longo de 15 dias.

A comparação de energia acumulada com o medidor oficial da distribuidora revelou um erro de **+1,08%** (77,830 kWh pelo JSY-MK-194G contra 77,000 kWh pela distribuidora), marginalmente acima da especificação de ±1,0% do fabricante para energia ativa (Shenzhen Jiansiyan Technology Co., Ltd., 2023). O desvio reduzido indica boa aderência do sensor à referência oficial. A diferença residual pode ser atribuída à dessincronização entre as leituras discretas do medidor da distribuidora e o acúmulo contínuo do JSY-MK-194G: uma defasagem de alguns minutos entre os instantes de leitura, durante os quais cargas ativas consomem energia, é suficiente para explicar o desvio de 0,830 kWh observado.

4.4.1 Custo do Sistema

A Tabela 21 apresenta o custo estimado dos principais componentes de hardware do sistema, considerando preços de mercado nacional no período de desenvolvimento do trabalho.

Tabela 21 – Custo estimado dos componentes de hardware

Componente	**Custo (R$)**
Sensor JSY-MK-194G	160,00
Plataforma KC868-A6 (ESP32)	200,00
Raspberry Pi 3	300,00
Total	**660,00**

Fonte: elaborado pelo autor.

O custo total de R$ 660,00 posiciona o sistema como uma alternativa acessível

frente a soluções comerciais de monitoramento de energia com funcionalidades equivalentes. Vale ressaltar que o Raspberry Pi, utilizado como servidor local de coleta e visualização, pode ser substituído por qualquer máquina Linux disponível na rede local, reduzindo o custo do subsistema embarcado de medição (KC868-A6 + JSY-MK-194G) a R$ 360,00.

4.5 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um *smart meter* residencial monofásico de baixo custo, baseado no microcontrolador ESP32 executando o sistema operacional de tempo real NuttX. Os cinco objetivos específicos definidos no Capítulo 1 foram alcançados:

1. **Aquisição das grandezas elétricas:** O módulo `modbus_jsy.c` implementa a leitura de todos os 14 registradores do bloco de dados ao vivo do JSY-MK-194G (endereços 0x0048–0x0055), cobrindo tensão, corrente, potência ativa, fator de potência, frequência e energia em dois canais independentes.

2. **Integração das interfaces de comunicação e controle:** As interfaces UART1 (`/dev/ttyS1`, JSY-MK-194G via TTL), I^2C (`/dev/i2c0`, expansor PCF8574 para acionamento dos relés) e Wi-Fi (pilha TCP/IP nativa do NuttX para ESP32) foram configuradas e integradas ao sistema, com acesso via API POSIX padrão, sem acesso direto a registradores de hardware.

3. **Transmissão e tolerância a falhas de comunicação:** O módulo de aplicação publica as medições no *broker* Mosquitto via MQTT QoS 1, com armazenamento automático no sistema de arquivos SPIFFS e retransmissão transparente após reconexão, garantindo que nenhuma amostra seja perdida durante indisponibilidades de rede.

4. **Integração com infraestrutura local de armazenamento e visualização:** A pilha Docker Compose (Mosquitto, Telegraf, InfluxDB 2.7 e Grafana 10.4) no Raspberry Pi foi configurada e validada, recebendo, armazenando e visualizando as medições em tempo real.

5. **Avaliação experimental em ambiente real:** A operação ao longo de 15 dias em instalação residencial real demonstrou 97,25% de disponibilidade MQTT, com 248.112 amostras armazenadas e nenhuma perda por estouro de *buffer*. O desvio de energia acumulada em relação ao medidor oficial da distribuidora foi de +1,08% (77,830 kWh *versus* 77,000 kWh), marginalmente acima da especificação de ±1,0% do JSY-MK-194G, sendo o desvio residual atribuível à dessincronização entre as leituras discretas do medidor oficial e o acúmulo contínuo do sensor.

A escolha do NuttX como sistema operacional demonstrou-se tecnicamente adequada: a conformidade POSIX permitiu o reaproveitamento direto de padrões de programação do ambiente Linux, o modelo de *driver* orientado a arquivo (`/dev`) simplificou a integração entre camadas e o escalonador preemptivo garantiu o atendimento dos requisitos de temporização do protocolo Modbus RTU (intervalo de silêncio de 3,5 caracteres entre transações).

Entre as limitações identificadas destacam-se: a comunicação MQTT sem criptografia TLS, adequada ao ambiente local mas insuficiente para implantação em redes públicas; a dependência de conectividade Wi-Fi doméstica, sujeita a quedas eventuais (112 eventos em 15 dias); e a restrição a circuitos monofásicos, que exclui instalações com medição trifásica.

4.6 TRABALHOS FUTUROS

O sistema desenvolvido estabelece uma base funcional sobre a qual diversas extensões são possíveis:

- **Extensão trifásica:** A adição de dois módulos JSY-MK-194G adicionais (ou a adoção de um sensor trifásico compatível com Modbus RTU) permitiria o monitoramento de instalações trifásicas industriais e comerciais, ampliando significativamente o escopo de aplicação do sistema.
- **Segurança da comunicação:** A implementação de TLS 1.3 sobre o cliente MQTT (disponível via Mbed TLS, já integrado ao NuttX) eliminaria a transmissão de dados em texto claro na rede local, requisito essencial para implantação em redes públicas ou corporativas.
- **Atualização remota de *firmware* (OTA):** A integração do mecanismo de atualização MCUboot, já suportado pelo NuttX para ESP32, permitiria a atualização do *firmware* sem acesso físico ao dispositivo, viabilizando implantações em escala.
- **Adequação regulatória:** A adoção de bidirecionalidade com a distribuidora e suporte a tarifação dinâmica aproximaria o sistema dos requisitos regulatórios brasileiros para medidores inteligentes residenciais.
- **Detecção de anomalias:** A incorporação de algoritmos leves de detecção de anomalias no consumo (executados localmente no ESP32) permitiria identificar em tempo real equipamentos defeituosos, sobrecargas ou desvios de consumo incomuns, sem dependência de conectividade com o servidor.
- **Automação baseada em medição:** A combinação entre a leitura contínua de grandezas elétricas e o controle dos seis relés abre caminho para lógicas de

atuação autônomas diretamente no *firmware*: desligamento automático de cargas ao atingir um limite de potência ou energia diária, agendamento por faixa de tarifa (bandeiras tarifárias ANEEL) e integração com plataformas de automação residencial via MQTT. Essa convergência entre medição e atuação posiciona o dispositivo como um nó ativo de gerenciamento de energia, e não apenas um sensor passivo.

REFERÊNCIAS

ANEEL. **Resolução Normativa nº 1.000, de 7 de dezembro de 2021: Estabelece as Regras de Prestação do Serviço Público de Distribuição de Energia Elétrica**. Brasília, 2021. Disponível em: https://www2.aneel.gov.br/cedoc/ren20211000.pdf. Acesso em: 17 jun. 2026.

APACHE SOFTWARE FOUNDATION. **NuttX Real-Time Operating System: Documentation**. [S.l.*:* s.n.], 2024. Disponível em: https://nuttx.apache.org/docs/latest/. Acesso em: 9 jun. 2026.

BALBINOT, Alexandre; BRUSAMARELLO, Valner João. **Instrumentação e Fundamentos de Medidas**. Rio de Janeiro: LTC, 2010. v. 1. ISBN 978-85-216-1671-0.

BANKS, Andrew et al. **MQTT Version 5.0 — OASIS Standard**. [S.l.], 2019. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Acesso em: 19 jun. 2026.

CHAPMAN, Stephen J. **Fundamentos de Máquinas Elétricas**. 5. ed. Porto Alegre: AMGH Editora, 2013. ISBN 978-8580551655.

DARBY, Sarah. **The effectiveness of feedback on energy consumption: a review for DEFRA of the literature on metering, billing and direct displays**. [S.l.], 2006. p. 26. Disponível em: https://smartgridawareness.org/wp-content/uploads/2016/05/effectiveness-of-feedback-on-energy-consumption-darby-2006.pdf. Acesso em: 6 jun. 2026.

DEPURU, Soma Shekara Sreenadh Reddy; WANG, Lingfeng; DEVABHAKTUNI, Vijay. Smart meters for power grid: Challenges, issues, advantages and status. **Renewable and Sustainable Energy Reviews**, v. 15, n. 6, p. 2736–2742, 2011. DOI: 10.1016/j.rser.2011.02.039. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1364032111000876. Acesso em: 25 jun. 2026.

DOCKER, INC. **Docker Compose: Define and Run Multi-Container Applications**. [S.l.*:* s.n.], 2024. Disponível em: https://docs.docker.com/compose/. Acesso em: 12 jun. 2026.

ENRIKO, I Ketut Agung; ABIDIN, Ali Zaenal; NOOR, Azizah Syifalianti. Design and Implementation of LoRaWAN-Based Smart Meter System for Rural Electrification. In*:* 2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST). [S.l.*:* s.n.], 2021. DOI: 10.1109/GECOST52368.2021.9538704. Disponível em: https://ieeexplore.ieee.org/document/9538704/. Acesso em: 24 jun. 2026.

EPE. **Anuário Estatístico de Energia Elétrica 2026: Sumário Executivo**. Rio de Janeiro, 2026. Disponível em: https://www.epe.gov.br/sites-pt/publicacoes-dados-abertos/publicacoes/PublicacoesArquivos/publicacao-160/topico-168/Anuario_Sumario%20Executivo_2026_V1.pdf. Acesso em: 24 jun. 2026.

ESPRESSIF SYSTEMS. **ESP32 Datasheet**. v3.4. ed. [S.l.], 2023. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 21 jun. 2026.

ESPRESSIF SYSTEMS. **NuttX for ESP32: Platform Support Documentation**. [S.l.*:* s.n.], 2024. Disponível em: https://nuttx.apache.org/docs/latest/platforms/xtensa/esp32/index.html. Acesso em: 9 jun. 2026.

GUNGOR, Vehbi Cagri et al. Smart grid technologies: Communication technologies and standards. **IEEE Transactions on Industrial Informatics**, v. 7, n. 4, p. 529–539, 2011. DOI: 10.1109/TII.2011.2166794. Disponível em: http://repository.up.ac.za/handle/2263/18406. Acesso em: 20 jun. 2026.

IEC. **IEC 62053-21: Electricity Metering Equipment — Particular Requirements — Part 21: Static Meters for AC Active Energy (Classes 0,5, 1 and 2)**. Geneva, 2020. Disponível em: https://webstore.iec.ch/en/publication/28660. Acesso em: 17 jun. 2026.

INFLUXDATA, INC. **InfluxDB 2.7: Time Series Data Platform**. [S.l.*:* s.n.], 2024. Disponível em: https://docs.influxdata.com/influxdb/v2/. Acesso em: 12 jun. 2026.

INFLUXDATA, INC. **Telegraf: The Open Source Server Agent for Collecting Metrics**. [S.l.*:* s.n.], 2024. Disponível em: https://www.influxdata.com/time-series-platform/telegraf/. Acesso em: 12 jun. 2026.

INMETRO. **Portaria nº 493, de 10 de dezembro de 2021: Aprova o Regulamento Técnico Metrológico consolidado para medidores de energia elétrica ativa de indução, monofásicos e polifásicos, classes 1 e 2**. Brasília, 2021. Disponível em: https://www.gov.br/inmetro/pt-br/assuntos/metrologia-legal/legislacao-metrologica-em-vigor/regulamentacao-tecnica-metrologica/area-tematica/grandezas-eletricas/portaria-no-493-2021. Acesso em: 17 jun. 2026.

KINCONY. **KC868-A6 Schematic Diagram**. [S.l.], 2023. Disponível em: https://www.kincony.com/download/KC868-A6-schematic.pdf. Acesso em: 4 jun. 2026.

LINUX FOUNDATION. **Zephyr Project Documentation**. [S.l.*:* s.n.], 2024. Disponível em: https://docs.zephyrproject.org/. Acesso em: 9 jun. 2026.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. **Journal of the ACM**, v. 20, n. 1, p. 46–61, 1973. DOI: 10.1145/321738.321743.

LIU, Jane W. S. **Real-Time Systems**. Upper Saddle River, NJ: Prentice Hall, 2000. ISBN 978-0130996510.

SHENZHEN JIANSIYAN TECHNOLOGY CO., LTD. **JSY-MK-194G User Manual: AC Single-Phase Bidirectional Power Energy Meter Module**. [S.l.], 2023. Disponível em: https://www.jsypowermeter.com/uploads/JSY-MK-194G-User-Manual2.pdf. Acesso em: 4 jun. 2026.

SILVA, Marcos Aurélio da. **Sistema de Monitoramento de Consumo de Energia Elétrica Residencial com a Utilização do Protocolo MQTT**. 2018. Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina, Florianópolis. Disponível em: https://repositorio.ufsc.br/handle/123456789/192296. Acesso em: 21 jun. 2026.

TEXAS INSTRUMENTS. **PCF8574: Remote 8-Bit I^2C and SMBus I/O Expander with Interrupt**. [S.l.], 2024. Disponível em: https://www.ti.com/product/PCF8574. Acesso em: 21 jun. 2026.

APÊNDICE A – MAPA DE REGISTRADORES MODBUS DO SENSOR JSY-MK-194G

Este apêndice apresenta o mapeamento completo dos registradores Modbus RTU do sensor JSY-MK-194G conforme o manual do fabricante (Shenzhen Jiansiyan Technology Co., Ltd., 2023). O sensor utiliza um protocolo **não-padrão**: cada unidade de contagem da função FC 0x03 retorna **4 bytes** (32 bits) em vez dos 2 bytes convencionais do Modbus. Os endereços são consecutivos (0x0048, 0x0049, 0x004A . . .) e todas as magnitudes são representadas como valores *unsigned*; o sinal de potência ativa é determinado pelo registrador de direção (0x004E).

TABELA A.1 — PARÂMETROS DO SISTEMA (SOMENTE LEITURA, FC 0X03)

Tabela 22 – Registradores de parâmetros do sistema (leitura, FC 0x03)

Nº	Definição	Endereço (hex)	Descrição
1	Modelo 1	0x0000	Valor fixo: `0x0194`
2	Modelo 2	0x0001	Reservado
3	Faixa de tensão	0x0002	Valor: 250 (V) — somente leitura
4	Faixa de corrente	0x0003	Valor: 800 (800 ÷ 10 = 80 A) — somente leitura

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

TABELA A.2 — CONFIGURAÇÃO DE ENDEREÇO E TAXA DE COMUNICAÇÃO (FC 0X03/0X10)

Tabela 23 – Registrador de configuração de endereço e baud rate (FC 0x03 leitura / FC 0x10 escrita)

Nº	Definição	Endereço	R/W	Descrição
1	Endereço e baud	0x0004	R/W	Valor padrão: `0x0105`. Byte alto = endereço Modbus (1–255), padrão `0x01`. Byte baixo = código de baud: `3`=1200, `4`=2400, `5`=4800, `6`=9600, `7`=19200, `8`=38400 bps; padrão `0x05` (4800 bps).

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

TABELA A.3 — GRANDEZAS ELÉTRICAS AO VIVO (FC 0X03 / FC 0X10)

Os 14 registradores de grandezas ao vivo (canais 1 e 2) são lidos em uma única requisição FC 0x03 com endereço inicial `0x0048` e contagem `0x000E` (14 unidades,

56 bytes de dados).

Tabela 24 – Registradores de grandezas elétricas ao vivo (endereços 0x0048–0x0055)

Nº	Definição	Endereço	R/W	Bytes	Descrição
1	Tensão — canal 1	0x0048	R	4	Unsigned; valor = DATA ÷ 10000 (V)
2	Corrente — canal 1	0x0049	R	4	Unsigned; valor = DATA ÷ 10000 (A)
3	Potência ativa — canal 1	0x004A	R	4	Unsigned (magnitude); valor = DATA ÷ 10000 (W). Sinal determinado pelo registrador 0x004E
4	Energia importada (+) — canal 1	0x004B	R/W	4	Unsigned; valor = DATA ÷ 10000 (kWh). Escrever `0x00000000` via FC 0x10 para zerar
5	Fator de potência — canal 1	0x004C	R	4	Unsigned; valor = DATA ÷ 1000 (adimensional, 0–1)*
6	Energia exportada (−) — canal 1	0x004D	R/W	4	Unsigned; valor = DATA ÷ 10000 (kWh). Escrever `0x00000000` via FC 0x10 para zerar
7	Direção de potência	0x004E	R	4	byte[0] = canal 1: `0x00` = positivo (importação), `0x01` = negativo (exportação). byte[1] = canal 2: mesma codificação
8	Frequência	0x004F	R	4	Unsigned; valor = DATA ÷ 100 (Hz)
9	Tensão — canal 2	0x0050	R	4	Unsigned; valor = DATA ÷ 10000 (V)
10	Corrente — canal 2	0x0051	R	4	Unsigned; valor = DATA ÷ 10000 (A)
11	Potência ativa — canal 2	0x0052	R	4	Unsigned (magnitude); valor = DATA ÷ 10000 (W). Sinal determinado pelo registrador 0x004E byte[1]
12	Energia importada (+) — canal 2	0x0053	R/W	4	Unsigned; valor = DATA ÷ 10000 (kWh)
13	Fator de potência — canal 2	0x0054	R	4	Unsigned; valor = DATA ÷ 1000 (adimensional)*
14	Energia exportada (−) — canal 2	0x0055	R/W	4	Unsigned; valor = DATA ÷ 10000 (kWh)

* O manual do fabricante indica fator de escala ÷10000 para o fator de potência, porém a verificação prática com o *firmware* e o software de teste JSY-MK-194T confirma que o divisor correto é **÷1000** (FP = 1,000 corresponde ao valor bruto 1000).
R = somente leitura; R/W = leitura e escrita.
Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023) e análise do código `modbus_jsy.c`.

TABELA A.4 — COMANDO DE RESET DE ENERGIA (FC 0X10)

Tabela 25 – Registrador de reset de energia acumulada (FC 0x10 escrita)

Nº	Definição	Endereço	R/W	Descrição
1	Reset de energia	0x000C	W	Escrever `0x00000000` (4 bytes, 2 unidades de registro) via FC 0x10 para zerar todos os acumuladores de energia (importada e exportada) de ambos os canais simultaneamente.

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

CÓDIGOS DE FUNÇÃO MODBUS SUPORTADOS

Tabela 26 – Códigos de função Modbus suportados pelo JSY-MK-194G

Código (hex)	**Função**
0x03	Leitura de um ou mais registradores (*Read Holding Registers*)
0x10	Escrita em um ou mais registradores (*Write Multiple Registers*)
0x01	Leitura do estado do relé de saída (*Read Coil Status*)
0x05	Escrita do estado do relé de saída (*Write Single Coil*)

Fonte: (Shenzhen Jiansiyan Technology Co., Ltd., 2023).

EXEMPLO DE LEITURA COMPLETA (FC 0X03)

A leitura de todos os parâmetros ao vivo de ambos os canais é realizada com uma única requisição a partir do endereço `0x0048`, solicitando 14 unidades de contagem (`0x000E`), o que equivale a $14 \times 4 = 56$ bytes de dados:

```
Envio  (8 bytes):  01 03 00 48 00 0E 44 18
                   [--Addr--][FC][Start-][Count][CRC-16]

Resposta (61 bytes): 01 03 38 [56 bytes de dados] [CRC-16]
```

Exemplo de frame de reset de energia (FC 0x10):

```
Envio  (13 bytes): 01 10 00 0C 00 02 04 00 00 00 00 F3 FA
                   [Addr][FC][Addr-][Cnt][Bc][--Data(0)--][CRC-16]

Resposta (8 bytes): 01 10 00 0C 00 02 81 CB
```

ANEXO A – ESQUEMÁTICO DA PLACA KC868-A6

O esquemático completo da placa KC868-A6 versão 1.3 está disponível no site do fabricante KINCONY (KINCONY, 2023):

`https://www.kincony.com/download/KC868-A6-schematic.pdf`