

Desenvolvimento de um *gateway* Modbus RTU-TCP usando plataformas abertas

Juliano Viero

Engenharia de Controle e Automação
IFRS - Campus Farroupilha
Av. São Vicente, 785,
Farroupilha, RS, Brasil
Email: juliano188@hotmail.com

Gustavo Künzel

Engenharia de Controle e Automação
IFRS - Campus Farroupilha
Av. São Vicente, 785,
Farroupilha, RS, Brasil
Email: gustavo.kunzel@farroupilha.ifrs.edu.br

Resumo—A expansão dos processos de automação nas indústrias desencadeou a necessidade do emprego de protocolos de comunicação, os quais viabilizam a transferência padronizada de informações entre dispositivos. Um destes protocolos é o Modbus, sendo desenvolvido no ano de 1979, com a finalidade de obter a troca de dados entre os Controladores Lógicos Programáveis (CLPs). O trabalho deste TCC II consiste na elaboração de um *gateway*, o qual realiza a intercomunicação de equipamentos ou sistemas que empregam os modos de operação Modbus RTU e Modbus TCP. O projeto compreende desde a elaboração de um fluxograma principal até a seleção do *hardware* principal e periféricos. Para o desenvolvimento do *gateway* utilizou-se plataformas e componentes *open source*. Com o intuito de validar o projeto foi utilizado um *software* supervisor *open source*, que objetiva cumprir o papel de cliente TCP, e foi elaborado um sistema de testagem e um sistema de verificação, sendo que, neste último utilizou-se um inversor de frequência como servidor RTU. Desta forma, engloba-se neste trabalho o conceito de integração vertical abordada na Indústria 4.0, uma vez que com o *gateway* é possível interligar um sistema supervisor com dispositivos como CLPs, inversores de frequência, entre outros.

Palavras-chave—Protocolos de comunicação, Modbus, *gateway*, Modbus RTU, Modbus TCP, Indústria 4.0, *open source*.

I. INTRODUÇÃO

O protocolo Modbus foi desenvolvido com a finalidade de ser utilizado em ambientes industriais, e desde seu lançamento tem sido implementado por diversos fornecedores de dispositivos, uma vez que é um protocolo simples e flexível, porém robusto. Sua função básica é promover a comunicação entre diversos dispositivos, conectados em uma rede ou em um barramento. Os modos de operação típicos são: TCP, RTU e ASCII. Através de um *gateway* pode-se realizar a comunicação entre equipamentos com modos de operação distintos [1].

Em uma perspectiva atual, este conceito de dispositivos interligados é abordado na Indústria 4.0 com a denominação de integração vertical, tendo por objetivo promover a conexão entre todos os níveis hierárquicos da pirâmide da automação. Esses níveis são divididos em: chão de fábrica (sensores e atuadores), nível de controle (CLP), nível de supervisão (SCADA), nível de operação (MES) e nível de planejamento (ERP) [2]. Desta maneira, a indústria obtém vários benefícios advindos da interconexão dos diferentes níveis, como: redução de custos, diminuição da burocracia, eliminação de garga-

los operacionais, maior produtividade e rotinas com menor número de conflitos [3]. Na Fig. 1 são ilustrados os níveis da pirâmide da automação e a intercomunicação que o *gateway* Modbus pode realizar entre os três níveis hierárquicos mais baixos da pirâmide.

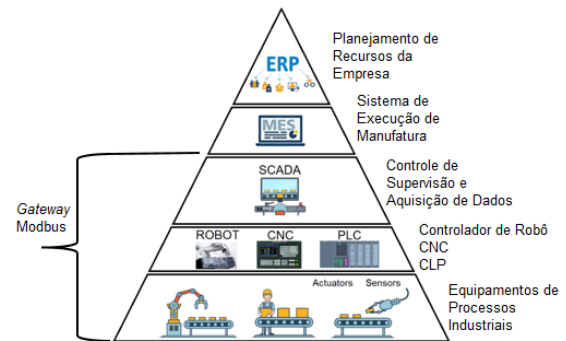


Figura 1. Pirâmide da automação [2]

A fim de realizar um projeto que aborde o tema da integração citada acima, visto ser cada vez mais necessária no ambiente fabril, é proposto neste trabalho um *gateway* Modbus, que realiza a comunicação entre as interfaces Modbus RTU e Modbus TCP utilizando plataformas e componentes *open source*. Com o intuito de atender ao objetivo do projeto, foi desenvolvido um fluxograma representado na Fig. 6, sendo a base para a elaboração do algoritmo principal, que foi transcrito a um *software* e carregado à uma placa microcontrolada. Esta placa, que possui a função principal do *gateway*, foi conectada a diferentes interfaces de comunicação com a finalidade de realizar a comunicação com os modos de operação RTU e TCP. Foram implementadas outras funcionalidades com o intuito de auxiliar o usuário, como exemplo: a alteração de configurações, um *display* para o usuário visualizar algumas informações, LEDs com funções de exibir *status* do dispositivo e um botão de *reset* que permite ao *gateway* regressar às configurações iniciais. Para validar o projeto, foi implementado e executado um sistema de testagem e um sistema de verificação, que serão detalhados no decorrer deste trabalho. O diferencial do *gateway* deve-se pela utilização de plataformas

e componentes *open source*.

Para uma melhor compreensão da organização, o trabalho está fragmentado em 5 seções. Na seção 2 é apresentada a revisão bibliográfica, que tem por objetivo elucidar os principais conceitos teóricos, a análise resumida de trabalhos relacionados e a síntese de *gateways* comerciais. A seção 3 retrata a proposta do projeto, que visa apresentar o fluxograma do algoritmo, o diagrama de blocos, o *hardware* principal, as configurações do *gateway* e os periféricos. A seção 4 apresenta a implementação e resultados, que engloba tópicos como: montagem e programação, supervisor ScadaBR, sistema de testagem e sistema de verificação. Na seção 5 são retratadas as conclusões.

II. REVISÃO BIBLIOGRÁFICA

A. Modelo OSI

No fim dos anos 1970, a *International Organization for Standardization* (ISO) propôs que as redes de computadores fossem descritas em torno de sete camadas, e este modelo foi denominado de *Open Systems Interconnection* (OSI). Por meio da arquitetura de camadas é possível analisar um fragmento bem delimitado advindo de um sistema grande e complexo. Uma das vantagens desta arquitetura é a simplicidade de efetuar uma alteração na execução de um serviço advindo de uma camada. Assim, desde que a funcionalidade do serviço em si não seja modificada, a operação global do sistema se manterá inalterada [4].

As sete camadas do modelo OSI são: (1) camada física – trata-se do canal de comunicação dos bits e estabelece as conexões elétricas, mecânicas e de sincronização dos dispositivos, (2) camada de enlace – tem objetivo de converter uma comunicação normal para uma aparentemente sem erros e divide os dados em pequenos pacotes, (3) camada de rede – determina como os pacotes são enviados/roteados desde sua origem até seu destino, (4) camada de transporte – sua função é de receber os dados da camada acima (camada de sessão), dividi-los em pacotes e encaminhá-los a camada de rede, e também assegurar que os fragmentos cheguem de maneira correta, (5) camada de sessão – proporciona que vários usuários, em diferentes dispositivos, realizem uma sessão de comunicação, (6) camada de apresentação – realiza uma codificação padrão a fim de tornar possível a comunicação entre diversas formas de estrutura de dados, (7) camada de aplicação – dispõe de diversos protocolos de comunicação essenciais ao usuário, como exemplo o *HyperText Transfer Protocol* (HTTP), que é a base da comunicação existente em toda Internet [5].

B. Conceituação geral do Protocolo Modbus

Modbus é um protocolo público, criado por Gould Modicon (atual Schneider Electric) em 1979, com finalidade de controle de processos, e em contraposição com os diferentes padrões industriais, não possui uma definição da camada física, nível 1 do modelo OSI. O protocolo possui uma grande aceitação entre os fornecedores de equipamentos e pelos usuários finais [6]. Provê a comunicação entre dispositivos conectados em uma

rede ou barramento, por meio das arquiteturas: cliente/servidor ou mestre/escravo. O protocolo localiza-se na camada de aplicação, ou seja, no nível 7 do modelo OSI. Permite a comunicação dentre as diversas arquiteturas de redes, e deste modo, inúmeros dispositivos como: CLPs, Interface Homem-Máquina (IHM), dispositivos de Entrada/Saída, são capazes de realizar comunicação via Modbus [1].

No modelo de comunicação do protocolo, um cliente inicia uma requisição (*query*) e os outros dispositivos denominados servidores respondem enviando os dados requeridos (*response*). Em certos casos, o cliente pode realizar a alteração de algum dado no servidor. Caso ocorra algum erro de comunicação ou quando algum comando não pode ser aceito pelo servidor é enviada uma mensagem de exceção (*exception*). Em alguns casos o servidor não responderá, e nesta situação o cliente irá identificar um *timeout*. O cliente pode enviar uma mensagem individual à cada servidor, ou enviar uma requisição à vários servidores por meio de um endereço de *broadcast* [7].

O protocolo Modbus, de forma genérica, independente das camadas inferiores, possui uma definição de unidade de dados denominada de *Protocol Data Unit* (PDU). Este contém as funções, campo *Function Code*, e os dados usados por cada função, campo *Data*. A Tabela I descreve as funções Modbus mais comuns. Para o modo de operação RTU, os campos *Additional Address* e *Error Check*, são acrescentados ao quadro PDU. Esse quadro denomina-se de *Application Data Unit* (ADU). Para o modo de operação TCP, utiliza-se uma outra estrutura de quadro o qual é acrescentado o cabeçalho *Modbus Application Protocol* (MBAP) ao quadro PDU. Esse quadro denomina-se de Modbus TCP/IP ADU [8]. Na Fig. 2 é ilustrado a estrutura de quadros PDU e ADU e na Fig. 3 é ilustrado o quadro com o cabeçalho MBAP.

Tabela I
FUNÇÕES MODBUS

Cod. Hex	Descrição
01	Leitura de saídas discretas
02	Leitura de entradas discretas
03	Leitura de registros <i>holding</i>
04	Leitura de registros de entrada
05	Escrita de única saída
06	Escrita de único registro <i>holding</i>
07	Leitura de status de exceção
08	Funções de diagnósticos
10	Escrita de múltiplos registro <i>holding</i>
0F	Escrita de múltiplas saídas

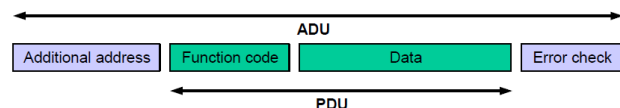


Figura 2. Estrutura geral do quadro Modbus RTU [1]

C. Modbus RTU

O modo de operação *Remote Terminal Unit* (RTU) define como os dados serão empacotados no quadro. Para este tipo de

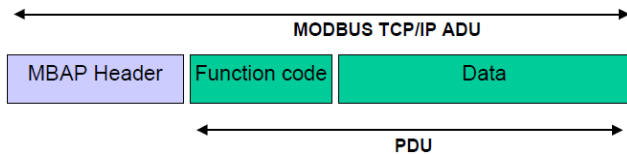


Figura 3. Estrutura geral do quadro Modbus TCP [8]

configuração são estabelecidos alguns parâmetros como: *baud rate*, *paridade*, *data bits* e *stop bits*. Desta maneira todos os equipamentos Modbus em uma rede devem estar configurados com o mesmo modo de operação. Neste modo, a arquitetura empregada é mestre/escravo [9]. A estrutura básica do quadro das mensagens do modo de operação RTU é ilustrada na Fig. 4 e cada campo do quadro será explanado posteriormente.

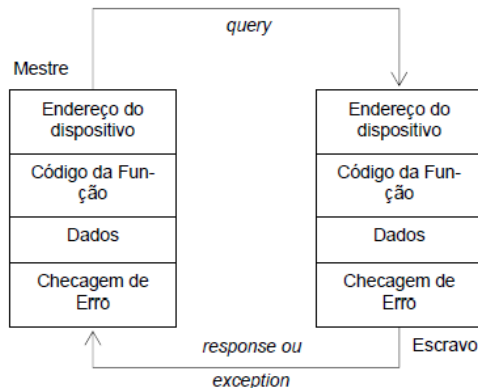


Figura 4. Modelo de mensagem Modbus RTU [7]

O primeiro campo no quadro do padrão Modbus RTU é o endereço do escravo, no qual é composto por um único byte, sendo que os escravos podem utilizar o endereçamento de 1 a 127. Os endereços de 248 a 255 são reservados, e para enviar uma mensagem de *broadcast* emprega-se o endereço 0. O segundo campo é denominado de função, representado por um único byte, que traduz a ação solicitada pelo mestre. O terceiro campo compreende os dados da mensagem, que varia de tamanho de acordo com a função requerida (máximo de 252 bytes), abrange a informação qual o escravo necessita para executar a função, e se tratando de um quadro de resposta, este campo irá conter os dados requeridos. O quarto campo representa a checagem de erro, possui dois bytes, e é responsável por assegurar a integridade da mensagem. Utiliza-se uma verificação de redundância cíclica, calculando o CRC-16 dos bytes do quadro [6].

Um exemplo de requisição de um mestre no modo de operação RTU é ilustrado na Tabela II, sendo requisitada a escrita de uma única saída digital. O campo denominado dados apresenta uma composição de 2 bytes para o endereço da saída requerido e 2 bytes para a ação atribuída. A ação possui os valores: 0xFF00 para ligar a bobina e 0x0000 para desligar a bobina. Para cada função Modbus, o campo denominado dados pode ter um significado diferente.

Se a função for executada corretamente, o escravo respon-

Tabela II
REQUISIÇÃO DO MESTRE MODBUS RTU

Endereço	Função	Dados	CRC-16
01	05	00 01 FF 00	DD FA

derá com um quadro idêntico ao recebido. Caso a requisição possuir uma função, endereço de registrador ou dado inválido, o escravo responde com um código de exceção. Tomando o exemplo do quadro da Tabela II, caso o endereço de saída 0x0001 não for um endereço válido, a resposta do escravo será aquela apresentada na Tabela III.

Tabela III
RESPOSTA DO ESCRAVO A UMA EXCEÇÃO

Endereço	Função	Código do erro	CRC-16
01	85	02	C3 51

Para indicar o início e fim de cada requisição, é utilizado um intervalo de silêncio (*silent*) que equivale ao tempo de transmissão de 3,5 bytes na codificação escolhida [9]. Por exemplo, supondo uma taxa de transmissão configurada de 9600 bps e a codificação de 11 bits para um byte, exemplificando um arranjo com: 1 *Start bit*, 8 *data bits*, sem paridade e 2 *stop bits*. O tempo do byte dar-se-á pela Eq. 1; já o tempo de início/termino é dado pela Eq. 2.

$$T_{byte} = \frac{11}{9600} \approx 1,14 \text{ ms} \quad (1)$$

$$Silent = (3,5 \times 1,14) \approx 4 \text{ ms} \quad (2)$$

Na Tabela IV é ilustrado o modelo OSI do padrão Modbus RTU. Na imagem verifica-se que as camadas: apresentação, sessão, transporte e rede, estão denominadas como “vazias”, a camada de aplicação é representada pelo padrão Modbus, a camada de enlace é representada pela arquitetura mestre/escravo e na camada física identifica-se o meio físico EIA-485, que será utilizado no projeto do *gateway* como meio físico para a transmissão de dados do padrão Modbus RTU [10].

Tabela IV
MODELO OSI- PROTOCOLO MODBUS RTU [10]

Camada	Modelo OSI	Protocolo Modbus RTU
7	Aplicação	Protocolo Modbus aplicação
6	Apresentação	Vazio
5	Sessão	Vazio
4	Transporte	Vazio
3	Rede	Vazio
2	Enlace	Mestre/Escravo
1	Física	EIA-485

1) *Meio físico EIA-485*: O padrão EIA-485 provém uma comunicação serial confiável, pois utiliza uma transmissão por dois fios. As características do padrão são: distância máxima de até 1.200 m, taxa de transmissão de dados de até 10 Mbps, conexão de até 32 dispositivos na rede, tensão diferencial em suas transmissões e imunidade a ruídos. Os dois terminais para

a comunicação são referidos como: A (D+, TxA, Tx +) e B (D-, TxB, Tx -). O sinal binário “1” é reconhecido quando o terminal A está mais positivo que o terminal B. O sinal binário “0” caracteriza-se quando o terminal A está mais negativo em comparação ao terminal B. Por meio de uma tensão diferencial mínima de 200 mV os dispositivos são capazes de reconhecer o sinal do bit [6].

Na rede EIA-485 é necessário o uso de resistores de terminação, tipicamente 120 Ω, a fim de reduzir reflexões que podem alterar os dados transportados. Estes resistores são inseridos nas extremidades da rede. Para evitar perda de dados pode-se utilizar filtros capacitivos ou indutores de modo comum na rede [11].

A fim de manter a tensão diferencial mínima entre os fios, quando a rede não está transmitindo dados, é necessário a utilização de resistores diferenciais (*Fail-safe bias*). Estes são inseridos em algum ponto da rede. O valor dos resistores depende da carga presente na rede [11]. Na Fig. 5 é ilustrada uma rede Modbus RTU contendo o meio físico EIA-485, com os resistores de (*Fail-safe bias*) denominados de *Pull up* e *Pull down* e os resistores de terminação.

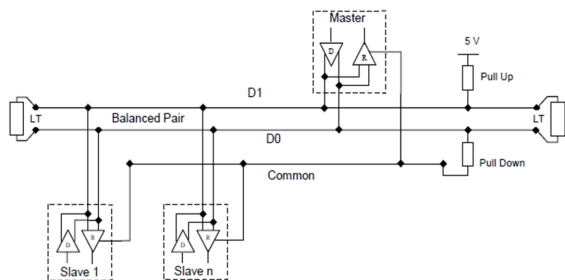


Figura 5. Ligação da rede EIA-485 [10]

D. Modbus TCP

O protocolo Modbus TCP emprega o *Transmission Control Protocol* (TCP) como protocolo de transporte e as redes físicas Ethernet ou Wi-fi para o transporte dos dados. Incorpora o quadro PDU, com o incremento do cabeçalho MBAP, o qual será analisado a seguir. O padrão Modbus TCP emprega a arquitetura cliente/servidor. A porta padrão reservada para o protocolo é a 502 [12].

O cabeçalho MBAP possui um tamanho de 7 bytes, dos quais são divididos entre: *Transaction Identifier* (2 bytes) – refere-se ao número da identificação da requisição, *Protocol Identifier* (2 bytes) – para o padrão Modbus o valor é sempre 0, e os demais valores são reservados para futuras extensões, *Length* (2 bytes) – este campo contém a contagem de bytes restantes da mensagem, *Unit Identifier* (1 byte) – usado para reconhecer o endereço do escravo oriundo de uma rede Modbus RTU [8].

Um exemplo de quadro Modbus TCP é ilustrado na Tabela V, no qual os primeiros dois bytes referem-se ao *Transaction Identifier*. O valor do *transaction ID* é incrementado a cada requisição. Na primeira requisição seu valor é 0x0000, já na

segunda requisição será 0x0001, e assim por diante. O restante do cabeçalho MBAP segue a descrição que foi apresentada anteriormente. A requisição do cliente para este exemplo será a de leitura de registradores internos. O campo denominado dados da requisição conterà: 2 bytes para o endereço do registrador inicial a ser lido e 2 bytes para o número de registradores a serem lidos.

Tabela V
REQUISIÇÃO DO CLIENTE MODBUS TCP

Trans. ID	Prot. ID	Length	Unit ID	Função	Dados
00 00	00 00	00 06	10	03	00 02 00 01

Caso a função for executada corretamente a resposta do servidor, ilustrada na Tabela VI, conterà com o devido cabeçalho MBAP e os campos correspondentes à identificação do valor requisitado.

Tabela VI
RESPOSTA DO SERVIDOR

Cabeçalho MBAP	Função	Bytes de dados	Dados regs.
00 00 00 00 00 05 10	03	02	03 E8

Na Tabela VII é ilustrado o modelo OSI, elencando todas as camadas necessárias para a implementação do protocolo Modbus TCP. No projeto do *gateway* será utilizado, como meio físico para transmissão de dados do modo de operação TCP o padrão *wireless* IEEE 802.11.

Tabela VII
MODELO OSI- PROTOCOLO MODBUS TCP [12]

Camada	Modelo OSI	Protocolo Modbus TCP
7	Aplicação	Protocolo Modbus aplicação
6	Apresentação	Vazio
5	Sessão	Vazio
4	Transporte	Protocolo TCP
3	Rede	Protocolo IP
2	Enlace	Protocolo 802.11
1	Física	Protocolo 802.11 (Wi-fi)

1) *Padrão IEEE 802.11*: O modelo IEEE 802.11 (tecnologia *wireless*) abrange duas camadas do modelo OSI: camada de enlace – a qual faz a transferência de dados para a camada posterior, e está subdividida em: *Logical Link Control* (LLC) e *Media Access Control* (MAC), e a camada física (PHY). Os sinais são transmitidos por meio de radiofrequência. As primeiras taxas de transmissão de dados utilizadas neste padrão encontravam-se na faixa de 1 ou 2 Mb/s e a frequência de operação era de 2,4 GHz [13].

A norma IEEE 802.11 está em um constante processo de melhoria com velocidade de transmissão e frequência mais elevadas e mecanismos de comunicação diferenciados, como o modelo IEEE 802.11b, que possui uma taxa de até 11 Mb/s, frequência de 2,4 GHz, e estabelece o modelo *high-rate DSSS* (HR-DSSS) [14]. Uma das últimas atualizações da norma, a qual foi denominada de IEEE 802.11n pode alcançar uma velocidade de transferência de até 600 Mb/s e opera

nas frequências de 2,4 GHz e 5 GHz [15]. Os métodos de segurança tipicamente utilizados são: *Wired Equivalent Privacy* (WEP), *Wi-Fi Protected Access* (WPA) e *Wi-Fi protected Access 2* (WPA 2) [16].

E. Inversor de frequência

O inversor de frequência é um dispositivo que tem por principal finalidade controlar a velocidade de motores de indução. Realiza o ajuste da frequência do motor por meio de uma técnica denominada de *Pulse Width Modulation* (PWM). Na maioria das vezes, contém uma IHM disposta de uma *display* e um teclado, permitindo ao usuário a visualização ou alteração de parâmetros. Em alguns inversores é possível realizar uma comunicação Modbus RTU via interface EIA-485. O módulo de comunicação pode vir incorporado, ou ser adicionado ao inversor [17].

F. Supervisório SCADA

Os supervisórios *Supervisory Control And Data Acquisition* (SCADA) propiciam que sejam monitoradas informações de processos provenientes de máquinas industriais, sensores, controladores automáticos e outros. A fim de apresentar as informações ao operador, os supervisórios proporcionam uma interface amigável. Por meio de drivers, o SCADA estabelece uma comunicação com os equipamentos. Através de uma IHM diversos recursos gráficos como botões, ícones e *displays* são disponibilizados ao operador [18]. Dois exemplos de *softwares* supervisórios SCADA são: ScadaBR e Elipse E3 [19].

G. Trabalhos relacionados

Para identificar possíveis contribuições e compreender o estado da arte do desenvolvimento de *gateways* Modbus, foram revisados alguns artigos. O artigo [20] aborda uma solução industrial a qual desenvolveu-se um *gateway* entre uma rede com o protocolo Modbus RTU e uma rede sem fio com o protocolo IEEE 802.15.4. Em parceria com a empresa Novus, o autor desenvolveu o *hardware* e toda configuração necessária para criar o *gateway*, e no ano de 2011, o produto AirGate-Modbus foi lançado no mercado pela empresa Novus.

O trabalho [21] desenvolveu um *gateway* entre o protocolo Modbus TCP e o padrão *Zigbee* (rede sem fio), possuindo como características: o baixo custo, baixo consumo de energia e baixa taxa de transmissão, possibilitando implementações industriais, além de aplicações em automação residencial. No desenvolvimento do trabalho foi elaborada toda a arquitetura para a troca de informação dos diferentes protocolos. Desta maneira, partindo do ponto de vista da rede Modbus, o *gateway* atua como um servidor Modbus que disponibiliza dados para um cliente Modbus, já pelo ponto de vista da interface *Zigbee*, o *gateway* atua como um coordenador e possui as funções de criar a rede, gerir o acesso e de atualizar os dados presentes nos nós da rede.

O artigo [22] propõe o desenvolvimento de um *gateway* que realiza a conversão dos protocolos Profibus-DP e Modbus. Desta forma estes dois protocolos, os quais são amplamente aplicados no ambiente industrial, podem ser interconectados

por um custo relativamente baixo, promovendo assim uma flexibilidade de comunicação entre os diversos dispositivos utilizados. Para o desenvolvimento do *hardware* do projeto, foi necessário a utilização de *chips* que possuem funções de incorporação e tradução dos protocolos, além de um micro-controlador principal.

H. Gateways comerciais

Com a intenção de agregar informações técnicas ao trabalho, alguns *gateways* comerciais foram pesquisados. Os três *gateways* identificados são apresentados a seguir.

O primeiro *gateway* analisado é o ETH 485, desenvolvido pela empresa Tecnologia Engenharia e Representações Técnicas, o qual tem função primária a integração de dispositivos conectados em redes EIA-485 via Modbus RTU com sistemas supervisórios Modbus TCP. O equipamento permite a comunicação entre o sistema de supervisão e até duas redes EIA-485 de campo. Outro modo de operação possível é através de 3 conexões simultâneas de mestres Modbus TCP. Desta maneira, cada porta TCP pode ser ajustada para uma das redes EIA-485. Outra arquitetura possível é denominada de *bridge*. Neste modelo é utilizado uma porta da rede EIA-485 como um mestre, operando concomitantemente com outros mestres Modbus TCP e a outra porta EIA-485 é destinada aos escravos da rede. As configurações dos parâmetros são ajustadas por meio de uma interface Ethernet, utilizando um navegador Web [23].

O segundo *gateway* examinado é o HF2211, desenvolvido pela empresa FlexPort, o qual fornece uma interligação entre os padrões seriais: EIA-485/EIA-422/EIA-232 e os protocolos Ethernet/Wi-fi. Suporta diversos modos de operação e destaca-se a possibilidade de comunicação entre o protocolo Modbus TCP para Modbus RTU e também do Modbus RTU para o Modbus TCP. Alguns componentes de *hardware* observados e considerados relevantes são: uma antena Wi-fi, um botão de *reset* (para o dispositivo voltar ao padrão de fábrica), 3 LEDs sinalizadores, sendo que o primeiro LED tem o papel de indicar se o *gateway* está energizado, o segundo de sinalizar caso a conexão Ethernet ou Wi-fi está em funcionamento e por fim o último LED indica se há alguma transferência de dados. As configurações dos parâmetros são realizadas via navegador Web [24].

O terceiro *gateway* pesquisado é o WLg-IDA/S, desenvolvido pela *Acksys Communication & Systems*, que tem por finalidade conectar os equipamentos que utilizam o padrão Modbus serial com a rede Modbus TCP industrial, por meio de uma conexão *wireless*. Possui diversas arquiteturas de operação, como exemplo a topologia de multi-mestre e diversos modos de operação como: Modbus (RTU ou ASCII) para Modbus TCP. Os componentes de *hardware* se equiparam ao *gateway* descrito no anteriormente, contendo: uma antena Wi-fi e LEDs para indicações [25].

III. PROPOSTA DO PROJETO

Nesta seção do TCC é retratada a proposta do projeto, sendo primeiramente apresentado o fluxograma, o qual é fundamental

para entender a tradução entre os protocolos Modbus, e o diagrama de blocos, que ilustra uma visão geral do *gateway*. Posteriormente é apresentado o *hardware* principal, com o detalhamento dos componentes, e por fim, as configurações do *gateway* e os periféricos.

A. Fluxograma e Diagrama de blocos

O *gateway* Modbus proporciona ao cliente, que opera via modo TCP, enviar requisições aos servidores, que operam via modo RTU. Todavia, nesta implementação um cliente RTU não é capaz de solicitar dados a servidores TCP. A Fig. 6 apresenta o fluxograma do algoritmo que foi responsável pela comunicação entre os padrões TCP (cliente) e RTU (servidor). O fluxograma é a base para a elaboração do código, pois são definidas, de forma organizada e sequencial, as etapas da tradução e comunicação dos modos de operação. O algoritmo é executado em um microcontrolador responsável pela tradução das requisições. A transferência física dos dados foi concebida com os módulos EIA-485 e IEEE 802.11 conectados ao microcontrolador. As configurações básicas do *gateway* podem ser modificadas através de um programa executável no computador. Os periféricos auxiliam na comunicação entre o usuário e o dispositivo. A Fig. 7 ilustra o diagrama de blocos da estrutura completa do *gateway*, onde são utilizados os módulos IEEE 802.11 e EIA 485 para a comunicação do microcontrolador com os respectivos dispositivos/sistemas, periféricos para auxiliarem o usuário e o computador (configurações) para o usuário poder alterar parâmetros.

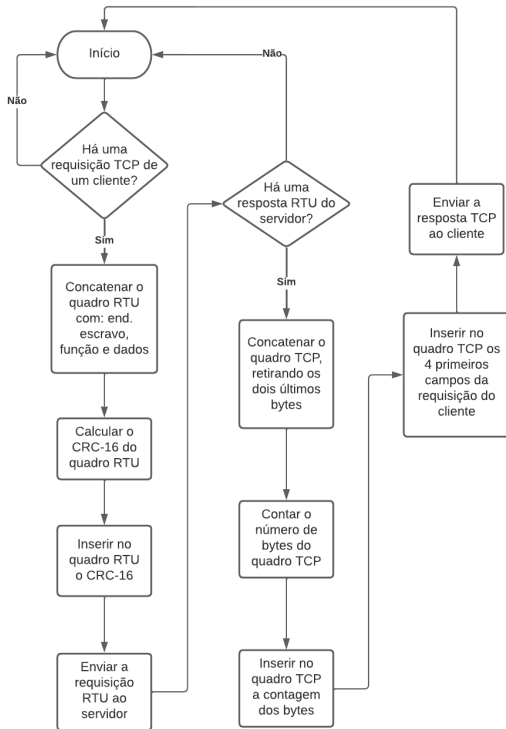


Figura 6. Fluxograma de tradução de quadros RTU e TCP

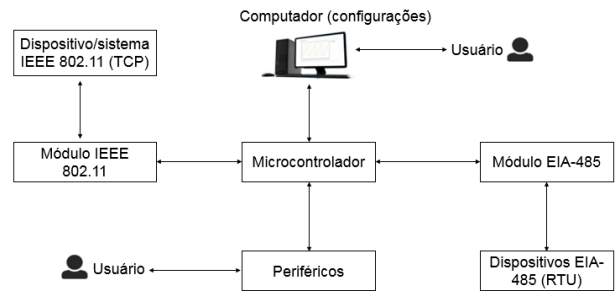


Figura 7. Estrutura geral do *gateway*

B. Hardware principal

Com a finalidade de executar a função principal do *gateway* Modbus RTU-TCP, alguns componentes físicos são de fundamental importância para a tradução de dados entre os protocolos. O controlador central é composto pelo microcontrolador Arduino Mega 2560, o qual estabelece a comunicação via Modbus TCP através do ESP8266-01 e por meio do Módulo Conversor RS485 realiza a comunicação serial via Modbus RTU. Estes componentes serão detalhados nos parágrafos abaixo. Na Fig. 8 é apresentada a ilustração dos três componentes referidos acima. Pode-se visualizar que o ESP8266-01 e o módulo RS485 podem receber e enviar dados do Arduino Mega.

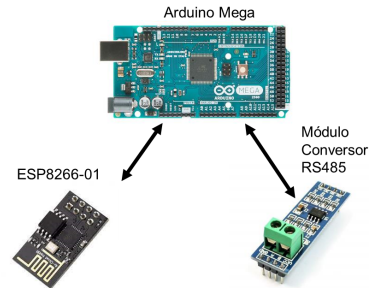


Figura 8. Hardware central

1) *Arduino Mega 2560*: O Arduino Mega 2560 é uma placa baseada no microcontrolador ATmega2560. Este dispositivo é de baixo custo e possui um *software* de código aberto. O meio físico de comunicação entre o Arduino Mega e os outros dois componentes do *hardware* principal, é realizado por meio de pinos seriais Tx e Rx.

O Arduino Mega 2560 possui um cristal oscilador de 16 MHz, conexão via USB, 4 UARTs (portas seriais), memória EEPROM de 4 KB, 54 pinos digitais e tensão de operação de 5 V [26].

2) *ESP8266-01*: O módulo Wi-fi ESP8266-01 é um dispositivo *System on chip* (SOC), sem fio, de alta integração e de baixo custo, projetado para plataformas móveis com restrições de espaço e energia. A sua principal função é de proporcionar, para outras plataformas ou sistemas, comunicação Wi-fi. Su-

porta o padrão 802.11 b/g/n e a pilha completa do protocolo TCP [27].

A tensão de trabalho deste módulo é de 3,3 V, sendo fornecida diretamente por um pino de saída dedicado do Arduino Mega. A fim de realizar a comunicação entre a placa microcontroladora e a placa Wi-fi, um dos pinos Tx do Arduino Mega é ligado ao Rx do módulo Wi-fi, com a necessidade de um incluir um divisor resistivo, com a finalidade de reduzir a tensão que chega no Rx. Neste caso foi fundamental introduzir um circuito com dois resistores de 10 k Ω . Um dos pinos Rx do microcontrolador é ligado diretamente ao Tx do módulo. O pino CH_PD do módulo é conectado aos 3,3 V, com a inserção de um resistor de 10 k Ω para limitar a corrente.

3) *Módulo Conversor RS485*: Este módulo tem por finalidade promover a conexão entre placas microcontroladoras e redes EIA-485. Propicia a comunicação entre diversos equipamentos ou dispositivo que utilizam este meio físico de transferência de dados. O *chip* central deste módulo é o MAX485, que é responsável pela conversão de níveis de tensão da porta. Os pinos encontrados na placa são: RO, DI, RE, DE, VCC, GND, A e B.

A tensão de trabalho desta placa é de 5 V, sendo fornecida pelo Arduino Mega. A fim de realizar a comunicação da placa microcontroladora e a placa RS485, o pino *Receive Output* (RO) é conectado diretamente ao pino Tx do Arduino Mega. O pino *Data Input* (DI) é ligado diretamente ao Rx da placa microcontrolada. O pino *Receive Enable* (RE) e o pino *Data Enable* (DE) são utilizados para o controle da transferência dos dados. Estes podem ser conectados a saídas digitais do microcontrolador.

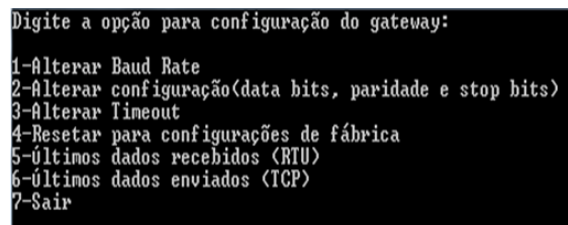
C. Configurações do gateway

Em diversos *gateways* comerciais analisados existe a possibilidade de alteração de parâmetros, como exemplo: o *baud rate*, a paridade, o *stop bit*, *data bits*, o *timeout* e a possibilidade de resetar o dispositivo para os parâmetros preestabelecidos. O dispositivo projetado também possui estas funcionalidades. Porém, diferentemente da maioria dos equipamentos comerciais, os quais usufruem de uma página Web para realizar as configurações, o dispositivo projetado conta com um programa executável no computador, pois não é possível utilizar dois servidores simultâneos no módulo ESP8266-01. Um dos servidores é responsável pela comunicação Modbus TCP, que é fundamental ao projeto, e para criar uma página Web seria necessário mais um servidor. Assim optou-se por um programa executável para alterar os parâmetros. Esse propicia uma navegação através de *menus* e possibilita ao usuário ajustar os parâmetros.

O algoritmo deste programa executável foi implementado no *software* Dev-C++ em linguagem C. A comunicação entre o programa e o Arduino Mega é através da porta USB/serial. Os parâmetros alterados por meio do usuário são armazenados pela memória EEPROM do Arduino Mega. Desta forma quando o microcontrolador é ligado, todos os parâmetros

retidos nessa memória serão recuperados e carregados para o programa principal.

Ao acessar as configurações do *gateway* pode-se verificar os últimos quadros Modbus RTU e TCP, que foram processados na comunicação. A Fig. 9 ilustra a tela principal de configurações do *gateway*, que será detalhada a seguir.



```
Digite a opção para configuração do gateway:
1-Alterar Baud Rate
2-Alterar configuração(data bits, paridade e stop bits)
3-Alterar Timeout
4-Resetar para configurações de fábrica
5-últimos dados recebidos (RTU)
6-últimos dados enviados (TCP)
7-Sair
```

Figura 9. Tela de Configurações do *gateway*

A tela principal possui sete opções para o usuário. Quando selecionada uma destas pode-se alterar parâmetros ou visualizar determinadas informações. A opção de “alterar *baud rate*” possibilita o usuário inserir a velocidade de transmissão serial desejada. Para modificar a configuração serial o usuário é direcionado a *submenus*, onde escolhe: *data bits*, paridade e *stop bits*, estas escolhas já são preestabelecidas pelo programa. A alteração do *timeout* é similar a alteração do *baud rate*, no qual é inserido manualmente o valor pretendido. A opção de “resetar para configurações de fábrica” proporciona que todos os parâmetros citados anteriormente retornem aos valores predefinidos no algoritmo do *gateway*. As opções de “últimos dados recebidos (RTU) e enviados (TCP)”, recebem do *gateway* o último quadro transferido via modo RTU e TCP, e exibe na tela. A opção “sair” encerra o programa.

D. Periféricos

Certos *gateways* comerciais pesquisados fazem uso de periféricos. Estes são utilizados para receber e enviar dados do *gateway*. Os primeiros periféricos retratados são os sinais luminosos de *feedback*, que permitem ao usuário visualizar o *status* do dispositivo. Um LED verde, em seu estado ligado, indica que o *gateway* está energizado. Um LED azul, em seu estado ligado, indica que a rede Wi-fi foi conectada corretamente. Um LED amarelo fica piscando no tempo em que ocorrer uma transferência de dados. Outro periférico inserido é um botão (*push-button*) com a função *reset*, e quando é pressionado o *gateway* retorna ao padrão de fábrica, ou seja, regressa para suas configurações iniciais, que foram definidas no momento da elaboração do algoritmo. O último periférico é o *Display LCD* 16x2, que tem o papel de exibir em sua tela algumas informações que estarão invisíveis ao usuário no momento da utilização do *gateway*, como exemplo, os quadros Modbus, tempos de transferência de quadros, parâmetros como o *baud rate*, paridade, etc. A Fig. 10 ilustra uma das telas do *display LCD* desenvolvida, e as demais telas serão especificadas a seguir.

Para navegar na tela do *Display LCD* foram inseridos quatro botões *push button*, e assim pode-se acessar o parâmetro que



Figura 10. Tela do *Display LCD*

deseja visualizar. Para alterar o tipo de informação na tela principal foram utilizados dois dos botões. Para acessar e navegar nas telas secundárias foram utilizados os outros dois botões. A tela principal e as telas secundárias possuem tais parâmetros:

- *Dados RTU*
 - *Baud Rate*
 - *Timeout*
 - Configuração serial (*data bits*, *paridade* e *stop bits*)
- *Quadro RTU*
 - Endereço do escravo
 - Função
 - Dados
 - CRC-16
- *Quadro TCP*
 - *Trans.ID*
 - *Prot.ID*
 - *Length*
 - *Unit.ID*
 - *Função*
 - *Dados*
- *Tempo transferência*
 - *Tempo (ms)*

IV. IMPLEMENTAÇÃO E RESULTADOS

A. Programação e configurações

Através do *Software Arduino (IDE)* foram elaborados: o código principal, responsável pela comunicação e tradução dos modos de operação, o algoritmo para enviar os dados ao *display LCD* e aos periféricos, os comandos para o fluxo de dados entre o *Arduino* e o programa executável, que tem por finalidade principal alterar as configurações do *gateway* e o código responsável pelo armazenamento (leitura e escrita) dos parâmetros seriais através da memória *EEPROM*.

Foi desenvolvido o código do programa executável no *Dev-C++*, que tem por finalidade principal auxiliar o usuário a modificar os parâmetros seriais. Este programa foi elaborado com um *menu* principal e uma estrutura que recebe e envia os dados ao *Arduino Mega*. O supervisor utilizado para a validação do servidor *TCP* também exigiu a escrita de códigos *HyperText Markup Language (HTML)* em determinados ícones criados nas telas desenvolvidas. O *timeout*, tanto de envio quanto de recebimento de dados, deve ser definido de acordo com os tempos estabelecidos na norma ou de acordo com a aplicação.

B. Resolução de problemas

No decorrer dos testes preliminares do *gateway Modbus* ocorreram algumas adversidades. A primeira ocorreu durante a

comunicação dos quadros, através do módulo *RS485*, os quais continham inconsistências, por exemplo, o errôneo acréscimo de bytes (0x00) no início do quadro. Para corrigir este problema, foi inserido um resistor de 1 k Ω entre o *GND* e pino *B* do módulo *RS485*. Desta forma, foi reduzido o valor do resistor de (*Fail-safe bias*) oriundo do módulo.

O segundo problema foi identificado no momento da elaboração do código do *gateway*, pois verificou-se que a função do *LED* amarelo, que deveria piscar no momento de uma transferência de dados, não poderia ser implementada por meio do algoritmo, pois não pode-se interromper o trecho da transferência de dados para executar a função de piscar o *LED*. Assim para resolver esta adversidade, foi inserido um circuito com um *CI 555*, que quando energizado faz o *LED* piscar. A Fig. 11 ilustra o circuito elaborado para tal finalidade, com os valores de resistores e capacitores calculados para obter-se uma frequência adequada para o *LED* piscar.

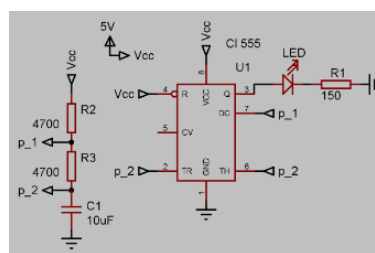


Figura 11. Circuito *CI 555*

A terceira adversidade deve-se ao fato que o *display LCD 16x2* possui uma limitação de caracteres. Desta maneira, o campo “*Dados*” do quadro *Modbus*, em certos comandos, necessita de um número maior de caracteres. A fim de solucionar esta questão, foi desenvolvido no *menu*, executado com a finalidade de configurações do *gateway*, a opção de acessar os últimos quadros *RTU* e *TCP*, como visualizado na Fig. 9.

C. Supervisor ScadaBR

Para teste da função de cliente *TCP* foi preferido o *software* supervisor *ScadaBR*. Primeiramente, foi efetuado a configuração de parâmetros para acesso aos dados do *gateway* como: a porta *TCP*, o endereço *IP*, *timeout*, etc. Em seguida, foi desenvolvido a representação gráfica com ícones de escrita e leitura. A tela inicial contém botões e campos de leituras relativos ao sistema de testagem, e a tela “*inversor*” refere-se a estrutura criada para o sistema de verificação. Os dois sistemas mencionados acima serão retratados posteriormente. A Fig. 12 ilustra a tela inicial e a Fig 13 demonstra a tela denominada “*inversor*” elaborada. A tela inicial exhibe quatro funções *Modbus* que foram escolhidas para realizarem algumas operações de escrita ou de leitura no sistema de testagem desenvolvido. A tela “*inversor*” possui um ícone que refere-se ao comando de liga/desliga do *inversor*.

D. Sistema de testagem

Com o intuito de realizar os primeiros testes e verificações do *gateway Modbus*, foi utilizado um *Arduino Mega 2560*



Figura 12. Tela inicial ScadaBr



Figura 13. Tela inversor ScadaBr

como servidor (RTU) e o supervisório *open source* ScadaBR como cliente (TCP). Os primeiros testes desenvolvidos enviaram a função Modbus 0x05 (*write single coil*). A fim de validar o funcionamento adequado do comando, foi inserido um LED em um pino de saída do servidor. Desta maneira, ao clicar o botão ligar da tela do ScadaBR o LED deveria acender, e ao clicar em desligar o LED deveria apagar. Esta funcionalidade foi testada com sucesso.

Por meio da porta serial USB do Arduino ligada à IDE foi possível debugar e verificar os quadros enviados e recebidos. Os quadros Modbus foram impressos, por meio do monitor serial, em determinadas linhas do código. Os quadros que podem ser visualizados pelo monitor situam-se em partes importantes do código como: quando chega uma requisição do cliente, a cada alteração realizada no quadro TCP para RTU, no envio do quadro ao servidor, quando chega uma resposta do servidor, a cada alteração do quadro RTU para TCP e no envio da resposta ao cliente. Na Fig. 14 é ilustrado a tela do monitor serial do *gateway*, podendo ser visualizado um quadro Modbus de uma requisição do cliente (TCP), com o devido cabeçalho MBAP, a função (0x05) escrita de única saída e a ação (0xFF 00) de ligar a saída.

O segundo teste desenvolvido foi o envio de um quadro TCP com a mesma função Modbus 0x05, porém configurado um endereço de saída não aceito pelo servidor, assim este não o reconheceria e responderia com uma mensagem de exceção. A detecção deste erro pelo cliente foi visualizada por uma mensagem de alarme. O terceiro teste foi o de

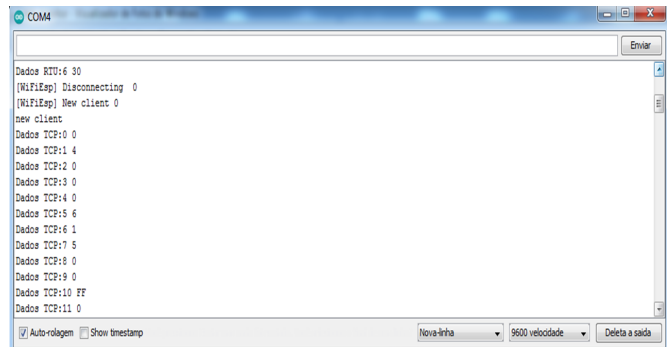


Figura 14. Tela monitor serial

enviar um quadro com a função Modbus 0x02 (*read discrete inputs*) e o servidor foi configurado para responder sempre como a entrada estivesse em nível alto. Já no quarto teste o cliente enviou a função Modbus 0x03 (*read holding register*), onde o servidor respondeu conforme o nível de tensão de um potenciômetro. Assim foi testado um sinal analógico através de um potenciômetro. A Fig. 15 retrata o sistema de testagem desenvolvido. Nesta pode-se visualizar o fluxo de dados que ocorre desde o supervisório até o servidor (RTU) e todos os componentes que envolvem este sistema. Para a comunicação entre o supervisório e o ESP8266-01 e utilizado um roteador Wi-fi. Os demais componentes são conectados através de cabos e fios.

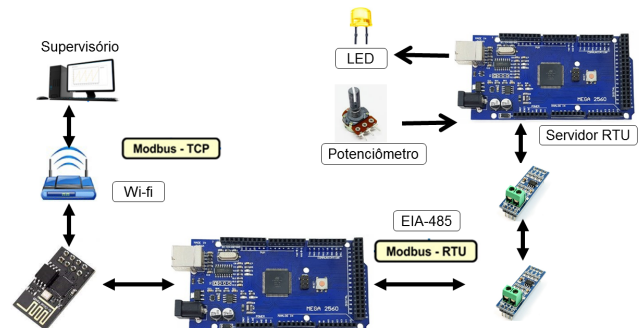


Figura 15. Sistema de testagem

A montagem física do protótipo foi elaborada inserindo e conectando os componentes em duas placas *protoboards*. O uso de duas placas foi necessária na conexão dos módulos RS485 e para dividir a estrutura referente ao *gateway* com a estrutura do servidor. Na Fig. 16 é ilustrado o protótipo do *gateway* Modbus RTU-TCP desenvolvido.

E. Sistema de verificação

A fim de realizar a validação principal, foi elaborado uma comunicação entre o supervisório ScadaBR e o inversor de frequência Delta VFD-E. O supervisório, que possui a função de cliente, envia uma requisição via Modbus TCP, e esta requisição será traduzida para o padrão Modbus RTU e após

Tabela IX
Quadro RTU - Desligar Motor

Endereço	Função	Dados	CRC-16
04	06	20 00 00 11	42 53

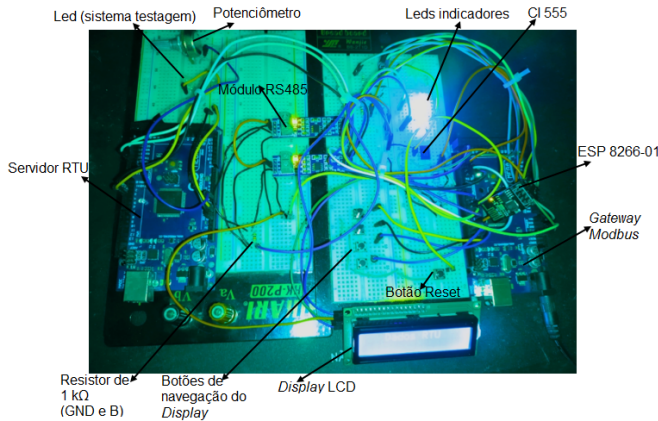


Figura 16. Protótipo do gateway Modbus RTU-TCP

enviada para o inversor de frequência. Os parâmetros alterados no inversor Delta, por meio do ScadaBR, foram de ligar e desligar o motor. Na Fig. 17 é ilustrado o sistema de validação proposto, nesta observa-se o fluxo de dados desde o supervisório até o inversor de frequência, que basicamente segue o mesmo princípio do sistema de testagem, porém a função de servidor é desempenhada pelo inversor.

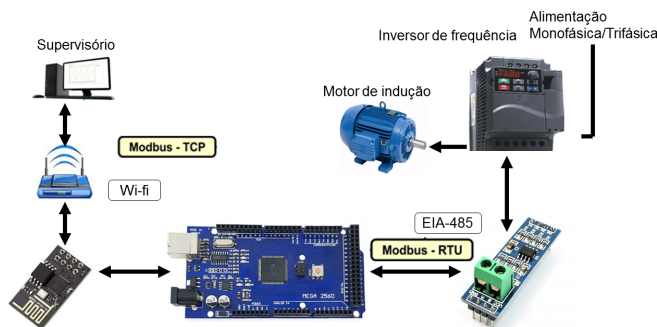


Figura 17. Sistema de validação principal

Com base nas informações de [28], foram definidos os quadros Modbus RTU que possuem as funções de ligar e desligar o motor via inversor de frequência. Na Tabela VIII é ilustrado o quadro para acionar o motor e na Tabela IX é apresentado o quadro que desliga o motor. O entendimento destes quadros foi a base para a elaboração das requisições do cliente TCP por meio do ScadaBR para tais comandos, que serão convertida para RTU pelo gateway e enviadas ao inversor de frequência.

Tabela VIII
Quadro RTU - Ligar Motor

Endereço	Função	Dados	CRC-16
04	06	20 00 00 12	02 52

Com as informações que foram apresentadas anteriormente, foram configurados no *software* ScadaBR: endereço de escravo, função e endereço de saída. Na tela “inversor” foi inserido um *data point* simples, para escrever o valor de saída a ser enviado. Assim o campo referido como “Dados”, terá o endereço de saída 0x2000, que não pode ser alterado na representação gráfica, já o valor de saída pode ser alterado pelo usuário. A fim de ligar o motor foi inserido o decimal 18, que é convertido pelo gateway para o hexadecimal 0x12. Para desligar o motor foi inserido o decimal 17, sendo convertido para o hexadecimal 0x11. A representação gráfica desta tela pode ser visualizada na Fig. 13.

V. CONCLUSÕES

O trabalho retratado neste TCC teve como objetivo apresentar um gateway Modbus funcional, capaz de realizar a troca de dados entre os padrões Modbus RTU e Modbus TCP usando plataformas e componentes *open source*. A integração vertical, que é conceituada na Indústria 4.0, tem relação com o projeto, pois o gateway possibilita a intercomunicação entre os níveis hierárquicos da pirâmide da automação.

O projeto teve como ponto de partida a revisão bibliográfica, a qual propiciou um conhecimento dos quadros Modbus e dos recursos auxiliares, que foram agregados ao protótipo final. Com o conhecimentos dos quadros Modbus foi elaborado um fluxograma, sendo a base para a criação do algoritmo, que foi transcrito ao Arduino *Software* (IDE) e posteriormente carregado à placa microcontrolada Arduino Mega 2560. Foram conectados à placa dois componentes essenciais: o módulo conversor RS485 e o ESP8266-01. Estes tiveram o papel de realizar a comunicação RTU e TCP, respectivamente.

O sistema de testagem elaborado, incorporando um Arduino Mega 2560 como servidor (RTU), mostrou-se importante para o projeto, pois foi com este que um dos problemas relevantes ao projeto foi detectado, porque no momento da comunicação via RS485, foi possível a identificação do acréscimo de *bytes* errôneos no início dos quadros e esta adversidade foi solucionada com sucesso. Um sistema de verificação foi desenvolvido para a validação do gateway Modbus, utilizando um dispositivo comercial como servidor RTU. O inversor de frequência Delta VFD-E foi o escolhido para esta atribuição, desta forma foi enviado comandos de ligar e desligar o motor. O gateway mostrou-se capaz de realizar a comunicação com os modos TCP e RTU. O cliente (TCP) utilizado para ambos os sistemas descritos acima foi o supervisório *open source* ScadaBR.

Para trabalhos futuros pode-se realizar *upgrades* nas funcionalidades do gateway como: agregar o modo de operação ASCII na comunicação, a possibilidade do cliente RTU solicitar dados aos servidores TCP e inserir outros meios físicos no

transporte dos dados como o EIA-232. Este projeto pode servir de base ou motivação para outros trabalhos, os quais podem optar por outros protocolos. Portanto, mediante o presente estudo, conclui-se que é possível utilizar componentes *open source* para o desenvolvimento de uma solução aberta que pode ser utilizada e desenvolvida como produto.

REFERÊNCIAS

- [1] Modbus Organization. *Modbus application protocol specification v1.1b3*. Disponível em: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf, 2012. [Online; acesso em 06-Novembro-2021].
- [2] J. A. Peixoto. Internet das coisas na manufatura industrial: uma ferramenta da indústria 4.0 para interação dos processos. Disponível em: <https://redeindustria40.com.br/>, 2021. [Online; acesso em 11-Maio-2022].
- [3] C. G. Gonçalves et al. Indústria 4.0 - Integração de Sistema. *Pesquisa e Ação*, 2019.
- [4] J. F. Kurose and K. W. Ross. *Redes de computadores e a Internet: uma abordagem top-down*. Pearson, 6 edition, 2013.
- [5] A. S. Tanenbaum and D. J. Wetherall. *Redes de computadores*. Pearson, 5 edition, 2011.
- [6] S. Mackay, E. Wright, D. Reynders, and J. Park. *Practical Industrial Data Networks: Design, Installation and Troubleshooting*. Elsevier, 2004.
- [7] J. M. A. do Nascimento and P. B. de Lucena. Protocolo MODBUS. *LECA-DCA-UFRN*, pages 1–4, 2003.
- [8] Modbus Organization. *Modbus messaging on TCP/IP implementation guide v1.0b*. Disponível em: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf, 2006. [Online; acesso em 06-Novembro-2021].
- [9] Alfa Instrumentos. Protocolo de comunicação modbus rtu / ascii. Disponível em: https://www.dca.ufrn.br/~affonso/FTP/DCA447/modbus/modbus_manual.pdf, 2000. [Online; acesso em 11-Novembro-2021].
- [10] Modbus Organization. *Modbus over Serial Line Specification and Implementation Guide V1.02*. Disponível em: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf, 2006. [Online; acesso em 06-Novembro-2021].
- [11] B. Perrin. The Art and Science of RS-485. *Circuit Cellar*, 1999.
- [12] M. R. Caldieri. Implementação Do Modbus Para Aplicação Em Sistema De Controle Via Rede Sem Fio. Master's thesis, Universidade Estadual Paulista "Júlio de Mesquita Filho", 2016.
- [13] B. Fleck and B. Potter. *802.11 Security*. O'Reilly, 2002.
- [14] M. S. Gast. *802.11 Wireless Networks The definitive Guide*. O'Reilly, 2002.
- [15] L. M. F. Duarte. Sistemas e Tecnologias de Informação para as Organizações. Master's thesis, Escola Superior de Tecnologia e Gestão de Viseu, 2020.
- [16] B. I. Reddy and V. Srikanth. Review on wireless security protocols (WEP, WPA, WPA2 & WPA3). *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 28–35, 2019.
- [17] J. M. Mascheroni, M. Lichtblau, and D. Gerardi. Guia de Aplicação de Inversores de Frequência. *Weg*, 2:196, 2004.
- [18] Fundação Centros de Referência em Tecnologias Inovadoras. *Manual do software SCADABR*. Disponível em: <https://scadabr.com.br/index.php/2017/06/06/2939/>, 2010. [Online; acesso em 11-Novembro-2021].
- [19] M. H. M. Faria et al. Estudo comparativo entre ferramentas de supervisão, controle e aquisição de dados e a importância destas para o ensino em engenharia. *XLI Congresso Brasileiro de Educação em Engenharia. Centro Federal de Educação Tecnológica de Minas Gerais*, page 11, 2012.
- [20] G. B. M. Guarese. Arquitetura híbrida de comunicação para ambientes de automação industrial: protocolos IEEE 802.15.4 e Modbus RTU sobre RS485. *Revista da Graduação (PUCRS)*, 2012.
- [21] P. M. C. Pereira. Gateway ZigBee - Modbus / TCP. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2009.
- [22] H. Zhang, Y. Li, and H. Zhu. Development for protocol conversion gateway of Profibus and Modbus. *Procedia Engineering*, 2011.
- [23] Tecnologia Engenharia e Representações. Eth 485 - manual de serviço. Disponível em: <https://www.tecnolog.ind.br/conversores-de-rede/201-conversor-gateway-ethernet-modbus-tcp-para-rtu-eth485.html#downloads>. [Online; acesso em 28-Outubro-2021].
- [24] FlexPort. HF 2211 - Serial Server Device User Manual. Disponível em: <https://flexport.com.br/Produtos/33/241/conversor-modbus-ethernet-wi-fi-para-serial-rs232422485.html>, 2017. [Online; acesso em 28-Outubro-2021].
- [25] Acksys Communication & Systems. Wireless rs232 / rs422 / rs485 serial device server for factory automation. Disponível em: <https://www.acksys.fr/en/product/11-wlg-idas/>. [Online; acesso em 28-Outubro-2021].
- [26] Arduino. Arduino mega 2560 rev3. Disponível em: <https://store-usa.arduino.cc/products/arduino-mega-2560-rev3>, 2021. [Online; acesso em 19-Outubro-2021].
- [27] AI-Thinker Team. *ESP-01 WiFi Module Version 1.0*. Disponível em: <http://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>, 2015. [Online; acesso em 06-Novembro-2021].
- [28] Delta Electronics. *VFD-E User Manual*. Disponível em: https://www.galco.com/techdoc/dlpc/vfd055e23a_um.pdf, 2009. [Online; acesso em 11-Maio-2022].